SafeSpection – A Systematic Customization Approach for Software Hazard Identification

Christian Denger, Mario Trapp, and Peter Liggesmeyer

Fraunhofer Institute Experimental Software Engineering, Fraunhofer-Platz 1,
67663 Kaiserslautern, Germany
{Christian.Denger,Mario.Trapp,
Peter.Liggesmeyer}@iese.fraunhofer.de

Abstract. Software is an integral part of many technical systems and responsible for the realization of safety-critical features contained therein. Consequently, software has to be carefully considered in safety analysis efforts to ensure that it does not cause any system hazards. Safety engineering approaches borrowed from systems engineering, like Failure Mode and Effect Analysis, Fault Tree Analysis, or Hazard and Operability Studies, have been applied on software-intensive systems. However, in order to be successful, tailoring is needed to the characteristics of software and the concrete application context. Furthermore, due to the manual and expert-dependent nature of these techniques, the results are often not repeatable and address mainly syntactic issues. This paper presents the concepts of a customization framework to support the definition and implementation of project-specific software hazard identification approaches. The key-concepts of the approach, generic guide-phrases, and tailoring concepts to create objective, project-specific support to detect safety-weaknesses of software-intensive systems are introduced.

Keywords: Software Safety, Guide-Phrases, SafeSpection, Software FMEA, Software FTA, Software HAZOP.

1 Introduction

Over the last decades, embedded systems have become an integral part of our daily lives. Especially in the automotive domain, software-intensive systems execute and control a variety of functions and safety measures. Without software, many innovative functions and features would be hard or even impossible to realize.

As a part of a safety-critical system, i.e., a system whose failure might endanger human life, cause extensive environmental damage, or lead to substantial economic loss [1], the software itself must be perceived as safety-critical. In other words, as part of the system the software has the potential of putting the overall system into a hazardous situation. In that sense, Leveson defines safety critical software as any software that can contribute to the occurrence of a hazardous system state either directly or indirectly [2]. As an example of the safety criticality of software, General Motors had to recall almost one million cars due to problems with their airbag system. On paved roads under normal conditions the software interpreted the unstable movement

M.D. Harrison and M.-A. Sujan (Eds.): SAFECOMP 2008, LNCS 5219, pp. 44–57, 2008.
© Springer-Verlag Berlin Heidelberg 2008

of the cars as a crash and activated the airbag. Similar examples can be found that demonstrate the importance of including software in system safety analysis activities executed during the development life-cycle.

Hence, in the automotive domain, recent standards (e.g., IEC 61508, ISO/WD 26262, MISRA Safety Analysis Guide) request a thorough software safety analysis. The standards require the application of safety engineering techniques on the functional concept and on the software architecture [3], [4]. In order to fulfill this, companies typically apply safety analysis techniques like Failure Mode and Effect Analysis (FMEA), Hazard and Operability Studies (HAZOP), and Fault Tree Analyses (FTA) to identify potential system hazards caused by the software. However, the applied techniques are often not customized to the characteristics of software and consequently do not support the identification of conceptual software faults. Mainly, the standard processes of system-FMEA, -FTA, and -HAZOP techniques are used to analyze the software work products. These processes are of a manual nature without concrete guidance on how to identify software faults. The results rely on the experience of the moderator and the participating experts and hence the analysis is not repeatable, subjective and results cannot be compared between different development teams. During the last decades, some approaches have evolved on how safety analysis techniques can be applied to software. As Section 2 demonstrates, the efficient application of these techniques still remains unclear. The main reason for this is that many approaches focus the analysts on very detailed, low-level software causes of hazards like uninitialized variables, too late or too early execution of algorithms, and wrong data models. Additionally, the guidance provided by existing approaches is often on a general-purpose level not tailored to the specific context characteristics of a (software-) project. Hence, only general aspects like correctness and completeness issues, and syntactic aspects are captured. Conceptual software faults, i.e., faults in the logic of the software models, are rarely in the scope of existing approaches. In consequence, what is missing today is an approach that provides systematic guidance on how to customize safety-analysis techniques to the characteristics of a software development context. This approach should support the identification of conceptual, semantic software faults that might cause system hazards. The identification should be performed during the early development phases to provide real added value. SafeSpection has been developed to close this gap.

Section 2 provides a detailed overview of the state of the art regarding software safety analyses and motivates the approach. Section 3 introduces the core concept of SafeSpection: guide-phrases and a grammar to systematically derive project-specific guidance on detecting conceptual software faults causing system hazards. Section 4 outlines the results of an initial feasibility study in an industrial setting. Section 5 concludes the paper and provides some future research topics.

