Improving Model-Based Testing in Automotive Software Engineering

Stefan Kriebel, Matthias Markthaler, Karin Samira Salman
Development Electric Drive
BMW Group
Germany
www.bmw.de

Timo Greifenberg, Steffen Hillemacher, Bernhard Rumpe, Christoph Schulze, Andreas Wortmann
Software Engineering
RWTH Aachen University
Germany
www.se-rwth.de

Philipp Orth, Johannes Richenhagen
FEV Europe GmbH
Germany
www.fev.com

ABSTRACT

Testing is crucial to successfully engineering reliable automotive software. The manual derivation of test cases from ambiguous textual requirements is costly and error-prone. Model-based development can reduce the test case derivation effort by capturing requirements in structured models from which test cases can be generated with reduced effort. To facilitate the automated test case derivation at BMW, we conducted an anonymous survey among its testing practitioners and conceived a model-based improvement of the testing activities. The new model-based test case derivation extends BMW's SMArDT method with automated generation of tests, which addresses many of the practitioners' challenges uncovered through our study. This ultimately can facilitate quality assurance for automotive software.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; Unified Modeling Language (UML);

KEYWORDS

Model-Based Testing, Test Case Creation

ACM Reference Format:
Stefan Kriebel, Matthias Markthaler, Karin Samira Salman, Timo Greifenberg, Steffen Hillemacher, Bernhard Rumpe, Christoph Schulze, Andreas Wortmann, and Philipp Orth, Johannes Richenhagen. 2018. Improving Model-Based Testing in Automotive Software Engineering. In *Proceedings of 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3183519.3183533

1 MOTIVATION

Modern vehicles are complex systems in which multiple components interact over dozens of controllers to provide system-wide software functions. Future challenges, such as autonomous driving or CAR2X communication, do not only increase the complexity for one system, but introduce an additional layer of complexity by requiring interactions between multiple systems in an open environment. Fig. 1 illustrates this: In the past, automotive engineering challenges mostly resided in mechanical engineering and electrical engineering (*e.g.,* transmission or power control), whereas an increasing number of present and future challenges can be found in software engineering (*e.g.,* connectivity, autonomous driving or gesture HMI).

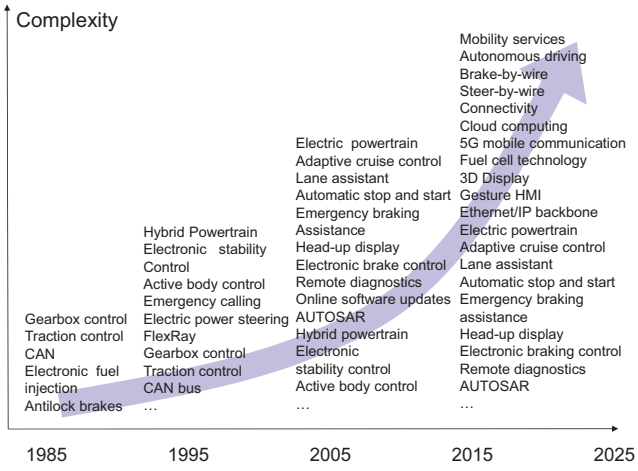

Figure 1: Complexity drivers in automotive [6].

Moreover, customers demand for a shorter time-to-market, dropping from average values of five years in the 1980s to three years in the 21st century [20], while at the same time requesting more functionality at similar costs. Considering inflation, this requires a cost reduction of 4% every year [17], which makes reducing the cost and time of software development a prime concern, when mastering growing complexity at the same time.

Considering these challenges, existing processes and business models for component-oriented development reach an impasse. To achieve shorter development cycles under high cost pressure, new approaches are required to provide safe and affordable mobility solutions.

ICSE-SEIP '18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183533

Similar challenges are being addressed successfully in other domains through model-based development [7] and model-based testing [5] (MBT), which aims at leveraging more abstract, domain-specific, and reusable models as primary development artifacts to reduce the conceptual gap between the problem domains (*e.g.,* automated driving) and the solution domains (*e.g.,* software engineering). For successful deployment to automotive software engineering, these modeling techniques need to be adapted and extended to specific requirements on safety and process standard compliance, while retaining their agile efficiency.

We identified potentials for improving the test creation process at BMW through a systematic survey. Based on these findings, we conceived adaptions on the test creation process to enable increasing the development quality upfront. This frontloading technique enables function specifiers and test specifiers to cooperate much earlier in the development process and work on the same artifacts, which can be used to systematically derive test cases. The contributions of this paper, hence, are

- results from a survey on test case derivation at BMW,
- a methodology for automated, model-based automotive software testing, and
- its implementation with the language workbench MontiCore [9].

In the following, Sect. 2 introduces preliminaries on model-based testing at BMW, before Sect. 3 describes design, execution, and results of our survey at BMW. Sect. 4 outlines our concept for improving model-based testing based on the survey results and Sect. 5 describes an implementation of our concept. Sect. 6 reports lessons learned from applying this concept at BMW and Sect. 7 discusses related work. Finally, Sect. 8 concludes.

2 PRELIMINARIES

The coming into effect of the ISO 26262 demanded a new specification method for safety-relevant automotive functions. Therefore, the *Specification Method for Requirements, Design and Test (SMArDT)*[1] was conceived at BMW. Initially, the functional safety departments used SMArDT to fulfill the ISO 26262. Later the method was adopted for non-safety driving assistance functions as well. Nowadays, SMArDT is used in further departments for automotive software quality assurance in the development of vehicle dynamics, electrical powertrain, and autonomous driving functions. For implementing the method, the System Modeling Language (SysML) [8] was chosen due its prevalence and support through professional software tools.

