Validating the Functional Design of Embedded Systems against Stakeholder Intentions

Marian Daun, Thorsten Weyer and Klaus Pohl
Paluno, The Ruhr Institute for Software Technology, University of Duisburg, Essen, Germany

Keywords: Functional Design, Correctness, Stakeholder Intentions, Behavioral Requirements, Embedded Systems.

Abstract: In the embedded systems industry, function-centered engineering is commonly applied to address the increasing number and complexity of system functions. During function-centered engineering, the functional design that is created based on the defined requirements for the system is the main artifact that serves as a basis for subsequent development activities. If stakeholder intentions change and modifications become necessary, they are frequently incorporated directly into the functional design without updating the behavioral requirements accordingly. As a consequence, the correctness of the interplay of system functions as defined in the functional design cannot be assessed by checking it against the defined requirements (since they are outdated) but needs to be checked against the current stakeholder intentions. More precisely, the requirements engineer has to validate the functional design against the stakeholder intentions because he is the expert concerning the stakeholder intentions and can communicate with the stakeholders regarding them, if necessary. However, the requirements engineer is typically not familiar with the functional design and its notation on the one hand, and, on the other hand, the overall behavior of the system is spread across various diagrams in the functional design. Therefore, the requirements engineer needs a more abstract and consolidated view of the functional design in order to be able to validate its correctness with regard to the current stakeholder intentions. In this paper, we present an approach which is based on a specific kind of review model that is automatically generated from the functional design and supports the requirements engineer in her task. The approach that is presented in this paper is subject of ongoing research.

1 INTRODUCTION

As described in (Pretschner et al. 2007) the paradigm of function-centered engineering is commonly applied in the embedded systems industry to address the increasing number and complexity of system functions. In function-centered engineering, the functional design is the core artifact within the entire engineering process. It serves as a basis for subsequent development activities like designing the system's software and hardware parts and determining the deployment of software functions to hardware components. Figure 1 illustrates a typical function-centered engineering process. In the first step (① in Figure 1), the behavioral requirements ([B] in Figure 1) of the system are defined. At first the stakeholder intentions ([A] in Figure 1) are elicited and consolidated. The stakeholder intentions refer to the required functionality and corresponding behavior of the system as well as to necessary qualities of the system (w.r.t. performance) and comprises constraints (w.r.t. legal regulations) that must not be violated by the system.

Based on the stakeholder intentions, the behavioral requirements are defined. In Step ②, the functional design ([C] Figure 1) is created based on the behavioral requirements. During this step, each system function is considered individually, i.e. its behavior as well as its functional interdependencies with other system functions are specified (cf. (Broy et al. 2009), (Beeck. 2007)).

However, stakeholder intentions may change after an initial version of the functional design has been created, for example, because stakeholders gain additional insights into solution aspects, while the functional design is created. Unfortunately, it is common practice during function-centered engineering processes that modifications resulting from such changed stakeholder intentions are incorporated directly into the functional design without updating the behavioral requirements.

During Step ② it also has to be ensured that the

Daun M., Weyer T. and Pohl K..
Validating the Functional Design of Embedded Systems against Stakeholder Intentions.
DOI: 10.5220/0004713103330339
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 333-339
ISBN: 978-989-758-007-9

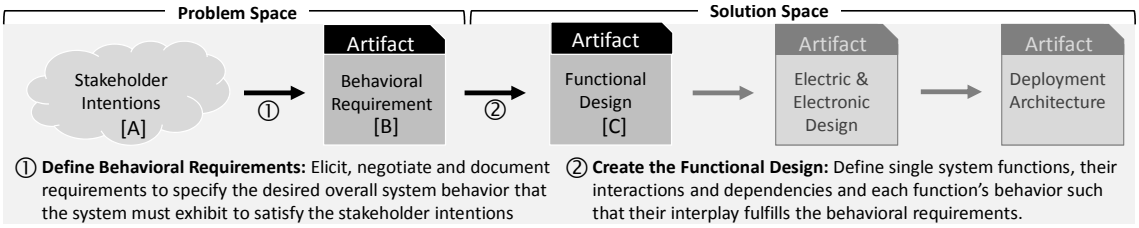

Figure 1: Function-centered engineering process.

interplay of system functions as defined in the functional design is correct. Yet, due to the circumstances described above, *correctness* here means that the functional design satisfies the current stakeholder intentions. Thus, correctness of [C] in Figure 1 has to be ensured against [A].