2 Existing Software Safety Analysis Approaches

Even though the idea of software safety analysis techniques has been around for several decades, the number of publications regarding this topic remains quite small [6]. The following subsections categorize the existing approaches and provide a critical review of these regarding their repeatability, customizability, and focus.

2.1 A General View on Existing Software Safety Approaches

According to Fenelon et al. [5], software safety analysis approaches are categorized according to the direction of the search for software causes of hazards. Explorative, inductive, deductive, and descriptive approaches are distinguished. This classification is based on the categorization of safety analysis techniques used in systems engineering. In order to provide a more intuitive, software-related classification we rephrased and extended this existing scheme.

On an abstract level software safety analysis approaches are classified according to the underlying systems engineering techniques they are based upon. Thus, *HAZOP-like approaches*, *FMEA-like approaches*, *Inspection-like approaches,* and *Formal-approaches* are distinguished indicating that the identification of software causes of hazards is based on standard FMEA, HAZOP, inspection and formal approach, respectively. FTA-like approaches are not considered in this scheme as these require software hazards as an input and therefore do not provide concepts to identify these. In addition to this abstract categorization each approach is classified according to the scheme illustrated in Fig. 1

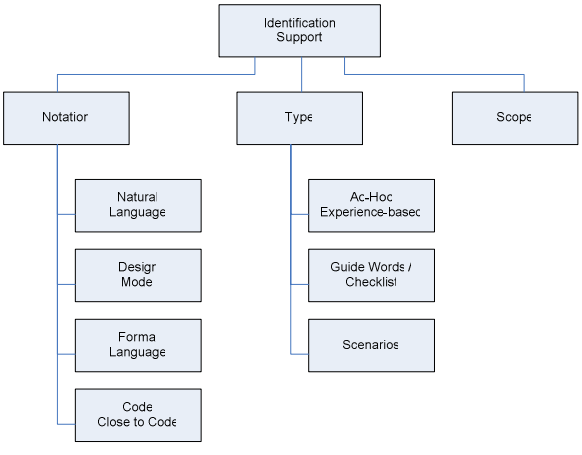

Fig. 1. Categorization Scheme for Characteristics of Software Hazard Analysis Approaches

Notation classifies the approaches according to the software development notations to which they can be applied. The sub-classes of *Notation* are not orthogonal, that is, it is possible that a technique is applicable to different notations. *Type* characterizes the support provided by the approach in detecting software causes of hazards. The subtype "ad-hoc/experience-based" indicates that no explicit support is provided; the sub-type "guide-words/checklists" indicates that some triggers are provided that point analysts to potential software faults; "scenarios" represent a special kind of support that gives procedural guidance to the analysts. *Scope* provides a classification of the types of software issues that are addressed (e.g., communication issues, correctness issues, completeness aspects). Additionally, the approaches are classified according to the software life-cycle phases they are designed for (i.e., requirements analysis, architecture definition, detailed design, and code).

We classified 60 references according to this scheme. 33 approaches explicitly mention the use of FMEA principles to analyze software caused hazards, including approaches that use a combination of FMEA and FTA. 20 approaches are based on HAZOP ideas and five on inspection ideas. This result indicates that FMEA is the technique applied most frequently for identifying and analyzing software causes of hazards. Classifying the approaches according to Fig. 1 shows that most of the approaches are defined for lower-level development phases, i.e., detailed design and code. This finding seems to contradict the finding that natural language is the notation to which most of the approaches are applied. However, a close analysis of the approaches shows that they operate on detailed design specifications of code modules written in natural language, technical models like state-charts, and variable definitions. Analyzing the scope of the existing techniques shows that due to the provided guidance, i.e., guidewords such as commission, omission, early, late, and the application of these on assets like services, variables, data rates, and signals mainly the detection of syntactic faults and correctness issues is supported. Software faults on a more subtle, logical level are often not detectable using these approaches.

2.2 Detailed Discussion of Selected Approaches

In the class of HAZOP-like approaches, the most prominent one is the SHARD approach defined by [9]. The underlying idea of the approach is the suggestion of potential failure modes of the software by means of guidewords. The focus is on the interfaces of major software components and on the data- and control-flow between them. SHARD provides the guidewords service commission, service omission, service timing (early, late), and service value (incorrect) to support the detection of potential deviations of the software behavior during the requirement phase. Lisagor et al. extended the SHARD approach for architecture evaluations [12]. Similar approaches are SoftwareHAZOP [10, 11] which applies standard HAZOP guidewords (more, less, part-of, other-than, before, after, etc.) to different software notations (data-flow diagrams, state-charts, class-diagrams). A formal variant of HAZOP-like techniques is defined by Reese et al. [13]: the software deviation analysis. The idea is that based on pre-defined software deviations, it is possible to derive deviation scenarios from a formal model of the software.

