Indicator-Based Inspections: A Risk-Oriented Quality Assurance Approach for Dependable Systems

Frank Elberzhager, Robert Eschbach, Johannes Kloos

Testing and Inspections Department
Fraunhofer Institute for Experimental Software Engineering
Fraunhofer Platz 1
67663 Kaiserslautern, Germany
{frank.elberzhager, robert.eschbach, johannes.kloos}@iese.fraunhofer.de

Abstract: We are surrounded by ever more dependable systems, such us driving assistance systems from the automotive domain or life-supporting systems from the medical domain. Due to their increasing complexity, not only the development of but also the quality assurance for such systems are becoming increasingly difficult. They may cause various degrees of harm to their environment. Hence, in order to reduce risks associated with these systems, development as well as quality assurance normally use risk analysis as a basis for constructive and analytical measures against these risks. One of the aims of quality assurance is fault detection and fault forecasting. In this paper, the authors present indicator-based inspections using Goal Indicator Trees, a novel risk-oriented quality assurance approach for fault detection. It can be used to detect faults of different types, like safety faults or security faults. Starting from typical risk analysis results like FMECA and FTA, the approach systematically derives quality goals and refines these goals into concrete quality indicators that guide the indicator-based inspection. Quality indicators can be mapped to concrete checklists and concrete inspection goals in order to support inspectors checking artifacts in a fine-grained way with respect to certain quality properties. The approach is explained and demonstrated with respect to the quality property safety, but tends to be generalizable to further quality properties.

1 Introduction

With the growing importance of systems that help control critical aspects of our environment, such as automotive driving assistants and medical life support systems, the importance of guaranteeing the dependability of software and embedded systems is also increasing steadily. As these systems often have high inherent complexity, both constructive measures and analytical quality assurance measures are necessary to ensure the required level of dependability.

To ensure that the dependability properties required of a system are fulfilled, a wealth of measures are taken, starting with risk analysis to identify possible problems, followed by constructive measures to ensure fault avoidance and fault tolerance, and concluding with quality assurance measures for fault removal and fault forecasting. Sadly, the results derived during risk analysis are often too coarse to allow a truly satisfying quality

assurance: The descriptions of different risks and countermeasures may lack the detail needed to effectively ensure that appropriate risk reduction and prevention steps have been taken.

In particular, common risk analyses like Failure Mode, Effect and Criticality Analysis (FMECA) and cause-effect analyses like Fault Tree Analysis (FTA) tend to terminate with fairly abstract causes for hazards, and the measures taken inside the system might not have obvious connections with the hazard causes. Moreover, existing inspection reading techniques like checklists or scenarios that support inspectors in finding problems often also contain questions that are not focused enough. One method that aims to solve this problem are Security Goal Indicator Trees (SGITs [19]), which allow the refinement of security goals into indicators that can be used directly to drive inspections of the system under consideration. On the other hand, this technique is specially tailored to the security domain, not considering other quality properties such as safety or reliability.

The approach presented here builds upon the idea underlying SGITs, generalizing them into Goal Indicator Trees (GITs) so that they may be applied to different quality properties. In particular, quality goals are identified, which are then refined into sub-goals making up a particular goal. This refinement is continued until one can derive indicators, i.e., statements about the system that can be directly checked by looking at one or more system artifacts. These indicators are then used to drive an inspection, with the benefit of supporting an inspector by presenting information how to read artifacts such as requirements, design or code documents in a detailed manner. As the construction of the indicators is a systematic process in which domain experts should be involved, it is reasonably certain that all important facets for a given quality goal are considered inside a GIT. The work presented in this article describes work in progress and presents the conceptual framework towards risk-oriented indicator-based inspections demonstrated with a safety-critical example from the automotive domain.