Existing approaches do not provide sufficient support regarding validation of the functional design of an embedded system against stakeholder intentions: Approaches dealing with model-driven development and model-transformations support the creation of a functional design from defined behavioral requirements ([C] from [B]). However, they are not applicable, since they do not take into consideration that stakeholder intentions might have changed after the requirements have been specified.

Approaches dealing with automated correctness checking only allow for checking [C] against [B], but not against [A]. Even if the behavioral requirements [B] were correct, automated approaches for checking design artifacts against requirements artifacts would not adequately support the correction of detected deficiencies (cf. (Borges, Garcez & Lamb. 2010)), since they only offer a single counterexample and do not make explicit where the detected deficiency is located in the original model.

Beside such automated approaches for correctness checking, manual review approaches have been evaluated as effective for checking and establishing correctness of a specification in general (cf. (Boehm & Basili. 2001), (Gilb & Graham. 1993)). Especially, perspective-based reviews conducted by the requirements engineer seem to be promising in our case (cf. (Basili et al. 1996)). Note that they have to be conducted by the requirements engineer and not the functional architect because the requirements engineer is the expert regarding the stakeholder intentions and can communicate with the stakeholders regarding them, if necessary. However, since the requirements engineer is usually not familiar with the functional design and its notation and the overall behavior of the system is spread across various diagrams in the functional design, the requirements engineer needs a more abstract and consolidated view of the functional design in order to be able to validate its correctness with regard to the stakeholder intentions.

In this paper, we suggest an approach which is based on a specific kind of *review model* that represents a consolidated view on the behavioral requirements and the functional design. The review model is documented in a notation that is frequently used to specify behavioral requirements for embedded systems. Thus, the requirements engineer can conduct the validation of the functional design against the stakeholder intentions based on a well-understood language.

The remainder of this paper is structured as follows. In Section 2, we discuss the related work. In Section 3, we sketch our solution approach and describe the underlying ideas in more detail. We illustrate our approach and its application by excerpts from an industrial case study on a *lane keeping system* in the automotive domain that we conduct to initially evaluate our solution idea. In Section 4, we draw conclusions and describe our future work.

2 RELATED WORK

A large number of approaches is proposed that address the correctness of design artifacts. In the following, we differentiate between four groups of techniques that offer potential solutions or could be part of potential solutions for achieving the correctness of the functional design.

I. Techniques supporting the Manual creation of Design Artifacts. The development of design artifacts from requirements is typically done manually to a large extent. To ensure correctness, rule-based or checklist-based approaches may be used (e.g. (Fagan. 1986) or (Leveson. 1995)). Note that manual approaches in general, especially if applied to large and complex systems, are highly time-consuming and lead to error-prone specifications (Arthur et al. 1999). According to our experience, these approaches are neither efficient nor effective for achieving the correctness of the functional design as regarded in this paper.

II. Techniques in Model-driven Development. Approaches dealing with model transformations (e.g. (Milicev. 2002)), model merging (e.g. (Abi-Antoun et al. 2008), (Sabetzadeh & Easterbrook. 2006)) and model synthesis (e.g. (Damas et al. 2009)) offer techniques for generating design artifacts from artifacts that were specified before in a consistent way. These approaches can potentially be used in our case to achieve correctness of the functional design by automating its creation from correct behavioral requirements (from [B] to [C] in Figure 1). However, the existing approaches have several deficiencies: First, they are generic and not model-specific, or only deal with the transformation between different kinds of behavioral models within the same engineering discipline (e.g. architectural design), for example, from interaction-based design models into state-based design models (e.g. (Uchitel, Brunet & Chechik. 2009), (Whittle & Jayaraman. 2010)). Second, they assume natural language requirements and do not support model-based requirements (Margaria & Steffen, 2009). Third, they do not consider the fact that new stakeholder intentions might have been elicited during development of the functional design and therefore, the behavioral requirements might be outdated.