Regarding FMEA-like approaches the most prominent ones are the HiPHOPS approach [14, 15] and the Bidirectional Analysis [8]. Both approaches are a combination of FMEA and FTA approaches during the software requirements and design phase. The idea is to investigate the impact of software failure modes on the software and system level using a FMEA. Then, the identified hazards that are most critical are analyzed in detail by means of an FTA to decide whether or not the hazard really can occur. In case of [15], the FTA can be automatically derived from the formal representation of the FMEA results. More recently, the SoftCare approach has been defined [16]. This is also a combination of FMEA and FTA but some more guidance is provided on how to identify initial software failure modes. For this purpose, the guidewords (commission, omission, service timing, service value) are applied to software-related constructs (data, procedures, variables). The resulting list of potential software failures, however, contains a huge list of items pointing mainly to syntactic issues (e.g., wrong data value, late procedure call).

Summarizing the analysis of the state of the art of existing approaches, the following open issues get evident: 1) Even though requested by many authors (e.g., [2], [6], [11]), there is no systematic approach on how to perform a customization of the analyses to a given project context. 2) Many approaches assume that the software failure modes, i.e., the software causes of a hazard, are already known when the analysis starts. In practice, this is not the case and the identification of the failure modes is dependent on the experience and the knowledge of the analysts. This results in subjective, non-repeatable, hard-to-compare results. 3) Even in the case that guidance is provided for detecting potential software failure modes this guidance mainly focuses on correctness and completeness issues of software work products. However, such aspects are also addressed by standard software inspection approaches and it is important to carefully analyze the overlap of software inspections and the proposed software safety analysis approaches. Independent of that is the fact that systematic guidance on how to detect conceptual faults that have an impact on software-safety are missing. Remember the airbag example given in the introduction. The software was correct and complete but contained a conceptual fault. The algorithms used in the software to detect the system state "crash" contained a conceptual fault as they were too shock-sensitive in certain driving situations.

3 The SafeSpection Framework

The overall objective of SafeSpection is the systematization of the detection of software caused hazards, i.e., software failure modes. In that sense, SafeSpection supports the customization and execution of software FMEA and software FTA analyses by guiding the analysts in the identification of software causes of hazards. Hence, SafeSpection must not be perceived as a substitution but as an add-on to these approaches to overcome the issues related to their execution. A framework for customizing software safety guide-phrases to a specific project context is the core of the SafeSpection approach that realizes the systematization.

3.1 Approach to Systematization

In order to efficiently and effectively detect software failure modes, it is essential to provide systematic guidance that focuses the analysts not only on syntactic issues but mainly on conceptual software faults. The following example taken from a real-world accident illustrates the importance of focusing on conceptual software faults: The system specification requires that certain functions of an electronic control unit of an aircraft are executable if and only if the plane is "on ground". The system-state "on ground" is realized by the software as:

aircraft is on ground if signal_wheels_turning == true and signal_pressure_wheels >= x lb.

Typically, the moderator of a FMEA or a HAZOP analysis is responsible for triggering the analysis team with suitable questions that point to potentially unsafe behavior. Using HAZOP guidewords, one would trigger the team with questions like "Is the signal_wheels_turning correct?", "Is the signal_wheels_turning late?", "Is the signal_pressure_wheels too early or omitted?" To answer these questions the software is analyzed in detail to determine whether or not these events can occur. This is mainly a

syntactic check. What is missing is the check of whether or not the software realization represents a safe solution. An experienced moderator might ask additional questions pointing at conceptual faults like: “Is it possible that the aircraft is on ground but the wheels are not turning?” or “Is a situation possible where the wheels are not turning or pressure is < x lb but the aircraft is on ground?” Asking these questions reveals that the software realization is correct but not safe: in case of aqua-planning the wheels are not turning but the aircraft is on ground!.