The remainder of this work is structured into three sections: Section 2 gives an overview of related work, both with regard to analysis techniques and inspections. Section 3 describes the approach in detail, illustrating it with an example from the automotive domain. The quality property under consideration is safety, and the derivation of questions guiding an inspection for a given safety property is shown, together with the actual inspection results. Finally, Section 4 summarizes the results and gives details about ongoing evaluation activities and further research directions.

2 Related Work

Two areas should be sketched: on the one hand, methods that support analysts and developers during certain development steps ensuring a certain quality constructively and, on the other hand, those that do so analytically. Our focus is on static quality assurance methods, which are suitable for analyzing certain development artifacts such as requirements documents, design documents, models, or code.

A common classification of the means for ensuring dependability is described in [12]: A method may focus on fault avoidance, fault tolerance, fault removal, or fault prediction. Construction-time methods tend to fall into the fault avoidance and fault tolerance classes. For many of these approaches, the first step is to carry out an analysis that will identify the risks inherent in the system, as well as possible causes that lead to the manifestation of these risks. From these analyses, possible countermeasures are derived that will help to either reduce the probability of occurrence of each hazard or its severity.

There are numerous examples of such analysis techniques with different specializations. In the context of safety, two of the most common methods, often used in tandem, are FMECA and related techniques like FMEA [13][14][15] and FTA [16][22]. Both techniques have complementary goals: While FMECA is used to identify failure modes of the system and assess their criticality; FTA is used to identify the underlying causes that may lead to the manifestation of a failure mode. These techniques are usually employed together: possible system failure modes are determined using FMECA, and iteratively refined to "basic events" using FTA. These basic events are defined as "component failures" [22]. For software, this means that basic events are formulated as "software component failure", without describing possible causes.

For reliability analysis and identification of reliability-critical components, Reliability Block Diagrams (RBDs, [17]) are used. These models are built from information about the system structure and individual component reliabilities. Component failure causes are not considered.

Finally, for security, attack trees [18] allow an approach similar to FTA, while SGITs [19] are used for fine-grained analysis. From our point of view, only SGITs have a sufficient level of detail for guiding inspectors when focusing on a certain quality property: The combination of FMECA and FTA stops at a level where the failure of individual components is considered, not going into the detailed analysis of failure causes of software, while RBDs do not consider failure causes at all.

For checking certain development artifacts analytically, software inspection is a suitable means. Inspections have existed for over thirty years and were first published in 1976 by Fagan [1]. They are a structured and well-defined static quality assurance method for verifying the quality properties of different artifacts. Two main research directions can be identified regarding inspections, namely, the inspection process itself (e.g., phased inspection [3], n-fold inspection [4]) and reading support for defect detection by inspectors.

While focusing on support for developers, analysts or, inspectors in general, in order to check certain qualities, different reading support were developed that can and should be used to find defects (which can also mean violations regarding a certain quality) in an effective manner. Three different kinds can be distinguished. First, ad-hoc reading, where an inspector does not get any reading support. In this case, the inspector can perform the defect detection based solely on his knowledge and experience. Second, checklists can be used. Beside one general checklist used by each inspector [5], focused checklists [6] and guided checklists [7] were developed in order to present different

perspectives or defect classes that can be looked for in an inspection. The main advantage of focused and guided checklists is the higher defect coverage within the artifact to be checked and the lower overlap of defects found by the inspectors, i.e., inspectors mainly find different defects, resulting in higher effectiveness. One main problem with checklists is that the checklist questions are often too general, as stated in [8].

The third class of reading techniques are scenarios. The idea is that an inspector should work actively with a document instead of only reading checklist questions in a passive manner. For this, an inspector gets, for example, a description of his perspective (perspective-based reading [5]) or a certain defect class (defect-based reading [9]), which sets the focus. Next, concrete instructions have to be followed and questions have to be answered. For example, imagine an inspector taking the tester perspective. One instruction might be, "Derive a number of test cases from the corresponding document", and a possible question is, "Is all necessary information for deriving test cases stated?"

