Modelling Behavioural Requirements and Alignment with Verification in the Embedded Industry

Grischa Liebel[1,2], Anthony Anjorin[3], Eric Knauss[1,2], Florian Lorber[4] and Matthias Tichy[5]
[1] *Chalmers University of Technology, Gothenburg, Sweden*
[2] *University of Gothenburg, Gothenburg, Sweden*
[3] *Universität Paderborn, Paderborn, Germany*
[4] *Aalborg University, Aalborg, Denmark*
[5] *University of Ulm, Ulm, Germany*

Keywords: Requirements Modelling, Verification, Test Case Generation, Empirical Software Engineering, Model-based Engineering, Model-Driven Engineering.

Abstract: Formalising requirements has the potential to solve problems arising from deficiencies in natural language descriptions. While behavioural requirements are rarely described formally in industry, increasing complexity and new safety standards have renewed the interest in formal specifications. The goal of this paper is to explore how behavioural requirements for embedded systems can be formalised and aligned with verification tasks. Over the course of a 2.5-year project with industry, we modelled existing requirements from a safety-critical automotive software function in several iterations. Taking practical limitations and stakeholder preferences into account, we explored the use of models on different abstraction levels. The final model was used to generate test cases and was evaluated in three interviews with relevant industry practitioners. We conclude that models on a high level of abstraction are most suitable for industrial requirements engineering, especially when they need to be interpreted by other stakeholders.

1 INTRODUCTION

Requirements Engineering (RE) has a substantial impact on the success of software projects (Procaccino et al., 2002). Using natural language to specify requirements poses numerous challenges including avoiding ambiguity, inconsistencies, and contradictions. Perhaps even more problematic is the task of keeping such informal requirements in sync with corresponding acceptance tests and implementation artefacts. Textual natural language requirements specifications are, nonetheless, still the norm in the embedded industry (Sikora et al., 2011; Graaf et al., 2003).

A viable approach with the potential to solve at least some of these problems is to replace informal requirements with formal models, e.g., to enable automated analyses such as consistency checks and test generation. Although expressing requirements as formal models has been studied for a long time, e.g., in (Lubars et al., 1993), widespread adoption of formal methods in industry has not yet been achieved (Woodcock et al., 2009). Increasing complexity and new standards have, however, led to renewed interest in model-based requirements engineering in industry, as seen with joint industry-academia projects targeting model-based requirements engineering, such as CESAR (CESAR Project, 2011) or CRYSTAL (CRYSTAL Project, 2013). To improve the adoption of formal requirements models in industry, we report in this paper on our experience in an industry-academia project with the aim of exploring requirements modelling. Our work was guided by the following two research questions.

RQ1: How can formal models be used to capture real-world requirements?

RQ2: Do formal models for requirements provide added value in practice? What are the challenges involved?

Over the course of 2.5 years, we created several formal models of the requirements of a safety-critical Adjustable Speed Limiter (ASL) function from a real-life automotive requirements specification. One of the models was used for the automated generation of

Liebel G., Anjorin A., Knauss E., Lorber F. and Tichy M.
Modelling Behavioural Requirements and Alignment with Verification in the Embedded Industry.
DOI: 10.5220/0006205604270434
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 427-434
ISBN: 978-989-758-210-3
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

test cases, which we compared to existing, manually specified test cases, provided by our industrial partner. We complement our own reflections with data collected in multiple interviews and workshops conducted together with relevant professionals at our industrial partner. While we explored different formal notations, we report here on the use of *timed automata (TA)* (Alur and Dill, 1994) only, as most resources were spent on creating these models.

2 RESEARCH METHOD

The study presented in this paper follows the design science research paradigm, roughly following the guidelines of Wieringa (Wieringa, 2014) to structure, plan, and execute the study. The two main activities in design science research are the design and the investigation of an artefact in its context (Wieringa, 2014). These activities take place in two iterative cycles, the design cycle and the evaluation cycle. The outcome of each iteration is an evaluated artefact, a requirements model in the case of our study.

The design problem in our case was to create formal models of existing requirements that correctly represent these and enable further analyses. For each of the iterations, the corresponding knowledge questions were therefore "Does the artefact (requirements model) correctly represent the textual requirements?" and "What kind of analyses are enabled by the model?".