III. Techniques for Automated Correctness checking. Performing correctness checks of design artifacts is typically done using model checking techniques (cf. (Clarke, Emerson & Sifakis. 2009), (Larsen. 1993), (Kupferman & Vardi. 2001), (Holzmann. 1997)). While approaches checking models of the same model type (cf. (Blanc et al. 2008)) cannot address correctness checking of the functional design, approaches dealing with different model types (cf. (Grundy, Hosking & Mugridge. 1998), (Fradet, Le Métayer & Périn. 1999)) could be used to check the functional design against the behavioral requirements ([C] against [B] in Figure 1). While model checking is widely used for verifying the correctness of design artifacts against a requirements specifications (commonly formulated in temporal logic), it suffers from insufficient support to resolve incorrect properties (Borges, Garcez & Lamb. 2010): The correctness checking tools in general provide the engineers with a single counterexample, not saying whether there are other incorrect properties in the design or not. Furthermore, the given counterexample in temporal logic or by a finite state machine would not support the requirements engineer in detecting and resolving a deficiency in the design, which is documented in a different notation. In addition, also these approaches do not address the issues of outdated behavioral requirements when validating the functional design.

IV. Techniques for Manual Correctness Checking. Within the field of manual approaches the relevant literature comes to the conclusion that performing reviews is in general the most effective manual technique (cf. (Boehm & Basili. 2001), (Gilb & Graham. 1993)). Many enhanced review techniques have been proposed and evaluated, e.g. checklist-based reviews (cf. (Fagan. 1986)), defect-class-based reviews (cf. (Porter, Votta & Basili 1995)), usage-based reviews (cf. (Abdelrabi et al. 2004) or perspective-based reviews (cf. (Basili et al. 1996)). Especially perspective-based reviews seem to be very effective (cf. (Shull et al. 2002)). By applying perspective-based reviews from a requirements engineering perspective to the functional design, the functional design cannot only be checked against the defined requirements ([C] against [B] in Figure 1). In addition, it can be checked against the stakeholder intentions ([C] against [A] in Figure 1). Since these techniques are conducted completely manually their application to large and complex specifications is highly time-consuming and error-prone.

3 SOLUTION APPROACH

Our approach combines several of the techniques discussed above to support the requirements engineer in conducting reviews of the functional design. We use, for instance, techniques from model-driven development to automatically generate a consolidated view of the defined behavioral requirements and the functional design that supports the requirements engineer in his task.

In addition, we use techniques for automated correctness checking in order to detect mismatches between the functional design and the defined (and maybe outdated) behavioral requirements.

As depicted in Figure 2, our solution approach distinguishes between four process steps (Steps ③ to ⑥) which have to be applied systematically in order to validate the functional design against the current stakeholder intentions and to correct the functional design, if necessary. In the following, we provide some insights into our solution approach. We first describe the two input artifacts in more detail. Afterwards, we explain the process steps.

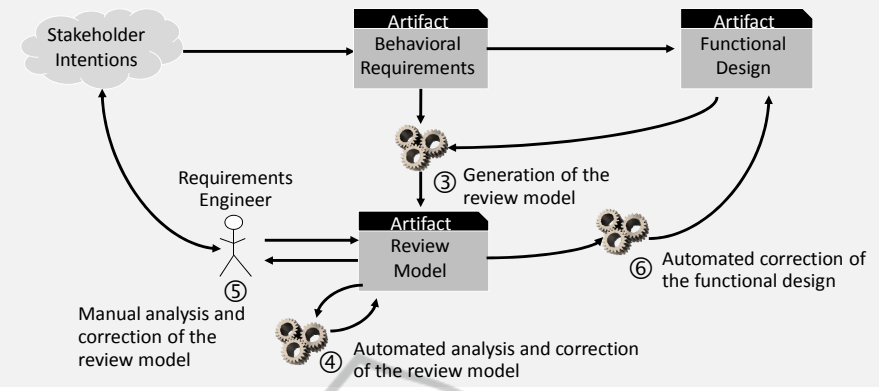

Figure 2: Overview of our solution approach.