In order to overcome the reliance on expert experiences, SafeSpection provides an abstract framework that allows the flexible customization of guide-phrases to a specific project and product context. The guide-phrases are defined in such a way that they support the detection of **conceptual** software faults that cause system level hazards. The underlying idea of the framework is the “formalization” of the provided support in terms of guide-phrase patterns that are derived from a guide-phrase meta-model. Based on the meta-model and the patterns, it is possible to instantiate project-specific guide-phrases that point to conceptual faults. Both the definition of the patterns and the instantiation of the guide-phrases are supported by SafeSpection guidelines. This results in more specific guidance on the detection of software-caused hazards, reduces the overlap of software-safety analysis and standard quality assurance by focusing on conceptual faults rather than syntactic issues (which are addressed by standard quality assurance activities like software inspections), and makes the results of the analysis repeatable, i.e., less dependent on individual experts, and easier to compare between teams. The core elements of the SafeSpection framework and their application are outlined in the following sections.

3.2 The SafeSpection Framework Concepts and Their Application

SafeSpection differentiates between three abstraction layers of guide-phrases (cf. Fig. 3). On the highest level, meta-meta-questions define the building blocks of a guide-phrase. The meta-meta-questions are the fundamental element for defining systematic and repeatable guidance for the detection of conceptual software faults, as they prescribe the structure of a general guide-phrase. According to the SafeSpection approach, a guide-phrase comprises two main parts, a Trigger-Part and an Effect-Part (cf. Fig. 2). The Trigger-Part is a sentence that represents a question pointing to elements in the functional specification of the software that might contain conceptual faults. The Trigger-Part element comprises three sub-elements: The *Object* represents elements of a software specification in the focus of the analysis for potential faults (e.g., a function, service or component). The *Influence Factor* describes issues that can have a potentially negative impact on the *Object*. Finally, *Interference* describes the type of impact that is imposed by the *Influence Factor* on the *Object*. Each trigger-part of a guide-phrase has one object, one influence factor, and one interference. The *Effect-Part* is either a closed question asking about the possibility that an already known hazard is caused by the question described in the trigger-part, or it is an open question asking about the possible / thinkable consequences or impacts if the question described by the Trigger Part becomes true.

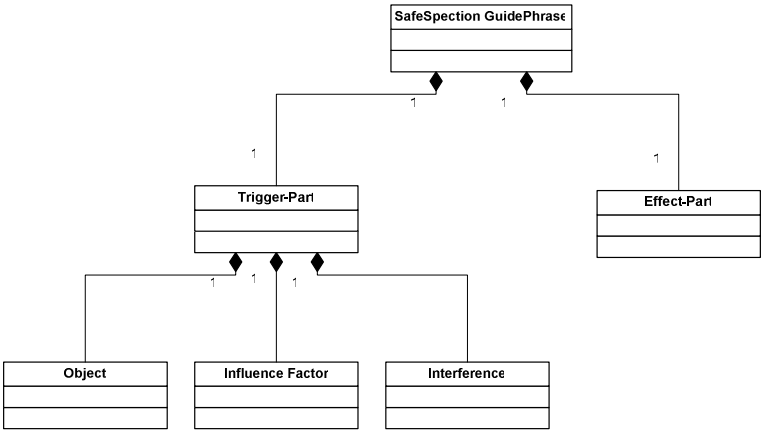

Fig. 2. The Meta-Meta-Questions Defining the Structure of SafeSpection Guide Phrases

Having introduced the basic building blocks, the SafeSpection framework defines meta-questions that represent domain-specific instantiations of the concepts *Objects* and *Influence Factors*.

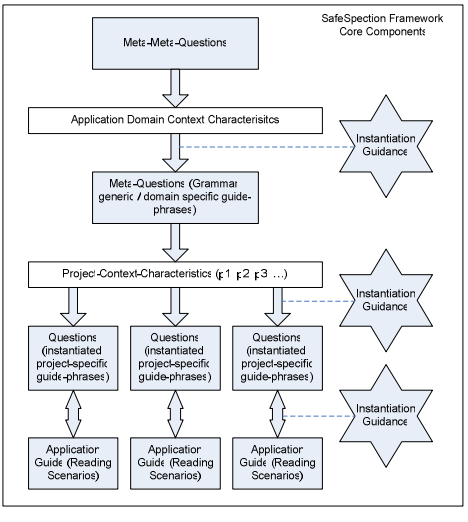

Fig. 3. The Hierarchies for the SafeSpection MetaModel

Consequently, these meta-questions represent generic guide-phrases that are applicable in a certain domain and that are an intermediate abstraction layer allowing the systematic customization of feasible, project-specific guidance.