SMArDT describes a semi-formal specification for requirement, design, and testing of systems engineering artifacts according to the ISO 26262 specifications. Four abstraction layers structure the method:

(1) The first layer contains a first description of the function under consideration and shows its boundaries from a customers point of view.
(2) The second layer contains functional specifications without details of their technical realizations.

[1]The abbreviation SMArDT is related to the German term " **S**pezifikations-**M**ethode für **A**nfo**r**derung, **D**esign und **T**est" (specification method for requirements, design, and test)

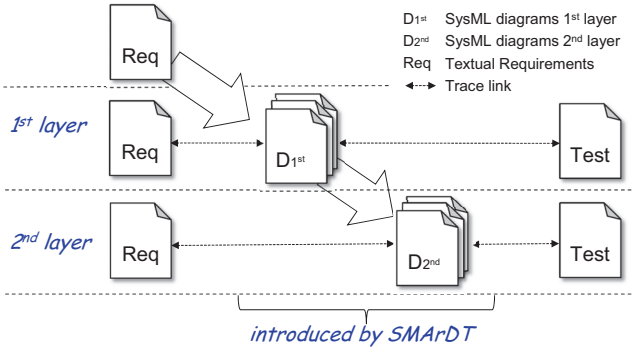

Figure 2: SMArDT artifacts of the first two layers.

(3) The third layer details technical concepts with its logical signals for the implementation including software architecture, architecture of the on-board power network, and a deployment mapping between these.
(4) The fourth layer represents the software and hardware artifacts present in the system's implementation.

The first two abstraction layers are conceptual in the sense that their diagrams lack a direct counterpart in the implementation. The behavior modeled within the diagrams can later be implemented across several components. Moreover, signals used in these diagrams are logical, *i.e.,* they abstract signals of the implementation. Consequently, corresponding values comprise a range of values present in the implementation. In contrast, the elements of the third and fourth layers have a direct representation within the implementation.

Our work focuses on the first two layers of SMArDT. In addition to the preexisting textual requirements and test artifacts, further artifacts are introduced by SMArDT as shown in Fig. 2. The input for the first abstraction layer are textual-conceptual requirements in natural language (short: textual requirements). These requirements can be further specified and refined by SysML structure and behavior diagrams, *e.g.,* use case diagrams and composite structure diagrams. Output of this layer are the modeled diagrams. In the second abstraction layer, a functional concept is developed. Based on the output requirement artifacts of the first layer, the functional-conceptual requirements are represented by more specified structure and behavior diagrams, *i.e.,* activity diagrams (ADs). These diagrams are the output of the second layer.

SMArDT facilitates clarifying vague customer requirements as well as functional, subsystem and system specifications. The natural language-based requirements, which are often complex and incomprehensible on their own, are complemented by diagrams. Moreover, SMArDT facilitates tracing across all and within single abstraction layers from textual requirements to derived tests. This helps finding missing specifications in the textual requirements by identifying model elements that are unrelated to textual requirements.

As SMArDT entails that requirement models are present, the basis for model based testing is given. Even though SMArDT describes the artifacts to be engineered and linked to each other, a fine grained methodology for engineering test cases from diagrams

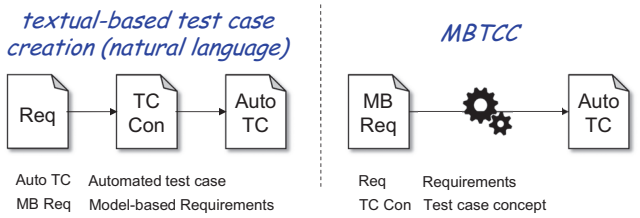

Figure 3: Textual vs. model-based test case creation.

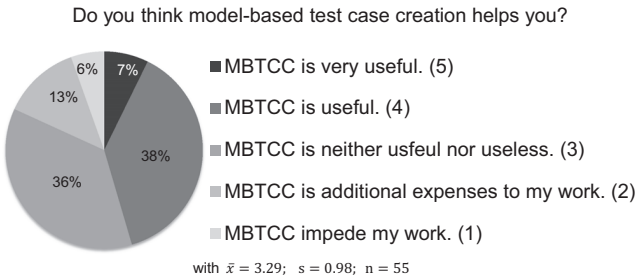

Figure 4: Expected benefits from MBTCC.

is not yet a part of SMArDT. In the next section the potential of model-based testing from the test engineers' point of view at BMW is investigated.

3 INVESTIGATING THE STATE OF TESTING

SMArDT aims, among others, to address challenges in requirements engineering by introducing requirements models on several abstraction levels. Despite the benefits traditionally claimed by model-based development (*e.g.,* better comprehension, analysis, transformation), its potentials in model-based test case creation (MBTCC) in automotive software engineering are yet to be discovered. To this end, we conducted a systematic survey on MBTCC at BMW. This section describes our research questions, the survey's design, and its results.

3.1 Research questions

To discover the potential for MBTCC based on SMArDT, we aim to probe who would benefit from employing this method at BMW as mandating model-based technologies top-down has proven futile in the past [27]. Similarly, we try to identify whether MBTCC could improve the test coverage and quality of the individual test cases. Moreover, we want to understand which environments are most suitable for MBTCC, the test bed, subsystems of the vehicle or the complete vehicle. Besides natural language-based and model-based test cases, the related processes can be compared (see Fig. 3).

The starting point for natural language-based test case creation in the BMW context are textual requirements. A test engineer develops a *test case concept* based on these (Fig. 3). The test case concept includes the test objective and the required basic test steps. Based on the test case concept an *automated test case* is implemented. Automated MBTCC can directly generate executable automated test cases. Consequently, the research questions our survey investigates:

- **R1** What are the test engineers' backgrounds? Who can benefit from the (semi-)automatic MBTCC?
- **R2** Is it possible to enhance the test quality and the test coverage by MBT? (from the test engineers' point of view)
- **R3** Which environments are the most promising for model-based testing in the BMW context?