3.1 The Behavioral Requirements

In general, behavioral requirements models can be differentiated into state-based and interaction-based models. During requirements engineering, especially interaction-based models are widely used, for example, to document scenarios and to specify the essential interfaces.

In the engineering of embedded software, message sequence charts (MSCs) are commonly used for the specification of interaction-based behavioral requirements models (cf. (Weber & Weisbrod. 2002)). The Z.120 standard (ITU. 2011) distinguishes between *basic message sequence charts* (bMSCs) and *high-level message sequence charts* (hMSCs). bMSCs define specific scenarios detailing the behavior in terms of messages exchanged between the system and entities in the environment. hMSCs structure the bMSCs according to their execution order and create a complete system specification.

The bMSC depicted on the left hand side of Figure 3 is an excerpt of the behavioral requirements specification for the lane keeping system. It specifies how the system should use video processing signals to determine unwanted deviations from the steering angle. More precisely, the diagram documents the behavioral requirement that the lane keeping system shall monitor the steering angle and check whether the steering angle will lead to departing the designated lane. The car's yawrate and a videostream of the lane are needed to determine whether the steering angle is acceptable or not.

3.2 The Functional Design

The functional design consists of specifications of the system functions to be implemented and their hierarchical structure. Additionally, the intended behavior of each system function is specified as well as the interactions and dependencies between system functions (cf. (Brinkkemper & Pachidi. 2010)).

Different diagram types are used to document the functional design. The hierarchical structure of the system functions is documented in *function hierarchy diagrams*. Feature trees may be used for that. The *function network diagram* documents the functional dependencies between system functions, which are embedded in given context functions. Context functions are functions that can be used by the system to be built, but are not subject of the development process. Afterwards each function is detailed by a *function behavior diagram* which specifies the behavior of the function in terms of an automaton (Alfaro & Henzinger. 2001). The right hand side of Figure 3 shows an excerpt of the functional design of the lane keeping system. The excerpt shows a functional network diagram and a corresponding function behavior diagram which specifies the behavior of the system function "Situation Evaluation".

In comparison to the behavioral requirements diagram on the left hand side, the functional design does, among others, not only document that a videostream and the yawrate should be used. Due to design decisions, it is specified that there are context functions dealing with video sensing and yawrate sensing and that these context functions shall be used.

3.3 Generation of the Review Model

To generate the review model, an overall behavioral model of the functional design has to be derived (③ in Figure 2). This is done by using the composition operator proposed in (Alfaro & Henzinger. 2001) and enhancing the operator in such a way that the dependencies and relations specified within a

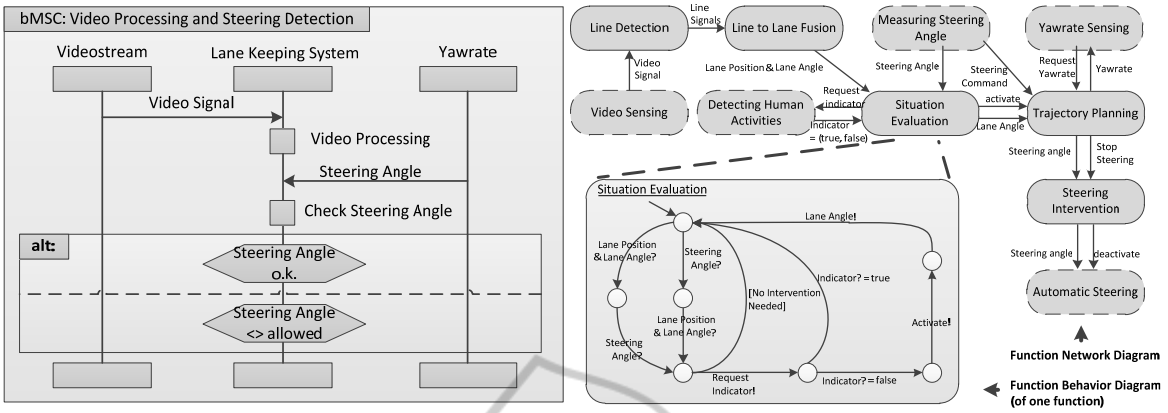

Figure 3: Excerpts from the behavioral requirements (left) and the functional design (right).

function network diagram are considered.