Despite many research activities regarding reading support and a number of existing reading techniques as mentioned above, little work has been done on how to support inspectors with detailed reading support to ensure certain qualities in a holistic way. The already mentioned SGITs and guided checklists are initial methods that go into this direction. Nevertheless, they only focus on the quality property security and have not been generalized to further qualities. Finally, little support is given today that describes how to derive reading support for inspectors.

In summary, despite a lot of existing constructive and analytical methods to ensure certain qualities, a generalized approach for focusing on the details of quality attributes that should be ensured across the software development cycle (i.e., in the created documents and the code) is still missing.

3 The approach

The approach for creating reading support and performing the indicator-based inspection is carried out in three steps, shown in Figure 1. To describe these steps, consider the following example: The object under consideration is the control unit for an electrically-powered car window, as described in [10]. As it is a safety-critical system, the quality assurance of this control unit must demonstrate that all safety-related non-acceptable risks have been reduced to an appropriate level [11][20].

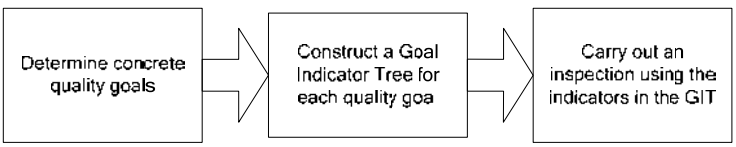

Figure 1: Steps of an indicator-based inspection

The system under consideration is called “Automotive Power Window System”. It describes an electronic control unit for an electrically controllable car door window. This control unit allows the driver (and the passenger) to move the window inside the car door using an up/down button. Additionally, it will detect whether the window is jammed by some object, using this information to avoid crushing that object. The inspection will be carried out on the requirements for this device as well as on a Matlab/Simulink model of the system (i.e., a model inspection as required in [11] for a software unit design and implementation).

3.1 Determination of concrete quality goals

The first step is the selection of the quality goals in question. For safety, the quality goals tend to be highly application-specific, since every system comes with a different set of potential safety hazards. In the automotive domain, the applicable standard (ISO WD 26262, [11]) prescribes that a safety risk analysis must be undertaken. In particular, the system’s safety hazards are to be identified, e.g., using FMECA, and the causes of each failure mode must be determined, e.g., by FTA.

Using a Fault Tree Analysis or similar techniques, one arrives at a description of all those situations in which a safety hazard may manifest itself. These situations are often described by so-called cut sets, e.g., sets of events that must occur together. One common analysis performed during FTA is the determination of minimal cut sets, e.g. of those cut sets that describe the minimum preconditions for the occurrence of a safety hazard.

For the power window controller, an FMECA includes (among others) the following failure modes:

Table 1: FMECA of power window controller (excerpt). See [14] for a description of the columns.

Error location	*Potential error*	*Consequences of error*	*S*	*CRIT*	*Causes*	*O*	*Detection*	*D*	*RPN*
Jam prevention	Window is jammed while motor moves it up	Object crushed	9	yes	Jam detection fails	4	No simple direct solution	9	72

This failure mode is hence highly safety-critical and must be further analyzed. By carrying out a Fault Tree Analysis, one arrives at the fault tree depicted in Figure 2. The software-related basic events are marked by stripes.

To ensure that a system is safe, one therefore considers each critical minimal cut set in turn, formulating a quality assurance strategy that demonstrates, for each minimal cut set, that not all its events can occur together. Thus, the concrete quality goals can be given in the form “not all events of the minimal cut set C may occur simultaneously”, where C ranges over all minimal cut sets. The minimal cut sets of the power window controller all have the form “object jams window”, “‘window up’ button pressed”, and

one of the statements “sensor does not notice jam”, “motor moves on its own”, “sensor data not received”, “sensor data misinterpreted”, and “motor still ordered to move up”.