Our investigation was exploratory in nature. The project sponsor rather wanted us to sketch different solutions and possibilities using formal requirements models.

The context of the design science project is the current process at the case company, an automotive original equipment manufacturer (OEM), and a provided demonstrator system. Furthermore, the alignment of RE with verification was identified as an important aspect during initial discussions.

In this study, the model resulting from each iteration apart from the final iteration was evaluated analytically, without collecting empirical evidence. That is, we evaluated whether we succeeded in expressing the entire textual requirements specification as a model or not. Furthermore, the models were discussed multiple times in presentations with experts from the case company. The focus of these discussions was mainly to evaluate the adequacy and to plan the direction for future iterations.

The final artefact was evaluated empirically using three interviews with verification engineers at the case company. The focus of this final evaluation was on the interaction of the produced artefact with its context. That is, whether the artefact was appropriate in terms of abstraction, modelling notation, and enabled analyses, for an industrial application.

2.1 Validity Threats

We applied several measures in order to reduce the threats to validity in this study as much as possible.

To avoid misinterpretation of concepts and interviewer bias, we discussed and refined the interview guide[1] for the final iteration among the authors and conducted one pilot interview.

Selection threats cannot be ruled out as potential interviewees were selected from our network and the network of contact persons at the case company. While this is a potential threat to validity, the interviewees participated on a voluntary basis and were granted anonymity.

Design science instead aims at middle-range generalisations (Wieringa, 2014), which only make realistic assumptions about the object of study. As the case company selected a demonstrator system which they deemed representative for the case company, we expect that our findings apply to the context of the automotive domain. However, the low number of evaluation interviews could further limit this generalisability.

3 ADJUSTABLE SPEED LIMITER

The Adjustable Speed Limiter (ASL) is an active safety function of an automobile that automatically adjusts the current vehicle speed to keep to a target speed set by the user and can be overridden using the kickdown pedal. The function is part of the cruise control functionality that controls the engine and the brakes of a vehicle.

A schematic overview of the controls and variables of the function is depicted in Fig. 1. Typical controls comprise a series of buttons that can be attached to the steering wheel (depicted schematically to the left of Fig. 1 as buttons `Off`, `Resume`, `Preset`, `+`, and `-` on the steering wheel), as well as firmly depressing the accelerator pedal (depicted schematically to the left of Fig. 1 as a so called `kickdown` action).

A set of variables (depicted to the right of Fig. 1 as a rectangle containing dotted circles) is used by the ASL function to store: the current vehicle speed `current`, the currently set target speed `set`, and a preset value for the target speed `preset`. The expected

[1]The interview guide is available at http://grischaliebel.de/data/research/LAKLT_ModTCG.zip.

behaviour of the ASL is informally specified in the following with these controls and variables:

- Pressing `Off` deactivates the ASL.
- Pressing `Resume` activates the ASL, which immediately starts adjusting the current vehicle speed to keep to the target speed. If the target speed `set` is 0, however, the current vehicle speed `current` is first assigned to `set` before the ASL is activated.
- Pressing `Preset` has a similar effect, i.e., it activates the ASL, only that a preset speed `preset` is assigned to the target speed `set` before the ASL is activated.
- Pressing `+` (`-`) either activates the ASL with the current vehicle speed `current` as target speed or, when the ASL is already active, increments (decrements) `set`.
- Finally, the ASL can be temporarily deactivated by firmly depressing the accelerator pedal, referred to as a `kickdown`. This function is used, e.g., to temporarily taking control of the vehicle speed for overtaking another vehicle. A kickdown deactivates the ASL and starts a counter. The ASL is automatically reactivated after a timeout `t_max`.

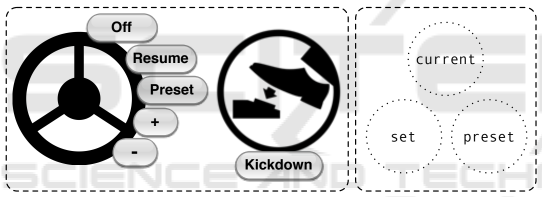

Figure 1: Events users can trigger (left) and variables of the system (right).