Thereafter, the refined interaction sequences of single bMSCs can be detected within the behavioral model of the functional design (e.g. by use of graph analysis (Cox, Delugach & Skipper. 2001)). The refined sequences are again documented in a new bMSC, this bMSC is a consistent refinement of the behavioral requirements that takes design decisions contained in the functional design into account. Figure 4 illustrates the generation of the review model based on the behavioral requirements and the functional design and sketches the procedural steps that have to be performed when generating the review model. The review model consists of MSC sets as shown in Figure 4. It is a refined version of the behavioral requirements that takes the information specified in the functional design into account. Thus, the bMSC shown in Figure 4 represents a refinement of the bMSC shown in Figure 3 with additional information from the functional design shown in Figure 3. The input signals in the functional design are mapped to concrete context functions derived from the function network diagram and the refined interaction sequence is derived from the function behavior diagrams.

Beside refined bMSCs, unrefineable bMSCs of the behavioral requirements are detected, marked as unrefineable and copied into the review model. In addition, interaction sequences contained in the overall behavioral model of the functional design that are not specified in the behavioral requirements have to be detected as well. These sequences are also transformed into bMSCs, which have to be marked specifically. To support the comprehensibility of the review model, parts of newly created bMSCs that are already represented by existing bMSCs are detected and discarded from the new bMSC. Thereafter, the hMSC is enhanced, such that the new bMSC is inserted at the correct position.

3.4 Automated Analysis and Correction of the Review Model

To detect error-prone parts in the review model which indicate deficiencies in the functional design or mismatches between the defined behavioral requirements and the functional design, several automated techniques can be used (④ in Figure 2). For example, techniques for estimating the probability of potential deficiencies which are based on pattern detection or graph analysis may be used. To support also the revision of the review model, techniques which derive a set of recommendations for correcting the review model can be used.

While applying automated techniques for analyzing the review model, emergent properties may be subject of investigation. For example, MSC-based specifications may contain implied interaction sequences (often referred to as implied scenarios (Uchitel, Kramer & Magee. 2001)). In this case, it is necessary that the requirements engineer decides whether these interaction sequences are desired or not. These decisions cannot be made automatically. However, the requirements engineer is supported since the implied sequences are detected and displayed to him/her. He/she may communicate with the stakeholders if necessary.

3.5 Manual Analysis and Correction of the Review Model

The requirements engineer analyses the review model to detect deficiencies that have not been detected by the application of the automated techniques (⑤ in Figure 2).

Special attention has to be paid to the unrefineable bMSCs as well as the new bMSCs derived from

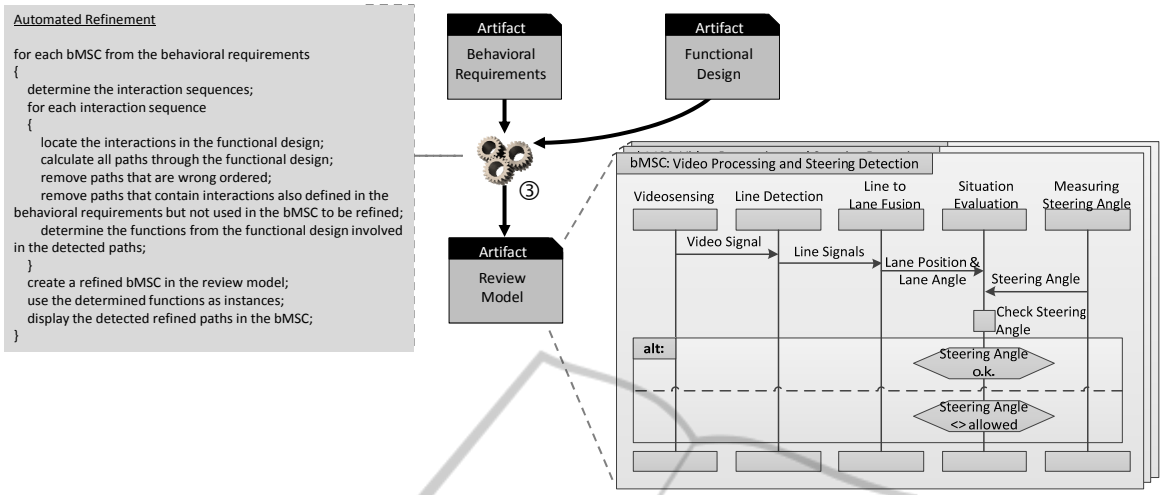

Figure 4: Generation of the review model with example (excerpt from lane keeping system case study).

the functional design, as they result from mismatches between defined behavioral requirements and the functional design.