The main subject of the survey is to investigate the potential of MBTCC in comparison to the textual requirements-based test case creation prevalent in automotive software engineering. To this end, on **R1**, the questionnaire requests the participants' main function, working background, years in the automotive industry, years of experiences in test case creation, the focus of testing, and the opinion on MBT. This data enables us to conclude from the participant's profiles to their indications. To clarify **R2** and **R3**, we interrogate regarding artifact sources, starting point, degree of automatization, test environment,*i.e.,* test bed, subsystem or the complete vehicle, purpose of testing, test case creation method, current and possible estimated test case coverage.

3.2 Study design and conduction

To estimate the potentials for MBTCC in the automotive industry, we created an online survey at BMW. The anonymous survey consists of 27 questions divided into three sections. The first section inquires the participants background information, *e.g.,* professional education. The second section inquires about the main test case artifacts and proceedings. The third section contains all further questions, inquiring, for instance, capabilities for automation, test case coverage, as well as quality assurance and estimations. Out of 27 questions, 14 are closed questions and 13 are open questions. To answer the closed questions, either multiple choices or Likert-rating scales were provided. The survey was executed from 23rd of August to 22nd of September 2017. An external department of BMW implemented and hosted the survey. The complete survey can be accessed online[2]. The raw survey data results are not published online, because of the company's confidentiality.

We contacted 196 professional test engineers and test managers of BMW. In the invitation mail, we asked personally to forward our questionnaire to directly accompanying service providers involved in the test case creation at BMW. For a statistically significant result, we calculated a necessary sample size of 65 out of 196 contacted test engineers. According to this sample size, we are 95% confident that the proportion of all contacted test engineers, with an error margin of 10% (approximately 20 participants), correspond similar to the ascertained survey findings [21]. We ultimately received 70 answers, with 69 valid questionnaires.

3.3 Results

One important aspect is a potential correlation between the personal background and the expected benefits of model-based testing. For this reason, the last question of the survey examines the profile with regard to the participants opinion on MBTCC (Fig. 4). A portion of 45% answered that they consider MBTCC as useful or very useful, 36% a neutral opinion towards MBTCC, and 19% consider it as a hindrance. For statistically data analysis we labeled a ordinal numerical scale with the Likert-scale, *e.g.,* from one with *MBTCC*

[2]www.se-rwth.de/materials/mbtcc

interferes with my work. to five with *MBTCC is very useful.* (Tbl. 1). On the basis of a mean ($\overline{x}$) of 3.29, a standard derivation (s) of 0.98, and a sample size (n) of 55 we conclude on a generally positive expectation from MBTCC. The sample size ($n = 55$) varies to the sample size of 69 valid questionnaires because of invalid or undeclared indications. On the impact of the test engineers' backgrounds (**R1**), we assumed that their opinion about the "benefit" of model-based test case creation is independent from their background. For statistical verification we use Pearson correlation, with correlation coefficient (r), p-value and the sample size (n) [21] (Tbl. 1).

Table 1: Correlation between background with expectation on model based testing.

Background	r	p	n
What is your main function?	-0.05	0.371	53
What did you study?	0.03	0.421	55
Experience in automotive industry	0.19	0.088	55
Experience in test case creation	0.10	0.240	55

The correlation coefficients of the questions "What did you study?" and "What is your main function?" are lower than the r of the questions "For how many years you have been working in the automotive industry?" and "For how many years you have been working in the sector of test case creation?". In contrast the p-values are higher for the functional and studying background questions. The correlation coefficients lower than $|\pm 0.20|$ support our hypothesis about only a small connection between the participant's opinion (see Fig. 4) and the background (Tbl. 1). Nevertheless, the p-values indicate statistically insignificant results.

Besides the expected model-based development for test engineers, we compare the test case creation based on textual requirements to MBTCC (Fig. 3). Therefore, we ask for the starting point and the result of the participants' step in the test case creation process (Tbl. 2).

Table 2: Input for test case creation and its results.

Initial situation	%	Produced result	%
Requirements	76.5	Concept of a TC	51.7
Concept of a TC	14.1	Draft of a TC	11.7
Draft of a TC	9.4	Automated TC	36.6

Three out of four test engineers start test case creation with requirements, whereas the other test engineers start with concepts or drafts of test cases (Tbl. 2). We directly linked the participants' indications on the *initial situation* with the *produced result.* The outcome reveals that one third of the processes start with requirements and end in automatically executable test cases.

On the basic environment for testing, the test engineers indicated that their test cases focus is generally the complete system with about 42.0%. The component tests are the second most indicated focus with 24.6% followed by the focus on subsystem (17.4%) and software (16%). For automatically performed test cases, the system environment association *Vehicle in the Loop* with 11.7% is less important. In contrast, the Hardware in the Loop (HiL) test cases are 81.6%. Software in the Loop (SiL) with about 5% and Model in the Loop (MiL) with 1.7% are less targeted with automatically performed tests. We also considered the sources for test case creation (Fig. 5). For 78% the most important source for test cases are natural language requirements, followed by - in this order - volatile personal experience, existing test cases, error descriptions and UML or SysML models.

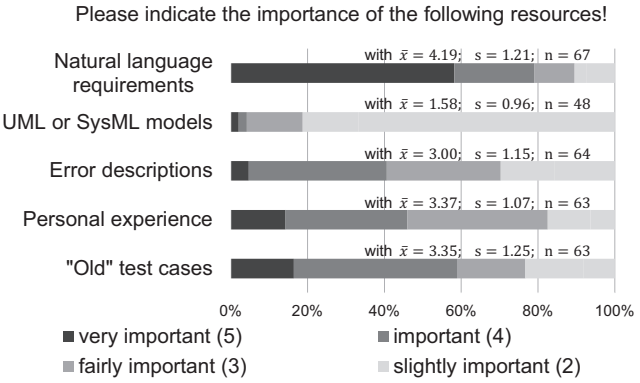

Figure 5: Sources used for test case creation.