Hence, for deriving indicators, three minimal cut sets need to be considered:

1. “object jams window”, “‘window up’ button pressed”, “sensor data not received”

2. “object jams window”, “‘window up’ button pressed”, “sensor data misinterpreted”

3. “object jams window”, “‘window up’ button pressed”, “motor still ordered to move up”

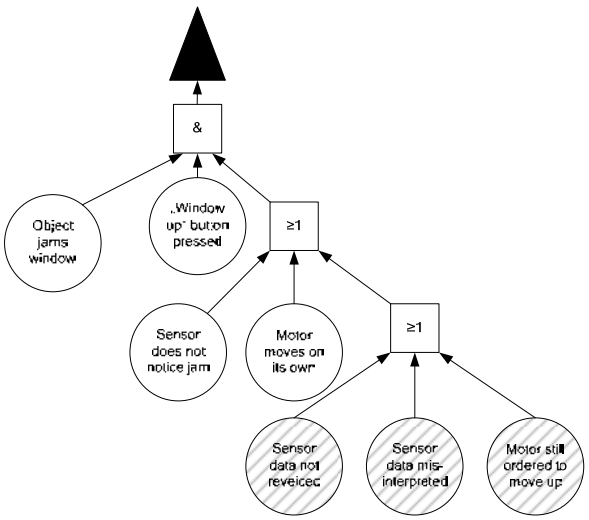

Figure 2: The fault tree for “Object jam detection fails”

From these minimal cut sets, one derives three goals:

1. When an object potentially jams the window while the “window up” button is pressed, the data from the jam sensor must be correctly received correctly.

2. When an object potentially jams the window while the “window up” button is pressed, data received from the jam sensor must be evaluated and interpreted to detect possible jamming conditions.

3. While the “window up” button is pressed, the motor must not be ordered to move up when a jam is detected.

3.2 Construction of Goal Indicator Trees for the quality goals

Next, the goals can then be iteratively refined into sub-goals and indicators. This refinement process is described exemplarily on goal 1, using a graphical notation. The result is shown in Figure 3. The detailed process and possible variations will be described in an upcoming paper; here, one variant, based on a goal refinement procedure, is illustrated on an example. A graphical notation was chosen as many common approaches (e.g., FTA and SGIT) for similar analyses also use a graphical representation.

The Goal Indicator Tree is built up from several elements, namely a single goal (at the top), two sub-goals below it, and several indicators. Goals, sub-goals, and indicators are connected by logical junctures (namely, "and", and "or"). If the goal is made up of sub-goals and indicators connected by "and", all of the sub-goals and indicators must be fulfilled for the goal to be satisfied. In the same way, if they are connected by an "or" node, at least one of them must hold for the goal to be fulfilled. The same holds for sub-goals and indicators made up of other elements. The difference between a sub-goal and an indicator is that a sub-goal may still be unspecific, while an indicator must be detailed enough to be directly checkable.