In order to use the SafeSpection framework efficiently it is the responsibility of a safety manager and a software development leader to identify the domain-specific instantiations of the meta-meta-questions. This activity has to be done in close cooperation between software development and safety management to gather the

Nr.	Question
1	How is the behavior of the application software typically described in the domain (in terms of functions, services, processes)?
2	Can software functions within this domain be characterized with respect to time constraints, pre-conditions, post-conditions?
3	Which modeling elements / components are used to describe the software function in the domain (e.g., sate-machines, data-flow models)
…	…..
12	Are assumptions regarding the realization of functions to be considered?
13	What environmental conditions can have an impact on the software behavior (e.g., weather conditions, road-conditions)?
14	Is the software behavior dependent on operational modes (e.g., power-up, power-down)?

Fig. 4. SafeSpection Interview-Guide to Identify Guide-Phrase Objects and Influence Factors

knowledge and experience of both worlds. The concepts need to be identified for the given domain of the systems developed (e.g., electronic control units for a car). SafeSpection provides an interview guide that supports the identification of relevant objects and influence factors (see an excerpt in Fig. 4).

The result of this activity is a set of generic guide-phrases that comprise the domain-specific objects and influence factors. The results of the interviews should be backed up with a comprehensive study of existing functional specifications of ECUs in the company to identify additional objects and influence factors. In the aircraft example, one could identify software realizations of external conditions as an object, i.e., the formula for "on ground". Examples of influence factors in this domain are weather conditions, flight situations like landing, take-of, and so on. Hence, the identification of a complete set of object and influence factors is a crucial success factor of the SafeSpection approach.

We elicited domain-specific objects and influence factors in the context of functional specifications of electronic control units of cars. In this domain, typical objects are the *functions/services* and their characteristics (pre-conditions, timing constraints, realization, accuracy, assumptions), the *interfaces* of the functions/services to other functions (i.e., exchanged signals, their syntax, their semantic, and timing); and the *interactions* the functions are involved in. The influence factors that can have a negative impact on these objects are in the SafeSpection approach: *environmental conditions* (e.g., weather conditions, road conditions), *operational situations* (e.g., high-speed driving, urban driving), *technical constraints* (e.g., latency of actuators, frequency of sensor polling), *realization assumptions* (e.g., algorithm xyz is used to approximate vehicle speed), *operational modes* (e.g., power-up, power-down, diagnosis), and the *change of technical constraints* (e.g., reusing software in another hardware environment, change of sensor characteristics due to aging).

In order to standardize the definition of the generic guide-phrases, SafeSpection provides a grammar. This grammar defines rules on how objects and influence factors are combined into a SafeSpection guide-phrase. The core structure of each guide-phrase follows the rule: $S \rightarrow$ *Intro* ● *Influence* ● *interference* ● *Object*?, where Intro is a phrase introducing a question, like "Does the…", "Is it possible that…". Influence and Object are the identified domain-specific objects and influence factors, and interference defines the type of impact on the object. In our aircraft example applying SafeSpection leads to the following generic guide-phrase: "Does the <<weather

condition>> invalidate the <<software realization>> of <<system condition>>. The words in <<..>> are the generic objects and influence factors that need to be identified by the experts using the SafeSpection interview guidelines.

According to the combination of objects, interferences, and influence factors, the SafeSpection approach predefines the following types of guide-phrases that address certain types of software-caused hazards in the context of an ECU.

Name	Scope
1. Overall assumptions	Supports the identification of software-caused hazards that stem from a violation of system-wide constraints, pre-requisites and assumptions by the software realization.
2. External Influence	Supports the identification of software-caused hazards that stem from an inappropriate consideration of special characteristics of driving situation, operational modes, and environmental conditions in the software.
3. Changed Environment	Supports the identification of software-caused hazards that stem from changes in the software environment (like changed technical constraints, changed application context, changed sensor characteristics) that are not properly mapped / considered in the software realization.
4. Communication	Supports the identification of software-caused hazards that stem from wrong or inappropriate interactions of software elements and software-realized functions / services / processes.
5. Functional Realization	Supports the identification of software-caused hazards that stem from an improper realization and an insufficient consideration of influences on the software behavior (like the fulfillment of assumptions, prerequisites constraints that are not given in certain operation of modes).
6. Special Functions	Supports the identification of software-caused hazards that stem from the implementation of degradation scenarios that are not properly integrated in the overall functional concept.