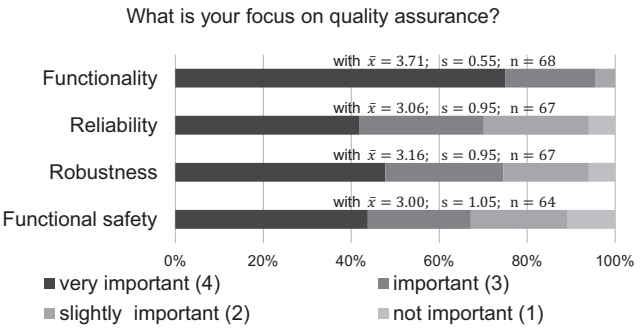

Figure 6: Focus in quality assurance.[3]

To investigate the quality focus **R2**, the questionnaire requests the focused quality assurance (Fig. 6). The data reveals that three out of four test engineers identify functionality as most important. The indicated "importance" measured by the mean of robustness is followed by reliability and functional safety, efficiency, reusability, integrability, and maintainability.

We also inquired to estimate the current and the potentially possible coverage (Fig. 7), to further identify potentials of coverage improvements. The current estimation mean ($\overline{x}$) is 64.7% with a standard derivation (s) of 25.3%, which is about 20% lower than the mean of the possible estimated coverage being 81.6% with standard derivation of 14.2%.

[3]The figure's visibility is compressed. The quality assurances, efficiency ($\overline{x}$=2.70;s=0.91;n=60), effectiveness ($\overline{x}$=2.16;s=0.83;n=63), integrability ($\overline{x}$=2.62;s=0.87;n=63), maintainability ($\overline{x}$=2.14;s=0.88;n=65) and reusability ($\overline{x}$=2.67;s=1.02;n=63) are not considered in the bar graph.

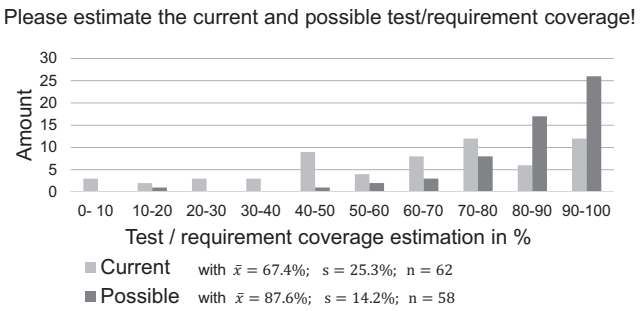

Figure 7: Estimated current and possible test coverage.

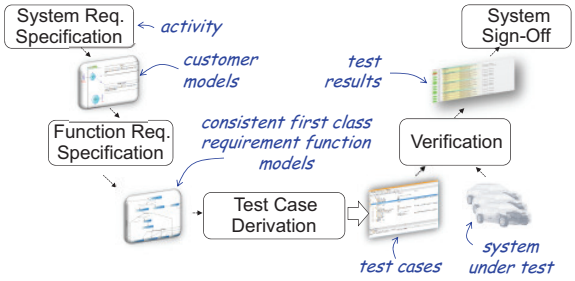

Figure 8: Overview of test case derivation.

3.4 Conclusions

Regarding the perceived benefits of MBTCC for engineers of different backgrounds (**R1**), we conclude a positive attitude towards MBTCC (*cf.* Tbl. 1). This positive attitude is largely independent of experience level, professional background, and focus of testing. Furthermore, the evaluation reveals that more than one third of the participants expect to benefit from MBTCC, despite a widely-adopted model-based approach is not in place (*cf.* Fig. 5).

A considerable portion of test engineers use old test cases, personal experience, and error descriptions as basis for test case creation (*cf.* Fig. 5). Creating new test cases from old ones suggests that new or changed requirements usually entail only small adoptions for new test cases especially for test cases in upcoming projects.

With MBTCC, requirements or system adjustments could directly create new exact model-based test cases instead of adjusting old test cases. Furthermore, MBT enables to check if the test cases are relevant for testing regarding to the model changes. The functional behavior in case of error is integrated in the early functional requirements, and can be included in test cases with MBTCC. The clear specification in SMArDT supports the test engineers and facilitates the automated test case creation. Moreover, the test engineer's experience is a fleeting knowledge. With SMArDT the implicit knowledge of experts becomes explicit knowledge in the diagrams. This hardens the test case creation and the whole process against personal changes and supports hypothesis (**R2**).

The main focus of testing in quality assurance is on functionality, functional safety, robustness and reliability (*cf.* Fig. 6). All these quality criteria are addressed by the second abstraction layer of SMArDT (*cf.* Sect. 2). The revealed potential for test case coverage improvement further strengthens hypothesis **R2** that MBTCC can enhance the test case quality. Referring to the most promising environments for model-based testing in the BMW context (**R3**), the participating test case engineers found system tests, subsystem tests, and component tests most useful. The low portion of SiL indication in the survey is probably related to the group of participants. SiL and MiL tests are performed by software developers and not by test engineers. The evaluation also reveals that automatically executable tests are mainly performed on HiL (*cf.* Sect. 3.3). Moreover, the focus of testing often is the complete vehicle or its subsystems. Nevertheless, the results indicate a gap in the context of MiL and SiL.