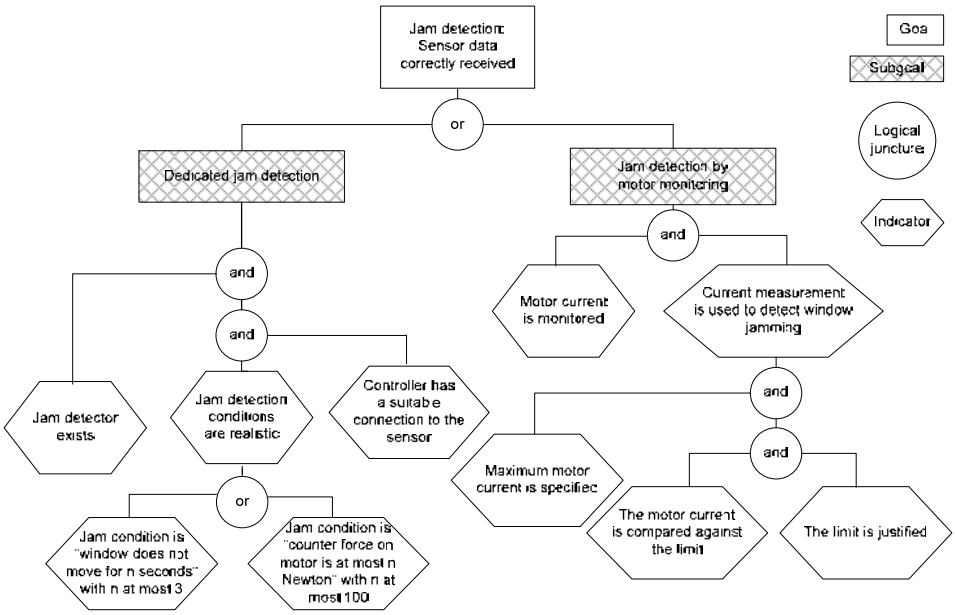

Figure 3: Derivation of indicators for a safety goal, using a Goal Indicator Tree

In the example, one starts with the first goal, "Jam detection: Sensor data correctly received". Since there are a number of possible implementations, the tree needs to be refined to allow an easier inspection. In this case, the goal describes the correctness of a system behavior that depends on the implementation of the underlying component. Thus, it is refined using an "or" juncture, resolving the dependency on the implementation by

enumeration. One arrives at two sub-goals, “Dedicated jam detection: Sensor data correctly received” and “Jam detection by motor monitoring: Sensor data correctly received”. Each of these goals describes the correctness of a system which consists of more than one component. Thus, an “and” juncture is used to split up the goal into sub-goals for each component. Following the second path, one finds that there must be both a way to monitor the motor current, and the current measurement must be used for jam detection. The existence of a motor current monitor can be directly checked by inspection (e.g., by checking whether the system has appropriate hardware). Therefore, this sub-goal is formulated as an indicator. In theory, the correct use of the current measurement could also be checked directly; hence, it is also marked as an indicator. In practice, not every inspector will be familiar with the necessary knowledge in electrical engineering to decide what an appropriate check is. Thus, this indicator is further refined.

For the second and third goal, the respective Goal Indicator Trees are shown in Figure 4.

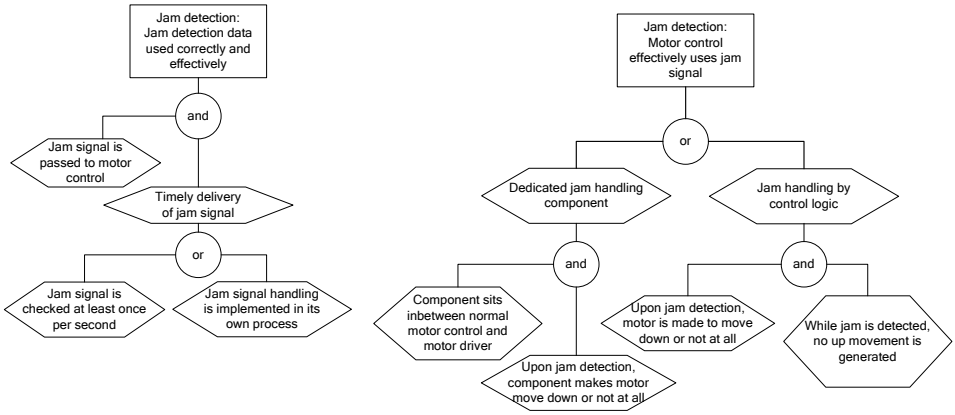

Figure 4: Goal Indicator Trees for the remaining two minimal cut sets

To keep the set of indicators manageable, one next defines assessment and selection criteria for indicators. For safety indicators based on cut sets, the following criterion can be used: An indicator is considered useful if it significantly contributes to the demonstration that the events in one of the most critical cut sets can only occur with low probability. A cut set is critical if it has a high risk priority number. As all minimal cut sets in this example have the same risk priority number, each of them must be considered.