For our work on the ASL case study, we were provided with the following documents by our industrial partner:

1. A textual requirements specification for the ASL subsystem in the same detail as would be provided to a supplier. This specification consisted of 50 pages containing 40 functional (design) requirements and numerous requirements describing legal constraints and configuration options.
2. A set of 52 manual[2] system tests, developed manually to test the ASL function.
3. Presentations and other high-level documentation of the ASL function.

4 RESULTS

In the course of the project we specified multiple models representing the same requirements but on different levels of abstraction. Changing the level of abstraction was mainly driven by our and our stakeholders' needs.

[2]This means the tests must be *executed* manually.

4.1 Analyses with Uppaal

With the help of numerous clarifications with a domain expert at our industrial partner, we were able to formalise the provided requirement specification as a network of timed automata. Our primary goal with this step was to represent the requirements as faithfully as possible, without any simplifications. This was indeed possible using timed automata as supported by Uppaal, and resulted in a relatively low-level network consisting of 13 automata with a total of 43 states in all automata. While we aimed for having a one-to-one mapping between requirements and single timed automata, this was not always possible or feasible. For example, requirements addressing the deactivation of the ASL system had to be incorporated into all automata addressing ASL activation, as they had to return to their initial states.

Although our results indicate that timed automata as a formalism was expressive enough to capture exactly what was informally specified, the resulting model was complex and could not be used for any manual checks including manual reviews and inspections to check for conformance. This was mainly due to the large amount of system and external environment variables that had to be provided and understood, e.g., the different engine and vehicle states.

Apart from having a high complexity in itself, the resulting model required a suitable environment model in order to be simulated properly. For example, information about the automobile being switched on or off would have been needed. The fact that we did not have such a model made the requirements model difficult to use for automated checks. This is an interesting observation for industrial practice as well, as a supplier would lack environment information in a similar fashion.

To simplify communication within our team and make progress towards other goals, we derived a further network of timed automata on a medium level of abstraction.

The medium-level model we obtained consisted of 8 automata and 15 states, and was now better suited for manual and semi-automated tasks, which require a deep understanding of the modelled functionality. We performed semi-automated conformance checks using the test cases we had, by implementing a simulator in Uppaal by additionally specifying an environment model.

4.2 Manual Inspection

To achieve our goal of checking the adequacy of the provided tests, we derived a final, high-level model of the requirements, this time as a single timed automaton with only three states. Concerning consistency, we were able to perform manual inspections and reviews in the team, discussing how exactly the requirements are to be interpreted and identifying particularly confusing aspects. This iterative review and refinement process identified two issues that were verified as outdated aspects of the requirements by our industrial partners.

4.3 Manual Conformance Checks

We were able to manually test the conformance of our high-level model to the provided tests by converting the tests to the same formalism as the high-level model (effectively formalising the test cases as timed automata). To do this, however, we had to exclude all tests that covered aspects abstracted away in our model and were left with 9 tests from an initial 52 . As a comparison, 18 tests could be executed on our medium-level model, 40 tests on our low-level model, and 12 tests were entirely out of scope of our requirements specification, as they addressed parts not covered in the provided requirements specification, e.g., GUI elements.

Figure 2 depicts one of these 9 provided tests, testing in this case a combination of `+` and `Preset`. The test is also depicted in the UML statechart visual concrete syntax and is to be interpreted as follows: the states are expected states of the System Under Test (SUT), i.e., if these states do not match then the test fails. This means that a precondition of the test is that the SUT is deactivated. Events in the test are stimuli for the SUT, e.g., the event `+` simulates a user pressing the `+` button (and the SUT is expected to transition to `Limiting`). Conditions such as `[set == preset]` are assertions on the variables of the SUT. A test passes if it reaches the final state `PASS` and fails otherwise (a suitable timeout is used to enforce termination).

In this manner, we were able to manually verify that our high-level model conforms to at least these 9 tests. Via this manual conformance check, we were able to identify and correct "mistakes" in our model, mainly due to a mismatch between tests and requirements, which could also be verified by our industrial partners.

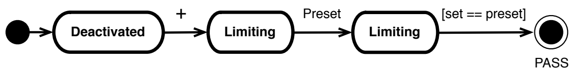

Figure 2: High-level model of a provided test case.