Fig. 5. Types of Guide-Phrase Patterns in the SafeSpection Framework

For each type, one or more generic guide-phrase is provided. With respect to the analysis of functional specifications of ECUs, we defined a set of generic guide-phrases for the types defined above. The following questions represent guide-phrases of the type external influence and changed environments:

Does the <<characteristic>> of <<driving situation>> invalidate the <<pre-condition>> || <<post-condition>> of <<function>>?
Does the change of <<characteristics>> of <<sensor>> || <<actuator>> violate the timing-constraints of <<function>>?

In the Appendix of thi paper an excerpt of the full set of generic guide-phrases supporting the detection of conceptual faults is listed. The advantage of the guide-phrases is their generic nature aimed at conceptual faults compared to the syntactic guidance provided by existing HAZOP guidewords. Moreover, the guide-phrases are already tailored to the application domain and hence more specific than general-purpose guidewords. The following comparison clarifies this advantage: the checklist questions defined by Leveson [2] typically aim at completeness issues, e.g.,: “A trigger involving the nonexistence of an input must be fully bounded in time”. Guide-phrases defined with the SafeSpection approach would perceive the definition of such a trigger and its time bounds as objects of the specification, i.e., SafeSpection takes these as inputs. These objects are combined with influence factors to check whether or not the time bound can be violated for example by external conditions or whether or not the time bound contradicts realization assumptions underlying the software.

Finally, the generic guide-phrases are instantiated to concrete guide-phrases that are applicable in a certain project. That is, the generic meta-questions defined for the application domain are instantiated with concrete objects and influence factors of a

software project. In our aircraft example, the generic guide-phrase is instantiated with the concrete objects and influence factors: “Does rainy weather invalidate the software realization plane on ground if the wheels are turning and the pressure is >= x lb?” The person responsible for this activity is typically a quality assurance person of the project team whose functional specification is analyzed. The resulting guide-phrases are used by the analysis team to identify conceptual faults in the functional specification. It is most important to identify those generic guide-phrases that are relevant for the specific project context. Again, the SafeSpection framework provides guidelines on how to perform this instantiation in terms of expert interviews. The project-specific guide-phrases result in systematically tailored guidance addressing the real safety needs in a project context. The detection of conceptual software faults in the project becomes a repeatable activity and focused on the project-characteristics rather than on providing general-purpose guidance. The identified guide-phrases can be used as a stand-alone technique similar to an inspection approach, using the guide-phrases as checklist questions or as part of the software FMEA and software FTA activities where they guide the analysis team in detecting software failure modes and software causes of hazards.

4 SafeSpection Application

In order to validate the applicability of the SafeSpection framework, we validated its core concepts in an industrial project. The objective of the project was the development of a complex, distributed system to realize new functionality in a car. Due to confidentiality reasons it is not possible to show details of the software system or its architecture, but on an abstract basis the project can be described.

4.1 The Application Context

The software system in this project realizes an innovative feature of a future car. The overall software system comprises 8 sub-systems interconnected by a network. Each sub-system is responsible for the realization of one or more features of the functionality. By applying SafeSpection, the manufacturer of the system wanted to ensure that the software system does not impact the overall value-adding processes in an unacceptable way. Hence, in this project, safety was not defined in the common way, i.e., loss of life, or injury to people, but as the loss of an immense amount of money due to such potential negative influences caused by the software system. The SafeSpection approach was used to support the identification of conceptual faults in the general functional specification of the software system and its conceptual architecture. The analysis was performed at the end of the requirements analysis step and after the conceptual architecture of the system had been defined. The manufacturer had already performed an analysis of the potential risks caused by the software system but without systematic guidance.

4.2 The Application Process

The execution of the software safety analysis was organized in 3 full-day workshops. Based on the already identified catastrophic influences of the software system, a

fault-tree analysis was performed to analyze the software causes of the unwanted events. In order to support this step, i.e., the identification of conceptual software faults causing the top-event, the SafeSpection framework was used to identify and apply supporting guide-phrases. As outlined in the last section, the first step of the SafeSpection approach is the definition of generic guide-phrases that combine objects and potential influence factors. The analysis of the 500-page software specification written mainly in natural language and the conceptual overview of the software architecture resulted in the following generic objects: processes, components, interactions, pre- and post-conditions, assumptions, and constraints; and in the generic influence factors operational mode, system assumptions, technical and environmental constraints. Based on these concepts generic guide-phrases could be created.

In order to identify potential software causes for the unwanted events, the concrete instances of the identified generic phrases needed to be identified. This was done as part of the FTA workshop. Starting from the unwanted event, those concrete system processes influenced by the software were identified that directly contribute to the unwanted event. Then the components realizing the identified processes as well as the interaction of these components were identified together with the customer's experts. This was done using the customization questions defined in the SafeSpection framework and the results of this step were documented by extended sequence charts showing all concrete objects of realizing the selected processes (see Fig. 6).