3.3 Carrying out the inspection

Based on the defined Goal Indicator Tree shown in Figure 3, the inspection of the corresponding development documents is performed in order to find problems that violate the goal “Jam detection: Sensor data correctly received”. With respect to the inspection process, the focus is on the defect detection phase and there is a description of how to apply the GIT. A requirements document (Simulink – Automotive Power

Window System Demo – Part 1 – Designing the Controller) and a Matlab Simulink model ("Power Window", as linked from the requirements document) are used as artifacts to be analyzed, i.e., each indicator should be checked against these artifacts [10]. The goal of an inspection is to find all potential defects. Thus, it is important to check and document each indicator to the extent possible in order to find all defects that do not comply with the indicators. The order of checking is depth-first, from left to right. Finally, the GIT presents for each indicator only a short description of what to check. This explanation is often not detailed enough for non-experts regarding certain goals or for people using a GIT the first time. Thus, it is possible to derive a checklist containing more details for each indicator. Basically, each indicator is transformed into one question and enhanced with more information about what to check and how to proceed [7]. This step is skipped here. Nevertheless, in the following, more detailed questions than seen in the GIT are presented exemplarily in order to clarify some indicators.

Starting with the goal as entry point, the focus of the GIT to be checked is given. Following the described order, the sub-goal "Dedicated jam detection" has to be checked initially. The first indicator leads to the question, "Does a dedicated piece of hardware exist for jam detection?" This can be checked both in the requirements and in the model. The requirements document presents information about how an obstacle is treated while the window is closing. No dedicated piece of hardware is defined that violates the indicator. Consequently, the indicators below, which describe how to analyze implementation details, cannot be checked because the implementation of the jam detection is not done this way. Furthermore, the indicator that is not fulfilled corrupts the sub-goal and a potential problem is found (which should be documented).

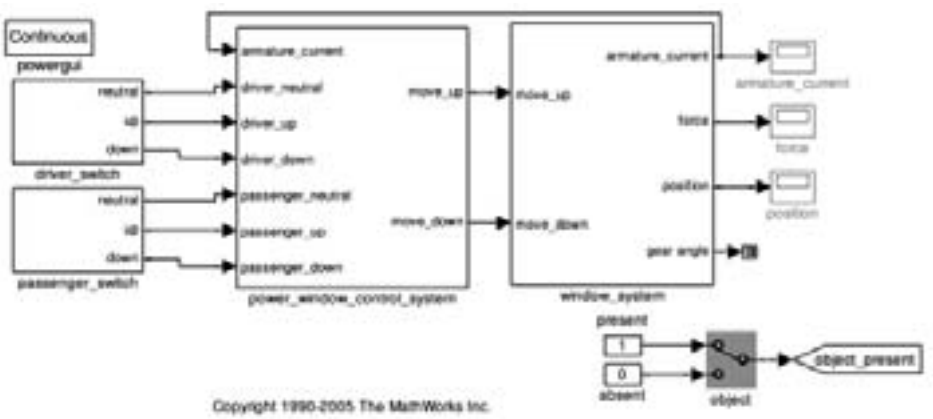

Figure 5: A high-level view of the power window control system, showing that the current running through the window movement motor is measured (the line between the armature_current connectors, at the top of the model).

Next, the second sub-goal "Jam detection by motor monitoring" should be ensured. For this, the first indicator to be checked leads to the question of whether the motor current is monitored. In the requirements document, it might say that "… the control switches to its emergency operation when a current is detected that is less than -2.5 [A]"; thus,

monitoring is documented. Next, a look is taken at the model in order to check the implementation and find out where the implementation of the requirement is done in the model (see Figure 5). A line from the window_system to the power_window_control_system exists for armature_current which indicates that the current measurement takes place.