4.4 Mutation-based Test Generation

Our final high-level model was not only simple enough to enable manual inspections and conformance checks, but could also be handled by MoMuT::TA and used for mutation-based test generation. Mutation-based test generation involves the creation of a set of intentionally faulty models, called *mutants*, according to a set of fault models. Traces that lead to situations in which the mutants behave differently than the original model are then searched for, and test cases are generated based on these traces. The resulting test suite guarantees the absence of the inserted faults, if they run through on a deterministic SUT.

We performed mutation-based test case generation using MoMuT::TA as follows:

1. After filtering and formalising all tests that could be run for the high-level model, this set of tests was used as an initial "seed" for MoMuT::TA. This means that the tool is forced to start from this possibly sub-optimal set of tests and generate additional tests to achieve full coverage of all mutants. By also performing the test case generation without the initial seed, one can create a minimal set of test cases and thus detect whether some of the initial tests are redundant.
2. The high-level model was supplied to MoMuT::TA as the input model and used to generate a set of mutants. Note that this set of mutants was based on general experience with test case generation and not on specific domain knowledge of the case study.
3. In our case, MoMuT::TA classified all 9 provided tests as necessary, and generated an additional 11 tests to achieve full coverage (to kill all generated mutants).
4. We then inspected these additionally generated 11 tests and their corresponding mutants manually, compared them with the initially provided 9 tests, and attempted to understand why they were necessary.

From our comparative analysis, we determined that the generated tests were largely of comparable size and complexity as the provided, manually specified tests, and we were able to identify three main sources of additionally generated tests:

Self-transitions in the Timed Automata. Self-transitions were implicit in the requirements and were not tested explicitly, i.e., were completely missing in the provided tests. The additionally generated tests for such transitions appear useful and sensible, and the corresponding mutants represent interesting, test-worthy cases. 6 tests were generated in total as a result of such explicit self-transitions in the model.

Missing Combinations. An additional 3 tests were generated due to missing *combinations* of certain events that were simply not covered by the provided tests. Note that our test generation approach is not combinatoric, i.e., we did not employ a brute force combination of all possibilities. This means that in all 3 of these cases, we could always inspect the corresponding mutant to understand why this particular combination was chosen by the test case generator.

Missing Tests. The 2 remaining tests were generated for elements in the model that were simply not at all covered by any provided test. Again the potential of automatically reacting to changes in the requirement model is to be noted here, and not the particular aspects identified as being missing.

5 EVALUATION

In order to evaluate the resulting TA model and the generated tests with respect to suitability for industrial use, we conducted three semi-structured interviews with professionals at the industrial partner. As the interviewees needed to be able to compare the model and the generated tests to the current baseline, we chose subjects from the area of verification on system level. The interviews were aimed at answering the following interview questions which together address our second research question RQ2:

IQ1. What would be the potential of model-based requirements and of test case generation to the case company?

IQ2. Are the tests generated from the TA model comparable to existing manual tests?

IQ3. What would be the primary challenges involved in introducing this technique at the case company?

Interviews started with a short introduction of our activities, the resulting TA model, and the generated tests. The remaining questions were grouped into three categories addressing the generated tests (IQ2), model-based requirements in general (IQ1 and IQ3), and the usefulness of potential features of test case generation for industry (IQ1 and IQ3).

5.1 Potential of Model-based Requirements and Test Case Generation

All interviewees stressed that the presented approach of formalising requirements in an abstract automata model is highly interesting and relevant to the case company. One interviewee stated that replacing natural language requirements with models is "Something I'm actively working towards today. But it takes a lot of convincing.". Similarly, another interviewee stated "As long as you can create logical requirements, such a model should be the solution.".

While we expected the abstract timed automata model, which we used to generate tests, to be much too simple and lacking too much information, all interviewees stated that this level of abstraction is indeed relevant and useful for system testing. Furthermore, one interviewee mentioned that the existing manual system tests might even be too detailed, as they are derived from design requirements and therefore contain lots of additional information.

5.2 Generated vs. Manual Tests

Regarding the tests addressing self-transitions in the automata model, the interviewees stated that these could be relevant for regression testing. However, they would have to be automated in order to reduce the testing effort and would probably not be prioritised if test execution must be performed manually. In fact, multiple interviewees stated that the case company had such tests in the past, but removed them due to time constraints once the system became stable. They all agreed that once test execution can be fully automated, this class of tests would indeed add value and should be incorporated again into the test suite.