4 A METHODOLOGY FOR SYSTEMATIC MBTCC IN AUTOMOTIVE SOFTWARE ENGINEERING

The models created by the requirements engineers were rarely used by test engineers so far. Instead, the textual requirements were used as input for verification. However, as indicated by the survey results, a high percentage of the survey participants assume to benefit from MBTCC. Moreover, with SMArDT now used in an increasing number of departments at BMW, we expect the amount of available high quality SysML diagrams to increase. To fully exploit the potential of these diagrams, we developed a methodology that enables systematically deriving test cases from these diagrams. This section presents the methodology for automotive MBTCC. The approach utilizes ADs representing the function models of the second layer of SMArDT to automatically derive test cases for verifying functional properties. The derived test cases can be executed automatically.

4.1 Systematically deriving test cases

With our extension to SMArDT, the systems engineers first develop function models (ADs) from requirements provided in natural language, use case diagrams, and composite structure diagrams. The ADs then are used as input for automated, systematic test case derivation. They still reference logical information flow entities. Output of MBTCC is a set of test cases fulfilling the path coverage criteria C2c [16] as much as possible with each loop iterated exactly once. Paths of the AD are identified invalid if no input values exist leading to the specific paths. Test cases are created for valid paths through the AD only. Corresponding input values for each test case are derived by evaluating all guards and assignments on the corresponding path. The relevant part for MBTCC of the adapted systems engineering process is illustrated in Fig. 8.

Fig. 9 shows an excerpt of an AD that can be used to derive test cases. For simplicity we allow to omit object flow within diagrams, which is possible as only unique object names are allowed, *i.e.,* the actual object flow can be derived.

The possible input and output information of each function is determined by the AD's interface. For a given path, guard conditions restrict the possible values of input information while assignments determine the output for the function. To derive the corresponding test for the left path, the input value for `headlightsSwitch` must be set to `on`, otherwise the path would not be taken. Then the expected values for `lightsIndicator` and `headlights` can be derived by inspecting the assignments within the actions.

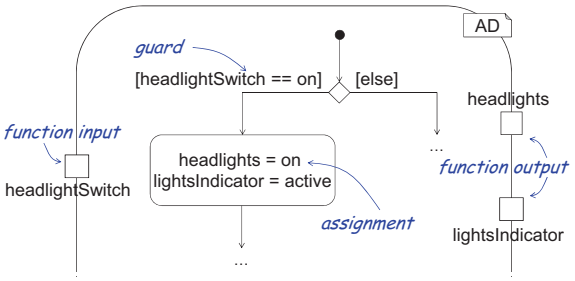

Figure 9: Excerpt of an AD describing requirements on the activation of headlights controls.

In general, we limit the amount of iterations per loop to one to minimize the number of resulting test cases. Applying this MBTCC method in practice has shown that multiple iterations do not contribute important content to test cases at function level. For complex scenarios, *i.e.,* where behavior for multiple iterations of a loop should be tested, manual test case creation still is necessary.

4.2 Structuring test cases

We structure test cases into three blocks: *preconditions*, *actions*, and *postconditions*. The preconditions define the initial situation prior to the test execution. If a precondition does not hold, *i.e.,* the initial situation cannot be precipitated, the corresponding test case cannot be executed. Action blocks comprise the actual execution of a test as derived from the platform-specific AD. Postconditions define the final situation after the execution. Each of the three blocks contains a list of atomic test steps. A test step comprises (1) the type of access action: setting or reading an information; (2) the name of the logical information; and (3) the value to which an information is set or which is expected when reading the information.

For the exemplary test of headlights requirements illustrated in Fig. 9 that should check the headlight controls of a car, Tbl. 3 illustrates the action block of a derived test case. The test case describes that, when using the switch to turn on the headlights, the actual headlights should turn on and the headlight indicator in the cockpit should be active. The test case derived for this scenario consists of three test steps. Each step is either setting (`Set`) the value or checking (`Check`) the value of a logical information. For the input information, a `Set` step and for each assignment, a `Check` step is calculated. In general, this might include `Check` steps for internal variables. The information names used in the test case refer to the switch for the headlights (`headlightsSwitch`) and the indicator light (`lightsIndicator`) in the cockpit as well as the actual headlights (`headlights`). Finally, the test case illustrates how the information values are used in the test case, *i.e.,* once the headlights are switched on, it is checked if the actual headlights are turned on and the indicator is active. This corresponds to the left path through the diagram of Fig. 9.

Table 3: Exemplary action block of a derived test case.

Action	Parameter	Expectation
Set:headlightsSwitch	on	
Check:headlights		on
Check:lightsIndicator		active

Since test cases correspond to valid paths through the AD, the order of test steps is defined by the occurrences of the information names on the respective path. Timing information (*e.g.,* between test step 1 and 2) are added later in the process as they depend on the implementation. Thus, the generated test cases remain implementation-independent.

4.3 Executing derived test cases

The ADs reference logical information names and values instead of software and hardware signals (*e.g.,* bus signals) and corresponding values. To execute the derived test cases automatically, this requires applying a mapping from logical information to concrete software and hardware signals prior to their execution (*cf.* Fig. 8). The mapping describes how to perform a test step for each pair of action and information and adds timing information where needed. A single test step at the logical level might result in several concrete test steps at the actual test bed. To ensure the derived test case is automatically executable, all test steps must be automatically executable on their own. Having such a mapping in place, the generated test cases can be executed automatically without any need for customization. Thus, the functionality modeled within the AD can be verified. The required mapping depends on product and test bed and currently must be developed manually. This potential for automating mapping creation is under investigation.

4.4 Configuring test case derivation

To ensure that all relevant use cases are covered, our methodology also provides a possibility to configure the way test cases are derived. This is due to the applied C2c coverage criterion, which results in exactly one test case per valid path through an AD. Hence, relevant use cases might not be covered by the derived test cases because there can be multiple input combinations that eventually result in the tested path. However, only one of the combinations is covered by a test case. Hence, to ensure that all relevant requirements as modeled by an AD are covered, it is possible to prioritize the atomic condition within guard decisions. With this, we can guarantee that certain atomic conditions of a decision node are covered by test cases. Using this extension to configure the process, all relevant use cases can be covered by the derived test cases. Moreover, test cases that do not relate to any use case, can be omitted.