When in doubt, the requirements engineer has to perform further elicitation and negotiation activities with the stakeholders to clarify their intentions in order to be able to decide whether specified behavior in the functional design has to be corrected to be in accordance with the stakeholder intentions. The corrections that are performed by the requirements engineer apply to both: deficiencies that are detected during the manual analysis of the review model and mismatches that were detected by the application of automated techniques but could not be corrected automatically.

Some automated analysis techniques that can be used in process step ④ derive propositions for correcting the review model. If these are applied, the requirements engineer can chose the best-fitting recommendation for correcting the review model. To do so, it can again be necessary to communicate with stakeholders to decide what the best recommended correction is.

3.6 Automated Correction of the Functional Design

The back-transformation from the corrected review model to the functional design can be performed by the use of model transformations techniques (⑥ in Figure 2). For each bMSC, a partial function network diagram is created. These partial diagrams are merged resulting in the final function network diagram. In addition, for each function, a function behavior diagram has to be derived. This can also be done using model-transformations as proposed in (Uchitel, Brunet & Chechik. 2009), or (Whittle & Jayaraman. 2010).

If incorrect or outdated behavioral requirements have to be corrected as well, composition techniques for MSCs as proposed in (Mauw & Reniers. 1999) and (Hélouët & Maigat. 2001) might be used to derive a corrected behavioral requirements specification. However, this is not within the scope of our approach.

4 CONCLUSIONS

The approach we presented in this paper addresses the validation of the correctness of the functional design in the sense that the requirements engineer can assess the interplay of systems functions that is specified in the functional design to validate the resulting system behavior against the current stakeholder intentions. We conducted an initial evaluation of our approach based on a case study of a lane keeping system in the automotive industry. From the case study, we gained first certain evidences concerning the applicability and usefulness of our approach. However, our approach is not finalized yet. So far, is only directly applicable if the behavioral requirements and the functional design are specified as described in this paper. We will examine the question of generalizability in further evaluation activities in the future. We intend to apply our approach to case studies from other domains like avionics and to more complex automotive systems.

ACKNOWLEDGEMENTS

This research was partly funded by the *German Federal Ministry of Education and Research* (BMBF), grant "SPES XTCore" (01IS12005C).

REFERENCES

Abdelrabi, Z., Cantone, E., Ciolkowski, M. & Rombach, D. (2004), Comparing code reading techniques applied to objectoriented software frameworks with regard to effectiveness and defect detection rate. *Proc. of the ISESE*, pp. 239-248.

Abi-Antoun, M., Aldrich, J., Nahas, N., Schmerl, B. & Garlan, D. (2008), Differencing and Merging of architectural views. *ASE Journal*, pp. 35-74.

Alfaro, L. & Henzinger, T. (2001), Interface Automata. *Proc. of the ESEC/FSE*, pp. 109-120.

Arthur, J., Göner, M., Hayhurst, K. & Holloway, C. (1999), Evaluating the effectiveness of independent verification and validation. *IEEE Computer*, October pp. 79-83.

Basili, V., Green, S., Lanubile, O., Laitenberger, F., Shull, F., Sorumgard, S. & Zelkowski, M. (1996), The empirical investigation of perspective-based reading. *Intl. J. Emp. SW. Eng.*, pp. 133-164.

Beeck, M. (2007), Development of logical and technical architectures for automotive systems. *Software Systems Modelling*, pp. 205-219.

Blanc, X., Mounier, I., Mougenot, A. & Mens, T. (2008), Detecting model inconsistency through operation-based model construction. *Proc. of ICSE*, pp. 511-520.