The second class of generated tests, combinations of behaviour tested in the manual tests, was evaluated differently by the interviewees. While they found them generally useful, one interviewee stated that many of these combinations are tested indirectly due to setup routines. For example, a test case might require the ASL function to be active, which requires a setup that activates this function first. Such a setup followed by a test case could then be equivalent to one of the "missing" combinations.

The third class of tests, functionality not at all tested in the manual tests, was considered very useful by the interviewees. Even though we pointed out that some of these missing tests might have been removed due to our abstraction, interviewees stated that especially the feedback that such a test is missing is useful. Given this information, verification engineers can then judge whether it needs to be added, is already covered on a lower level of abstraction, or is not relevant.

Regarding the issues we found with the existing requirements and system tests, interviewees stated that they were all well aware that requirements can be wrong or outdated. One interviewee stated: "We report everything that we think is bad with the requirements. But since it is a huge amount of issues, it is difficult to handle them all.". However, another interviewee added that finding these issues could be useful as it is difficult to identify such mismatches between tests and requirements "if you only look at high-level requirements". You would instead need to consider "all (detailed) requirements together and see the whole picture". Essentially, this quote describes very well the process we went through in the different iterations.

One interviewee stated that visualising mutants that are killed by a test case (which faulty behaviour it tests) could be very useful in the context of safety-critical functions. Test cases could then be ranked according to severity, e.g., according to the criticality level of the tested function. This could help when prioritising requirements, for coverage analysis, and for choosing a (small) subset of tests that can be executed manually for given resources. A study showing how visual representations of timed automata mutants can be used for debugging has already been conducted (Aichernig et al., 2014) and the same technique could also be applied in the case company.

5.3 Challenges of Model-based Requirements and Test Case Generation

As challenges the presented approach faces, the interviewees named mainly the effort to create requirements models, and industrial tool support. Interestingly, the former challenge is not related to the complexity of creating models and the knowledge to do so, but concerns rather the costs of formalising existing requirements. When asked about the challenges of introducing model-based requirements and test case generation, one interviewee stated that the challenge is "First of all, to have the time to do it.".

Regarding tooling, interviewees stated that it is essential to integrate the chosen modelling tool into the existing tool chain. Furthermore, professional support for this tool and test case generation must be provided.

One interviewee mentioned that it could be a problem to get support from management, stating that it is difficult "To have the people with the budget aware of what we get if we do this.". Another interviewee mentioned lacking support from developers as an issue, stating that they would not see the problems with textual requirements: "Not everyone thinks like that. Especially developers without a testing background, as you don't encounter the problems without having the exactly defined behaviour.".

None of the interviewees did however express a concern that formal methods could be difficult to apply given an engineers' typical education. This could be attributed to the fact that many of the engineers in this domain are used to modelling notations such as Simulink. This is an interesting result as training effort is commonly mentioned in the related work as a challenge in model-driven and model-based techniques, e.g., in (Agner et al., 2013; Hutchinson et al., 2011; Liebel et al., 2016).

While we were especially concerned with establishing trust in generated tests, the interviewees did not share this concern to the same extent. One person stated that "I'd say from a testing point of view there are certain trust issues in some places. But generally, I would say that we are for a model-based approach. Definitely.". They did however all agree that it would be helpful to be able to understand on an abstract level what is tested by a test case and that visualising mutants could be helpful (and this of course increases trust in the approach).

6 RELATED WORK

Using models to specify requirements has been explored extensively over the last decades. In the context of automotive and embedded software, several studies report on experiences with modelling languages that prescribe the structure of the requirements specification. For example, Boulanger and Văn describe a methodology to develop embedded automotive systems, using EAST-ADL and SysML for requirements modelling (Boulanger and Văn, 2008). Similarly, Piques and Andrianarison report on industrial experiences with using SysML in the automotive domain (Piques and Andrianarison, 2011). Albinet et al. introduce a similar approach, but use the UML profile MARTE for real-time systems (Albinet et al., 2008). All three approaches have in common that the requirements themselves are expressed in natural language and only the structure of the specification is prescribed by the modelling language.