5 A MONTICORE-BASED IMPLEMENTATION

We realized the generator creating test cases from ADs using the MontiCore [9] workbench for the efficient development of compositional modeling languages based on context-free grammars (CFGs). Using a modeling language's CFG, MontiCore generates the infrastructure to process models, check their well-formedness, and translate these into other artifacts. It has been applied to create

languages for various domains, including automotive [23], cloud systems [18], and robotics [2].

We export the diagrams modeled using the SysML modeling tool used at BMW to XML and transform these into UML/P [22] ADs. Based on the ADs, test cases are created and afterwards exported to a file format that is used as input by the tool responsible for executing the tests. The corresponding tool chain comprises three components: (1) a converter transforms the ADs exported from the SysML modeling tool to UML/P ADs; (2) a test case creator transforms the ADs into an intermediate test case representation. To this end, the configuration (*cf.* Sect. 4.4) is processed influencing the resulting test cases; and (3) an exporter transforms the resulting test cases to input formats specific to the tool used with product and test bed.

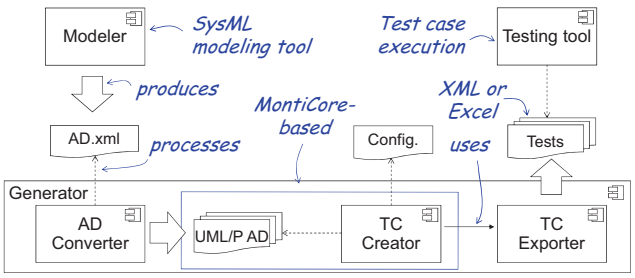

Figure 10: Generating test cases from ADs modeled with the SysML modeling tool used at BMW.

The overall tool chain is illustrated in Fig. 10. The component `ADConverter` processes the XML ADs and creates UML/P ADs. The `TCCreator` component processes the UML/P ADs together with a configuration model for the test case generation, as described in Sect. 4. The `TCExporter` component is used to export the derived test cases. Finally, the exported test cases are used as input for the actual testing tool. Its modular infrastructure enables employing MBTCC with changing input and output tools, *i.e.,* if the requirements ADs are not specified by the current modeling tool anymore, exchanging the `ADConverter` suffices to adjust it accordingly. A similar argument holds for changing the test tool we produce the outputs for.

The generation of actual test cases from the exported XML document as performed by the `Generator` component consists of four steps:

1.) The set of input and output information is determined by investigating the input AD's interface. In addition, intermediate variables are taken into account by inspecting the guards and assignments of the AD.

2.) All paths through the AD are calculated. To this end, the ADs are normalized, *i.e.,* hierarchies and parallel control flows of an AD are resolved using model transformations. Resolving parallel control flows is done by transforming parallelism into a sequential control flow in a defined order. This transformation can be performed under the assumption that the parallel flows are independent of each other. For each path, expected output values as well as values of intermediate results can be determined by taking into account the assignments of the given path.

3.) Each calculated path is transformed into a logical formula. The expression is created corresponding to the guards and assignments that are part of the respective path. Under the assumption that the values of information flows do not change during activity execution, the conjunction of all guards and assignments must hold on each valid path. For the left path of the exemplary activity diagram shown in Fig. 9 the calculated formula is:

$$headlightSwich = on \wedge headlights = on \wedge lightsIndicator = active$$

For paths where all information within guards and assignments are distinct, input values can be easily determined without calculation of a logical formula. However, our approach is also capable to find input values or declare a path as invalid in more complex situations.

4.) Finally, the logic expression for each path is solved using Z3 Theorem Prover [4], which can be used to check the satisfiability of logical formulas. For valid paths Z3 creates a witness for a solution of the formula. This witness can be transformed back to value assignments for input information fulfilling the given path of the diagram. If no witness exists, the path is invalid (due to contradictory guard conditions) and is discarded.

In the end, the generation process as shown in Fig. 10 comprises the final step of exporting the actual test cases. For each validated path the test actions are exported into the format supported by the tool responsible for the test execution. The implemented generator also allows for an easy extension of the many coverage criteria. As for now only the structured path coverage is available. Additionally, this criterion can be adapted by the configuration as presented earlier. The implementation of other coverage criteria is already planned for the future.

6 DISCUSSION

The survey results are subject to various threats typical to empirical research. This section, reports on these threats and highlights lessons learned from deploying our method at BMW.

6.1 Threats to study validity

We cannot eliminate duplicated submissions and wrong interpretations. Threats to the internal validity arise from the distribution of the survey by email and forwarding. We carefully selected our participants by their specialization in the context of test case creation. With the forwarding, we intend to additionally reach external test engineers involved in the process of test case creation. We can neither ensure to have all external companies included nor to have interrogated all external test engineers, because both weren't invited directly or might not have all permissions needed to participate in the survey.

For instance, the portions of the produced result (*cf.* Tbl. 2) could be distributed differently with all the external test engineers included. In particular, the automation of test cases is assigned to external service providers.

Moreover, test engineers are often of external companies. Especially the estimation of the current and the possible test case/requirement coverage (*cf.* Fig. 7) could be influenced if more internal test engineers existed. Although, we added imaginary circumstances "all resources and time you need" for the possible coverage, it could be perceived as a performance review and lead to adjusted estimations.

Threats to construct validity arise from employing an online survey. The style of question and answer options might have influenced the actual answers. For instance, the question about the focus on quality assurance has no alternative for a neutral statement, except "don't know" (*cf.* Fig. 6).