The next indicator "Current measurement is used to detect window jamming" is checked in the requirements and could be found in a section where a description of the power window control process is given. Finally, three dependent indicators are checked, starting with the indicator "Maximum motor current is specified". Within the mentioned description, certain values for normal operation and for when an obstacle is detected are defined. That the motor current is compared against the limit can be checked in the model where the (absolute value of the) armature current measurement, whose value is read from input 2, is compared to the constant 1.2 in the comparator named "object" (see marked rectangle in Figure 6). For the last indicator, "the limit is justified", a positive answer can again be found in the requirements where the values for normal use and deviations are defined and explained.

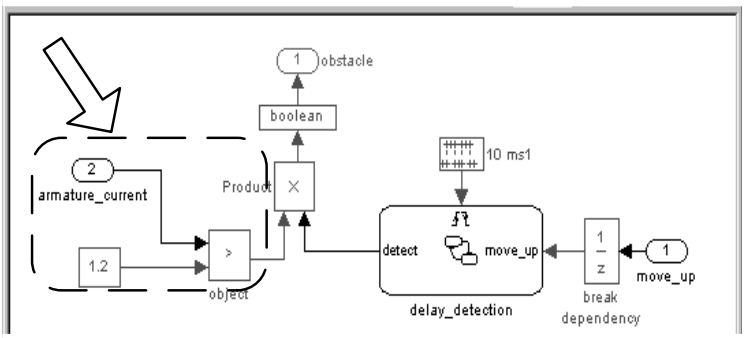

Figure 6: The position of the current comparison that is used to detect jammed windows[1].

After checking all indicators to the extent possible and documenting the issues as described, the defect detection phase is finished. Finally, in order to judge if the overall goal "Jam detection: Sensor data correctly received" is fulfilled, one has to go over the indicators and logical connectors once again. The left sub-goal "Dedicated jam detection" is not fulfilled. Regarding the right sub-goal "Jam detection by motor monitoring", each indicator is fulfilled, which results in fulfillment of the sub-goal and moreover to overall fulfillment of the goal, because the two sub-goals are connected via an "or"-node (meaning that only one of the sub-trees has to be fulfilled).

[1] Path inside the model:
powerwindow03/powerwindow_control_system/detect_obstacle_endstop/detect_obstacle

4 Conclusion and Future Work

This paper presented a novel approach that defines how an inspection could be performed using indicators in order to ensure certain quality goals. Output from existing analysis techniques like FMECA or FTA is used to identify risks and subsequently to derive goals and sub-goals, which are refined into indicators on certain artifacts, like requirements or models that can be checked by an inspector. We explained the approach with respect to a jam detection system where safety requirements should be ensured. With this, we could show that the defined approach is applicable to ensure the quality property safety (beside the initial focus on security) which gives initial evidence that the approach tends to be generalizable.

Quality assurance techniques like FMECA, FTA, or existing reading support for inspections often analyze corresponding documents on a level that is too coarse-grained, only identifying general risks. Unfortunately, only little or even no information is given on how to reduce the risk concretely. Thus, indicators are a new possibility to close this gap and to improve the artifacts to be checked with respect to the goals to be ensured, respectively the risk to be reduced. Consequently, the quality of the whole product is improved with a special focus on the qualities that are checked.

The indicator-based inspection is currently being evaluated on examples in the ViERforES research project [21]. In particular, a robot control system and an example from industrial automation are in the process of being examined using the techniques described in this paper, with more case studies planned later. The goal of these studies is to evaluate whether the techniques presented herein lead to an effective method for driving safety and security inspections (i.e., evaluation of the defect detection ability and effectiveness regarding certain quality properties, evaluation of scalability, evaluation of ensuring additional quality properties, and evaluation of the GIT construction step).

Another aspect regarding future work is an application of this approach to other domains, including relevant standards, and to further quality properties. Until now, we have been able to show that the indicator-based inspection approach is adaptable to the quality properties security [19] and safety (in this paper). We believe that more qualities can be refined with our approach and used for quality assurance in order to reduce risks in software development and thus, improve the products of practitioners.