For expressing requirements as models, different languages have been used in the embedded domain, for example, formal specification languages as in (Zave, 1982) and (Broy et al., 1992). Such languages are, however, often aimed at system design and are on a too detailed level to be considered requirements. In some cases, they could nonetheless be on a level of abstraction comparable to our first, low-level requirements model.

An early empirical study by Lubars et al. (Lubars et al., 1993) also reports that languages such as Entity-Relationship diagrams, flow diagrams, and object-oriented models were already commonly used in RE during the early 90s. The popularity of models at that time could partly stem from the widespread use of methods and processes such as the Rational Unified Process and Structured Analysis, which both include the use of models during RE. However, a more recent study by Sikora et al. (Sikora et al., 2011) reports that practitioners in the embedded industry advocate a more intensive use of models during RE. The authors attribute this to the automation possibilities that RE models could offer.

More recently, Böhm et al. (Böhm et al., 2014) used the tool AutoFOCUS3 in an industrial project to model requirements of a train automation system. Braun et al. (Braun et al., 2014) use several different models to express automotive requirements, e.g., goal models.

Regarding the area of test case generation, timed automata have been used for test case generation multiple times. Nielsen and Skou (Nielsen and Skou, 2001) propose a test case generation framework for timed automata. The UPPAAL tool chain contains the tool Tron (Mikucionis et al.,) performing online testing and the tool CoVer (Hessel and Pettersson, 2007) that allows coverage-based offline test generation from timed automata. However, these approaches are not mutation-based. Mutation-based test case generation has been performed on several different formalisms, including timed automata (Aichernig et al., 2013), pushdown automata (Belli et al., 2012), UML state machines (Aichernig et al., 2015a) and requirement interfaces (Aichernig et al., 2015b).

All approaches presented above, academic or not, have in common that the contribution is the resulting model, generated tests, or a discussion thereof. In contrast to this, we discuss the iterative process using different languages and levels of abstraction to formalise real-life requirements from the automotive domain and generate tests from them. Additionally, we provide empirical evidence in the form of qualitative data gathered from interviews with industrial practitioners. The focus is therefore on the entire process and the lessons learned throughout it, in contrast to single steps in the chain.

7 CONCLUSION AND FUTURE WORK

In this paper, we reported on the outcome of a 2.5-year project carried out together with an automotive OEM with the goal of exploring the potential of model-based requirements engineering in the embedded domain. Over the course of the project, we created several models on different levels of abstractions of a real-life, safety-critical requirements specification. Using timed automata as modelling language, we created three different models, going from the lowest level of abstraction, directly translating the design requirements, to a single automaton with only three states. We used this final automaton to generate test cases and evaluated the overall outcome with system-level verification engineers at our industrial partner.

Our evaluation indicates that it is indeed possible to use formal models to capture real-world requirements from the automotive domain, thus answering our first research question. Based on our experience, models on a very high level of abstraction appear to be especially suitable for detecting inconsistencies in requirements and for generating useful test cases.

Our interviewees indicated that the approach is highly interesting and relevant, and that they are in fact currently planning to introduce a model-based approach for requirements engineering and connected to verification, answering our second research question. As challenges, the interviewees see the effort to create requirements models of an existing requirements base, appropriate tool support and tool integration, and the (lack of) support from management and developers.

ACKNOWLEDGEMENTS

The research leading to these results has received partial funding from the European Union's Seventh Framework Program (FP7/2007-2013) for CRYSTAL-Critical System Engineering Acceleration Joint Undertaking under grant agreement N^{o} 332830, from Vinnova under DIARIENR 2012-04304, and from the Austrian Research Promotion Agency (FFG) under grant agreement N^{o} 838498.

REFERENCES

Agner, L. T. W., Soares, I. W., Stadzisz, P. C., and Simão, J. M. (2013). A brazilian survey on UML and model-driven practices for embedded software development.

Journal of Systems and Software, 86(4):997–1005. {SI} : Software Engineering in Brazil: Retrospective and Prospective Views.

Aichernig, B. K., Brandl, H., Jöbstl, E., Krenn, W., Schlick, R., and Tiran, S. (2015a). Killing strategies for model-based mutation testing. *Software Testing, Verification and Reliability*, 25(8):716–748.