Besides the survey execution, there are threats to statistical validity. Based on the total sample of 70 participants in the mentioned survey (see Sect. 3.3), we are 95% confident that the margin of error for our sampling procedure and its results is no more than ± 20 participants. Furthermore, we mentioned that our results are statistically insignificant, *i.e.*, the p-values of Tbl. 1 indicate such results. Because of the margin error, the possibility of not every participant responding to every question and results being statistical insignificant, we can only assume trends.

In addition to the feedback from the survey, we learned from experience in the current project.

6.2 Lessons learned from applying MBTCC at BMW

Successfully introducing new methodologies requires identifying and convincing all involved stakeholders. Therefore, different interests and necessary organizational changes need to be more than accepted: They have to be understood by all affected persons.

Despite the benefits of MBTCC being understood by all partners, we underestimated the effort required to convince all involved stakeholders. The term *automated test case creation* was the main issue inflicted by a wrong communication. This term could be interpreted as a full automation and thereby classifying the testers' work as replaceable by a more or less simple algorithm.

In consequence, instead of achieving a cooperation between specifier and tester, which would decrease the workload of both and improve the overall model quality, the new approach was misinterpreted as an additional workload for the specifier while the created test cases could not replace manually created tests - as the intellectual input of the tester cannot be replaced.

The first perception of the test engineers was that the test cases are created without their knowledge. On the contrary, the defined approach suggest to include the test engineers' expertise into the model. Once this general misunderstanding was clarified, it was possible to convince the team members of MBTCC based on SMArDT. During a pilot phase, test engineers adapt six functions related to e-drive and created over 272 test cases with an additional effort of only 55 hours as shown in Fig. 11.

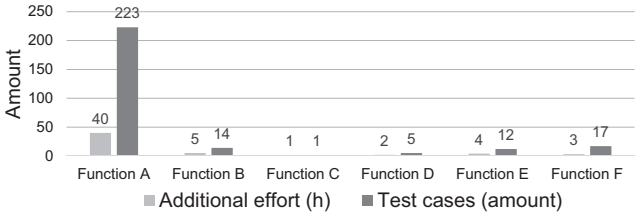

Figure 11: First results on MBTCC for six e-drive functions.

Despite that learning MBTCC requires additional effort, all involved parties agreed that these efforts will be reduced significantly with further tasks and ultimately will pay off for all parties. Moreover, additional effort was necessary to adapt already existing models. Conducting MBTCC from scratch would reduce the additional effort. The amount of created test cases relates to the amount of decision logic defined in the model. Function *A* already includes diagnosis related functionality, which results in a more complex decision logic. In consequence, this function has a significantly higher potential for automated test case creation. The measured effort is lower in comparison to the current manual approach, especially considering further iterations of each function.

Altogether, based on the current experience, the feedback is very positive. Based on first statements, the cooperation between tester and specifier seems to improve the model quality and the overall effort for test case creation could be reduced. Further feedback has been collected to identify additional potentials of improvement on test case creation regarding coverage criteria and use case or requirements related test case creation.

7 RELATED WORK

The presented work relates to surveys on model-based testing in automotive, as well as to other MBTCC approaches.

While there are, for instance, surveys on employing formal methods to automate design and specification in industry (*cf.* [28]), reports from automotive software engineering are rare. Where these surveys have been reported, such as in [3], they focus on tool use during requirements engineering and testing instead of the potential benefits of MBTCC. With this focus, the findings presented in [3] did not address our research questions.

Model-based test case creation is subject to active research [5]. Several approaches to (partially or fully) automated derivation of executable test cases from various input modeling languages have been brought forth. The most prominent approaches focus on UML or SysML. These approaches generally can be distinguished based on their support of concurrency in the input models. This includes various approaches to MBTCC that employ UML Statecharts or sequence diagrams as input models [12, 24, 26]. Approaches employing ADs as input prohibit concurrency in the models (*cf.* [10, 14, 15]). Moreover, various tools to support and facilitate model-based testing have been presented as well. These include, for instance, MBTsuite [1] and UMLTest [19]. MBTsuite translates ADs into test cases and can export these into a variety of different outputs, but does not support concurrency in its input models. UMLTest processes Statecharts, yet it's subject to the same restrictions. UMLTGF [11] processes ADs and, similar to our approach, supports ADs featuring concurrency, but cannot be extended to produce the output format required at BMW.

With the presented extension to SMArDT addressing the transformation from originally natural-language requirements to executable function tests, it relates to approaches translating natural-language requirements to test cases directly. Such transformations are, for instance, supported by the aToucan4Test framework presented in [29]. With aToucan4Test, input is restricted to a subset of structured English that does not support the precision and complexity achieved through manual function test modeling by domain experts. Despite applying MBTCC to various domains, such as web services [13], automated test case creation in automotive software

engineering with its established development and testing processes is rarely documented. Where it is, it focuses on specific tooling, or proposes general, abstract overviews only [25, 30].

8 CONCLUSION

Shortened development processes and frequently modified requirements affect the balance between time to market and high quality standards. To maintain high quality standards comprehensive testing is crucial to automotive software engineering, yet test case creation is still largely manual. We have conducted a survey to investigate if test engineers in the automotive sector could reap the benefits claimed by model-based testing. The study was sent to 196 participants at BMW of whom 69 valid questionnaires were returned. The study has shown that the participants expect to benefit from automated test case creation independent of their backgrounds, that they expect the test quality to increase, and that MBTCC is considered most useful for system, subsystem, and component tests. Based on this survey and the experiences from working with the automotive industry, we conceived a method to extend an existing specification method for requirements and testing.

We created an adjustable tool chain that can transform functional requirements modeled as ADs from various input formats into executable test cases for various output formats. Overall, after underestimating the persuasion efforts required to implement our methodology due to ambiguous terminology on our side, the feedback is very positive. Our methodology seems to improve the model quality and the overall effort for test case creation as hinted by the survey. In addition, the method improves the cooperation between specifier and tester and reduces redundancies as function and test model can be combined.