The swim-lanes show the concrete components that participate in the identified processes. The grey-boxes represent the objects, pre-conditions, post-conditions, constraints, and assumptions. These were also identified as part of the workshop in cooperation with the customer's experts. For example, the component Pre-Processing 1 requires as a pre-condition the availability of a certain data-item (xyz) and that the initialization has been performed successfully. The component Pre-Processing 2 must fulfill the constraint that the processing of data is completed within 5 ms. The component Data-buffer contains the implicit assumption that not more than 25 requests are sent within one second. Finally, as a post-condition of the whole process the plausible data are presented at the software interface as a output.

The negative form of the post-condition of the whole process represents the unwanted event, i.e., the top-event of the fault tree. Now, the selected guide-phrase patterns guide the identification of the causes of the unwanted top-event. In other words, the guide-phrases were used to systematically identify potential software causes of the unwanted top-events. As it was not possible to derive explicit influence factors prior to the workshop (due to time limitations in the project) the guide-phrases were used as open questions. That is, the guide-phrase patterns were modified in such a way that they ask for potential influence factors that invalidate the object under discussion. The following list shows an excerpt of instantiated guide-phrase patterns derived for analyzing the objects in the sequence chart.

Is it possible that the realization of pre-processing 1 violates the timing constraint "needs to finish in 5 ms" of pre-processing 2?

- Which characteristic of the operational mode contradicts the realization of pre-processing 1?
- Which external condition invalidates the realization of pre-processing 2?
- Which change of characteristics of external components interacting with the application software violate the pre-conditions of pre-processing 1?

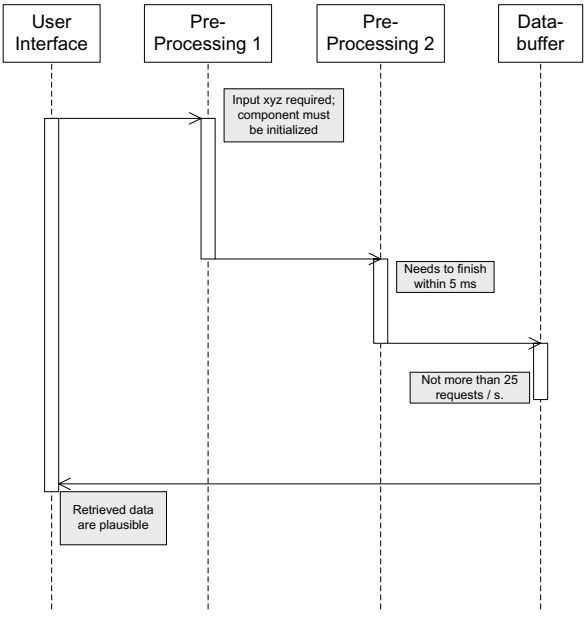

Fig. 6. Application of the Guide-Phrases

- Which change of characteristics of external components interacting with the application software violate the assumptions of the data buffer?
- Is it possible that the semantic of messages is different for pre-processing 1 and pre-processing 2?

The analysis starts from the unwanted top-event and asks whether or not an intermediate event that is described by the guide-phrase triggers the top-event. If this is the case, the event is added to the fault tree, if not, the event described by the next guide-phrase is investigated until all guide-phrases have been considered. The workshop leaders (two of the authors of this paper) derived the guide-phrases, asked the related questions, and modeled the results as extensions of the fault tree.

4.3 The Application Results

Using the guide-phrases defined by the SafeSpection approach resulted in a systematic and easy to apply refinement of the fault tree top-events. The developers involved in the analysis perceived the fault tree technique and the systematic consideration of potential causes as a highly valuable technique to detect conceptual faults in their functional software specification. The application of SafeSpection resulted in project-specific guidance, which could be quickly derived during the FTA-meeting. The management perceived the approach as a success, as the results provided additional conceptual weaknesses in the software specification.