Aichernig, B. K., Hörmaier, K., and Lorber, F. (2014). Debugging with timed automata mutations. In *Computer Safety, Reliability, and Security - 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings*, pages 49–64.

Aichernig, B. K., Hörmaier, K., Lorber, F., Nickovic, D., and Tiran, S. (2015b). Require, test and trace IT. In *Formal Methods for Industrial Critical Systems - 20th International Workshop, FMICS 2015, Oslo, Norway, June 22-23, 2015 Proceedings*, pages 113–127.

Aichernig, B. K., Lorber, F., and Nickovic, D. (2013). Time for mutants - model-based mutation testing with timed automata. In *Tests and Proofs - 7th International Conference, TAP 2013, Budapest, Hungary, June 16-20, 2013. Proceedings*, pages 20–38.

Albinet, A., Begoc, S., Boulanger, J., Casse, O., Dal, I., Dubois, H., Lakhal, F., Louar, D., Peraldi-Frati, M., Sorel, Y., et al. (2008). The memvatex methodology: from requirements to models in automotive application design. In *4th European Congress on Embedded Real Time Software (ERTS '08)*.

Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical computer science*, 126(2):183–235.

Belli, F., Beyazit, M., Takagi, T., and Furukawa, Z. (2012). Model-based mutation testing using pushdown automata. *IEICE Transactions*, 95-D(9):2211–2218.

Böhm, W., Junker, M., Vogelsang, A., Teufl, S., Pinger, R., and Rahn, K. (2014). A formal systems engineering approach in practice: An experience report. In *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*, SER&IPs 2014, pages 34–41.

Boulanger, J.-L. and Văn, Q. D. (2008). Requirements engineering in a model-based methodology for embedded automotive software. In *IEEE International Conference on Research, Innovation and Vision for the Future (RIVF '08)*.

Braun, P., Broy, M., Houdek, F., Kirchmayr, M., Müller, M., Penzenstadler, B., Pohl, K., and Weyer, T. (2014). Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science - Research and Development*, 29(1):21–43.

Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. F., and Weber, R. (1992). *The design of distributed systems: an introduction to Focus*. Technische Universität München.

CESAR Project (2011). CESAR - Cost-efficient methods and processes for safety relevant embedded systems. http://www.cesarproject.eu. last accessed Mar. 2016.

CRYSTAL Project (2013). CRYSTAL - CRitical sYSTem engineering AcceLeration. http://www.crystal-artemis.eu. last accessed Jan. 2016.

Graaf, B., Lormans, M., and Toetenel, H. (2003). Embedded software engineering: the state of the practice. *IEEE Software*, 20(6):61–69.

Hessel, A. and Pettersson, P. (2007). Cover-a test-case generation tool for timed systems. *Testing of Software and Communicating Systems*, pages 31–34.

Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011). Empirical assessment of mde in industry. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 471–480.

Liebel, G., Marko, N., Tichy, M., Leitner, A., and Hansson, J. (2016). Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling*, pages 1–23.

Lubars, M., Potts, C., and Richter, C. (1993). A review of the state of the practice in requirements modeling. In *IEEE International Symposium on Requirements Engineering (RE '93)*, pages 2–14.

Mikucionis, M., Nielsen, B., and Larsen, K. G. Real-time system testing on-the-fly. In Sere, K. and Waldén, M., editors, *NWPT 2003*, number 34 in B, pages 36–38. Abo Akademi, Department of Computer Science, Finland.

Nielsen, B. and Skou, A. (2001). Automated test generation from timed automata. In *TACAS 2001, held as Part of ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, pages 343–357.

Piques, J. and Andrianarison, E. (2011). Sysml for embedded automotive systems: lessons learned. *Interfaces*, 3:3b.

Procaccino, J. D., Verner, J. M., Overmyer, S. P., and Darter, M. E. (2002). Case study: factors for early prediction of software development success. *Information and Software Technology*, 44(1):53–62.

Sikora, E., Tenbergen, B., and Pohl, K. (2011). Requirements engineering for embedded systems: An investigation of industry needs. In Berry, D. and Franch, X., editors, *Requirements Engineering: Foundation for Software Quality*, volume 6606 of *Lecture Notes in Computer Science*, pages 151–165.

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer.

Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36.

Zave, P. (1982). An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, SE-8(3):250–269.