Boehm, B.& Basili, V. (2001), Software Defect Reduction Top 10 List." *IEEE Computer*, January, pp. 135-137.

Borges, R., Garcez, A. & Lamb, L. (2010), Integrating Model Verification and Self-Adaptation. *Proc. of the ASE*, pp. 317-320.

Brinkkemper, S. & Pachidi, S. (2010), Functional Architecture Modeling for the Software Product Industry. *Proc. of the ECSA*, pp. 198-213.

Broy, M., Gleirscher, M., Merenda, S., Kluge, D., Wild, P., & Krenzer, W. (2009), Toward a Hollistic and Standardized Automotive Architecture Description. *IEEE Computer*, pp. 98-101.

Clarke, E., Emerson, E. & Sifakis, J. (2009), Model checking: algorthmic verification and debugging. *Commun. ACM*, pp. 74-84.

Cox, L., Delugach, H. & Skipper, D. (2001), Dependency Analysis Using Conceptual Graphs. *Proc. of the ICCS*, pp. 117-130.

Damas, C., Lambeau, B., Roucoux, F. & van Lamsweerde, A. (2009), Analyzing Critical Process Models through Behaviour Model Synthesis. *Proc. of the ICSE*, pp. 441-451.

Fagan, M. (1986), Advances in Software Inspections. *TSE*, pp. 744-751.

Fradet, P., Le Métayer, D. & Périn, M. (1999), Consistency Checking for Multiple View. *Proc. of the ESEC/FSE*, pp. 410-428.

Gilb, T. & Graham, D. (1993), *Software Inspection.* Addison-Wesley.

Grundy, J., Hosking, J. & Mugridge, W. (1998), Inconsistency Management for Multiple-View Software Development Environments. *TSE*, pp. 960-981.

Hélouët, L., & Maigat, P. (2001), Decomposition of Message Sequence Charts. *SDL Forum*, pp. 348-364.

Holzmann, G. (1997), "The Model Checker SPIN." *TSE*, May, pp. 279-295.

ITU. (2011), *Recommendation Z.120.*

Kupferman, O., & Vardi, M. (2001), Model Checking of Safety Properties. *Formal Methods in System Design*, pp. 291-314.

Larsen, K. (1993), Efficient Local Correctness Checking. *Computer Aided Verification*, pp. 30-43.

Leveson, N. (1995), *Safeware: System Safety and Computers.* Addison Wesley, Reading.

Margaria, T., & Steffen, B. (2009), Continuous Model-Driven Engineering. *IEEE Comp.*, Oct., pp. 106-109.

Mauw, S., & Reniers, M. (1999), Operational Semantics for MSC'96. *Journal of Computer Networks*, June pp. 1785–1799.

Milicev, D. (2002), Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments. *TSE*, April, pp. 413-431.

Porter, A., Votta, L. & Basili, V. (1995), Comparing Detection Methods for Software Requirement Inspection: a Replicated Experiment. *TSE*, June pp. 563-575.

Pretschner, A., Broy, M., Kruger, I. & Stauner, T. (2007), Software Engineering for Automotive Systems: A Roadmap. *Proc. of FOSE*, pp. 55-71.

Sabetzadeh, M. & Easterbrook, S. (2006), View merging in the presence of incompleteness and inconsistency. *RE Journal*, pp. 174-193.

Shull, F. et al. (2002), "What we have learned about fighting defects." *Proc. of the Intl. Conf. SW Metrics*, pp. 133-154.

Uchitel, S., Brunet, G. & Chechik, M. (2009), Synthesis of Partial Behavior Models from Properties and Scenarios. *TSE*, May/June pp. 384-406.

Uchitel, S., Kramer, J. & Magee, J. (2001), Detecting Implied Scenarios in Message Sequence Chart Specifications. *Proc. of the ESEC/FSE*, pp. 74-82.

Weber, M. & Weisbrod, J. (2002), "Requirements Engineering in Automotive Development - Experiences and Challenges." *Proc. of the RE*.

Whittle, J. & Jayaraman, P. (2010), "Synthesizing Hierarchical State Machines from Expressive Scenario Descriptions." *TOSEM*, January, pp. 8:1-8:45.