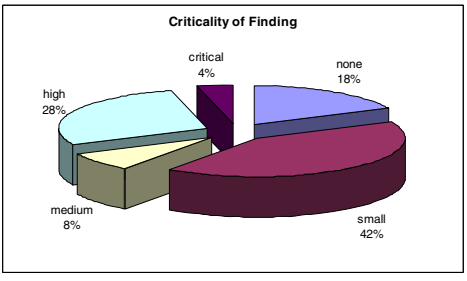

Fig. 7. Criticality of the Software Faults detected with SafeSpection

For the highest prioritized top-event, for example, we could identify 50 additional software faults that could cause the top-event. For the 32 findings rated as high and critical a careful re-consideration of the software specification was performed and mitigation strategies needed to be defined. These results show that the SafeSpection approach created customized guide-phrases that identify so far undetected conceptual software faults quickly and in a feasible way during the FTA-meeting. We could detect and resolve several faults that might have caused catastrophic events for the company.

5 Conclusion

The SafeSpection framework introduced here provides a feasible approach to identify customized and project-specific guidance for the detection of conceptual software faults that have the potential of causing safety-critical system events. We demonstrated the feasibility of the core concept of the approach (a grammar for defining generic guide-phrases) in an industrial case study. The customized guide-phrases supported the identification of 50 additional software faults; 32 of them required the definition of suitable mitigation strategies to prevent a catastrophic top-event.

In future steps, it is important to validate the applicability of the customization approach in more detail. First, it is important to validate the completeness of the provided support on identifying objects and influence factors of the guide-phrases. Second, the resulting guide-phrases will be compared in an empirical study with standard software-safety analysis techniques (like FMEA or FTA) with respect to the type of detected software faults (is it possible to detect more conceptual faults) and the repeatability and comparability of the results.

References

1. Knight, J.C.: Safety Critical Systems: Challenges and Directions. In: 24th International Conference on Software Engineering (ICSE 2002), pp. 547–550. ACM, New York (2002)
2. Leveson, N.: Safeware – System Safety and Computers. Addison Wesley Publishers, Boston (1995)
3. IEC 61508: Institute of Electrical and Electronics Engineers. Functional Safety of electrical/electronic/programmable electronic safety-related systems Part 3 Requirements on Software (1999)

4. ISOWD 26262, Road vehicles, Functional Safety Part 6: Product development software. Working draft (2006)
5. Fenelon, P., McDermid, J.A., Pumfrey, D.J., Nicholson, M.: Towards Integrated Safety Analysis and Design. ACM Computing Reviews 2(1), 21–32 (1994)
6. McDermid, J.A.: Software Hazard and Safety Analysis. In: Lecture Notes in Computer Science, vol. 2469, pp. 23–34 (2002)
7. Papadopoulos, Y., et al.: A Method and Tool Support for Model-based Semi-automated Failure Modes and Effects Analysis for Engineering Designs. In: 9th Australian Workshop on Safety Related Programmable Systems (SCS 2004), pp. 89–95. Australian Computer Society (2004)
8. Lutz, R.R., Woodhouse, R.M.: Bi-directional Analysis for Certification of Safety-Critical Software. In: The proceedings of the International Software Assurance Certification Conference (ISACC 1999), pp. 1–9. Springer, Heidelberg (1999)
9. Pumfrey, D.J.: The Principled Design of Computer System Safety Analysis. PhD thesis. Department of Computer Science, University of York, UK (1999)
10. Chudleigh, M.: Hazard analysis using HAZOP: A case study. In: 12th International Conference on Computer Safety, Reliability and Security (SAFECOMP 1993), pp. 99–108. Springer, Heidelberg (1993)
11. Redmill, F., Chudleigh, M., Catmur, J.: System Safety: HAZOP and Software HAZOP, p. 248. John Wiley & Sons Ltd., Chichester (1999)
12. Lisagor, O., et al.: Safety Analysis of Software Architectures – Lightweight PSSA. In: The proceedings of the 22nd International System Safety Conference (ISSC 2004). IEEE Computer Society, Los Alamitos (2004)
13. Reese, J.D., Leveson, N.G.: Software Deviation Analysis. In: 19th International Conference on Software Engineering (ICSE), pp. 250–260. IEEE, Los Alamitos (1997)
14. Papadoupoulos, Y., et al.: Hierarchically Performed Hazard Origin and Propagation Studies. In: Felici, M., Kanoun, K., Pasquini, A. (eds.) SAFECOMP 1999. LNCS, vol. 1698, pp. 139–152. Springer, Heidelberg (1999)
15. Papadopoulos, Y., et al.: Automating the Failure Mode and Effects Analysis of Safety Critical Systems. In: The proceedings of the 8th International Symposium on High Assurance Systems Engineering (HASE 2004), pp. 310–311 (2004)
16. Rodriguez-Dapena, R.: Software safety verification in critical software intensive systems. Phd Thesis, Eindhoven Technical University, University Printing Office (2002)