REFERENCES

[1] 2017. MBTsuite - the Testing Framework. (2017). http://www.mbtsuite.com/ Accessed: 2017-06-11.

[2] Kai Adam, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. 2017. Engineering Robotics Software Architectures with Exchangeable Model Transformations. In *International Conference on Robotic Computing (IRC'17)*. IEEE, 172–179.

[3] Harald Altinger, Franz Wotawa, and Markus Schurius. 2014. Testing methods used in the automotive industry: Results from a survey. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing - JAMAICA 2014*, Christof Budnik, Gabriella Carrozza, David Faragó, Baris Güldali, Barath Kumar, Vittorio Manetti, Roberto Pietrantuono, and Stephan Weißleder (Eds.). ACM Press, New York, New York, USA, 1–6.

[4] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems* (2008), 337–340.

[5] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. 2007. A survey on model-based testing approaches. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies held in conjunction with the 22nd IEEEACM International Conference on Automated Software Engineering (ASE) 2007*, Eileen Kraemer and Jonathan I. Maletic (Eds.). ACM, New York, NY, 31–36.

[6] Christof Ebert and John Favaro. 2017. Automotive Software. *IEEE Software* 34, 3 (2017), 33–39.

[7] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07)* 2 (may 2007), 37–54.

[8] Object Management Group. 2006. SysML Specification Version 1.0 (2006-05-03). (aug 2006).

[9] Arne Haber, Markus Look, Pedram Mir Seyed Nazari, Antonio Navarro Perez, Bernhard Rumpe, Steven Völkel, and Andreas Wortmann. 2015. Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components. In *Model-Driven Engineering and Software Development Conference (MODELSWARD'15)*. SciTePress, 19–31.

[10] Abdelkamel Hettab, Elhillali Kerkouche, and Allaoua Chaoui. 2015. A Graph Transformation Approach for Automatic Test Cases Generation from UML Activity Diagrams. In *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*. ACM, 88–97.

[11] Yuan Jiesong, Wang Linzhang, Li Xuandong, and Zheng Guoliang. 2006. UMLTGF: A Tool for Generating Test Cases from UML Activity Diagrams Based on Grey-Box Method. *Journal of Computer Research and Development* 1 (2006), 008.

[12] Supaporn Kansomkeat and Wanchai Rivepiboon. 2003. Automated-generating Test Case Using UML Statechart Diagrams. In *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology (SAICSIT '03)*. South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, 296–300.

[13] ChangSup Keum, Sungwon Kang, In-Young Ko, Jongmoon Baik, and Young-Il Choi. 2006. Generating test cases for web services using extended finite state machine. In *IFIP International Conference on Testing of Communicating Systems*. Springer, 103–117.

[14] Monalisha Khandai, Arup Abhinna Acharya, and Durga Prasad Mohapatra. 2011. Test Case Generation for Concurrent System using UML Combinational Diagram. *International Journal of Computer Science and Information Technologies, IJCSIT 2* (2011).

[15] Jonathan Lasalle, Fabien Peureux, and Jérôme Guillet. 2011. Automatic test concretization to supply end-to-end MBT for automotive mechatronic systems. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*. ACM, 16–23.

[16] Peter Liggesmeyer. 2009. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Springer Science & Business Media.

[17] Detlev Mohr, N Müller, A Krieg, P Gao, HW Kaas, A Krieger, and R Hensley. 2013. The road to 2020 and beyond: What's driving the global automotive industry? (2013), 2014 pages.

[18] Antonio Navarro Pérez and Bernhard Rumpe. 2013. Modeling Cloud Architectures as Interactive Systems.. In *MDHPCL@ MoDELS*. 15–24.

[19] Jeff Offutt and Aynur Abdurazik. 1999. *Generating Tests from UML Specifications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 416–429.

[20] Biren Prasad. 1997. Analysis of pricing strategies for new product introduction. *Pricing Strategy and Practice* 5, 4 (1997), 132–141.

[21] Irene Rößler and Albrecht Ungerer. 2008. *Statistik für Wirtschaftswissenschaftler*. Physica-Verlag Heidelberg, Heidelberg.

[22] Bernhard Rumpe. 2016. *Modeling with UML: Language, Concepts, Methods*. Springer International. http://www.se-rwth.de/mbse/

[23] Bernhard Rumpe, Christoph Schulze, Michael von Wenckstern, Jan Oliver Ringert, and Peter Manhart. 2015. Behavioral Compatibility of Simulink Models for Product Line Maintenance and Evolution. In *Software Product Line Conference (SPLC'15)*. ACM, 141–150.

[24] Mahesh Shirole, Amit Suthar, and Rajeev Kumar. 2011. Generation of improved test cases from UML state diagram using genetic algorithm. In *Proceedings of the 4th India Software Engineering Conference*. ACM, 125–134.

[25] Mark Utting and Bruno Legeard. 2010. *Practical model-based testing: a tools approach*. Morgan Kaufmann.

[26] Thi-Dao Vu, Pham Ngoc Hung, and Viet-Ha Nguyen. 2015. A method for automated test data generation from sequence diagrams and object constraint language. In *Proceedings of the Sixth International Symposium on Information and Communication Technology*. ACM, 335–341.

[27] Jon Whittle, John Hutchinson, and Mark Rouncefield. 2014. The State of Practice in Model-Driven Engineering. *Software, IEEE* 31, 3 (2014), 79–85.

[28] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods. *Comput. Surveys* 41, 4 (2009), 1–36.

[29] Tao Yue, Shaukat Ali, and Man Zhang. 2015. RTCM: a natural language based, automated, and practical test case generation framework. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 397–408.

[30] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. 2011. *Model-based testing for embedded systems*. CRC press.