Acknowledgements

This work was funded by the German Federal Ministry of Education and Research (BMBF) in the context of the project ViERforES (No.: 01 IM08003 B).

5 References

[1] M.E. Fagan, Design and code inspections to reduce errors in program development, IBM Systems Journal, 1976

[2] R. Ford, A. Howard, A Process for Performing Security Code Reviews, IEEE Security & Privacy, 2006
[3] J.C. Knight, E.A. Myers, An improved inspection technique, Communication of the ACM, 1993
[4] J. Martin, W.T. Tsai, N-fold inspection: a requirements analysis technique, Communication of the ACM, 1990
[5] O. Laitenberger, C. Atkinson, M. Schlich, K. El Eman, An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents. The Journal of Systems and Software, 53, p. 183-204, 2000
[6] C. Denger, M. Ciolkowski, F. Lanubile, Does active guidance improve software inspections? A preliminary empirical study, Proceedings of the IASTED International Conference Software Engineering, 2004
[7] F. Elberzhager, A. Klaus, M. Jawurek, Software Inspections using Guided Checklists to Ensure Security Goals, Secure Software Engineering Workshop, part of the ARES conference, 2009
[8] B. Brykczynski, A Survey of Software Inspection Checklists, Software Engineering Notes, vol. 24, no.1, ACM SIGSOFT, 1999
[9] A. Porter, L.G. Votta, Comparing Detection Methods for Software Requirements Specification: A Replication Using Professional Subjects, Empirical Software Engineering 3, p. 355-379, 1998
[10] The MathWorks: Matlab/Simulink Demo "Automative Power Window System", distributed with Matlab 2008a. A newer version of this example is available online at http://www.mathworks.com/products/simulink/demos.html?file=/products/demos/simulink/PowerWindow/html/PowerWindow1.html#3.
[11] ISO TS 22/SC3: ISO/DIS 26262: Road Vehicles – Functional Safety. Draft International Standard, ISO, 2009.
[12] A. Avižienis, J.-C. Laprie, B. Randell, C. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Trans. on Dependable and Secure Computing, vol. 1, no. 1, 2004
[13] D. J. Lawson: Failure Mode, Effect and Criticality Analysis, Electronic System Effectiveness and Life Cycle Costing, J.K. Skwirzynski, ed., NATO ASI Series, F3, Springer-Verlag, Heidelberg, Germany, 1983, pp. 55–74
[14] Deutsche Gesellschaft für Qualität e. V. (DGQ): FMEA. Fehlermöglichkeits- und Einflussanalyse. Beuth-Verlag, Berlin, 2001.
[15] P.L. Goddard, Software FMEA techniques, Reliability and Maintainability Symposium, 2000. Proceedings. Annual , p. 118-123, 2000
[16] D.F. Haasl, Advanced Concepts in Fault Tree Analysis, presented at System Safety Symposium, 1965. Available at http://www.fault-tree.net/papers/haasl-advanced-concepts-in-fta.pdf.
[17] A. Birolini, Reliability Engineering: Theory and Practice, Springer, 2005.
[18] B. Schneier, Attack Trees: A formal, methodical way of describing the security of systems, based on varying attacks, Doctor Dobb's Journal, vol.24, p. 21–31, M&T publishing, 1999.
[19] H. Peine, M. Jawurek, S. Mandel, Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection, 11th IEEE High Assurance Systems Engineering Symposium, 2008
[20] IEC/DIN EN 61508, Funktionale Sicherheit sicherheitsbezogener elektrischer / elektronischer / programmierbar elektronischer Systeme, 2001
[21] www.vierfores.de, last visited: 2009-10-18
[22] Fault Tree Handbook, Systems and Reliability Research, Office of Nuclear Regulartory Research, U.S. Nuclear Regulartory Commission, Washington D.C., January 1981