



What am I testing and where? Comparing testing procedures based on lightweight requirements annotations

Florian Pudlitz¹ · Florian Brokhausen¹ · Andreas Vogelsang¹

Published online: 06 May 2020
© The Author(s) 2020

Abstract

Context The testing of software-intensive systems is performed in different test stages each having a large number of test cases. These test cases are commonly derived from requirements. Each test stages exhibits specific demands and constraints with respect to their degree of detail and what can be tested. Therefore, specific test suites are defined for each test stage. In this paper, the focus is on the domain of embedded systems, where, among others, typical test stages are Software- and Hardware-in-the-loop.

Objective Monitoring and controlling which requirements are verified in which detail and in which test stage is a challenge for engineers. However, this information is necessary to assure a certain test coverage, to minimize redundant testing procedures, and to avoid inconsistencies between test stages. In addition, engineers are reluctant to state their requirements in terms of structured languages or models that would facilitate the relation of requirements to test executions.

Method With our approach, we close the gap between requirements specifications and test executions. Previously, we have proposed a lightweight markup language for requirements which provides a set of annotations that can be applied to natural language requirements. The annotations are mapped to events and signals in test executions. As a result, meaningful insights from a set of test executions can be directly related to artifacts in the requirements specification. In this paper, we use the markup language to compare different test stages with one another.

This article belongs to the Topical Collection: *Requirements Engineering for Software Quality (REFSQ)*

Communicated by: Eric Knauss, Michael Goedicke and Paul Grünbacher

✉ Florian Pudlitz
Florian.Pudlitz@tu-berlin.de

Florian Brokhausen
Florian.Brokhausen@tu-berlin.de

Andreas Vogelsang
Andreas.Vogelsang@tu-berlin.de

¹ Technische Universität Berlin, Berlin, Germany

Results We annotate 443 natural language requirements of a driver assistance system with the means of our lightweight markup language. The annotations are then linked to 1300 test executions from a simulation environment and 53 test executions from test drives with human drivers. Based on the annotations, we are able to analyze how similar the test stages are and how well test stages and test cases are aligned with the requirements. Further, we highlight the general applicability of our approach through this extensive experimental evaluation.

Conclusion With our approach, the results of several test levels are linked to the requirements and enable the evaluation of complex test executions. By this means, practitioners can easily evaluate how well a systems performs with regards to its specification and, additionally, can reason about the expressiveness of the applied test stage.

Keywords Markup language · Requirements modeling · Simulation · Test stage evaluation · Test stage comparison

1 Introduction

In many areas, software systems are becoming increasingly complex through the use of open systems, highly automated, or networked devices. The complexity leads to an increasing number of requirements, which are often expressed in natural language (Mich et al. 2004; Kassab et al. 2014). In several domains, development standards prescribe the alignment of requirements and test procedures to assure that software-intensive systems are tested effectively (cf. ISO26262 (ISO 2011) for automotive systems, EN 50128 (DIN 2012) for railway systems, or DO-178C (RTCA 2012) for avionic systems).

Software-intensive systems are tested on different levels to enable a stepwise testing process and to increase the confidence in the system's functionality. In recent years, the V-Model has established itself as a procedural model for software development. It describes the progressing phases of development together with their associated quality assurance and testing measures. The recommended test stages range from unit and component tests through the testing of subsystems to integration test on a system wide scale. In particular, techniques like software-in-the-loop (SiL), hardware-in-the-loop (HiL), and human-in-the-loop testing are used for validation and verification. Each test stage has its own benefits, drawbacks, and constraints. SiL tests, for example, are cheap, flexible, and easy to scale. On the other hand, they strongly rely on realistic environmental models and usually do not consider effects related to hardware component behavior. Human-in-the-loop tests, on the other hand, are closest to reality but expensive and hard to scale.

A software system that stimulates a system-under-test (SuT) with specific inputs, executes it, and maps the resulting outputs to models of the prospective environment of the system is called a simulator. Simulators have become increasingly complex in recent years to anticipate system behavior in complex environments. Using such complex simulators has several advantages over conventional software testing like unit tests. Conventional tests only pass or fail, but there is little information about the contextual situation. Further, simulators are more flexible towards covering diverse variations in context behavior. Simulations also provide a high degree of flexibility regarding the abstraction of the environment or context of the system which is why they can be applied in all phases of development of the SuT. Additionally, compared to static software tests like unit tests, simulations introduce the capability of time-dependent analyses. It is possible to explore the behavior of a

SuT in its context over hours or days within a much shorter amount of time due to the time compressing execution of a simulation.

In current practice and especially in large companies, aligning testing and requirements activities is a challenge (Bjarnason et al. 2014; Galko et al. 2014). Simulation scenarios are often not directly derived from requirements but handcrafted by specialized engineers based on their own expertise of the problem domain (Huang et al. 2016). Further, the results of simulation runs are often not fed back to the level of requirements, which means that a requirements engineer does not benefit from the insights gained by running the simulation. This misalignment has several reasons. First, requirements engineering and simulation is often conducted in different departments. Second, simulators are complex systems that need to be configured by simulation experts which makes it hard for requirements engineers to use them. Third, requirements and simulations are on different levels of abstraction which makes it hard to connect events generated by the simulation to the requirements - especially, when they are written in natural language. As a result, the simulation scenarios are often unrealistic and do not ensure that all requirements are covered and verified. Modeling can help closing this gap between requirements and simulation. However, if the necessary models are too formal, requirements engineers are reluctant to put in effort to model the requirements.

In a previous publication (Pudlitz et al. 2019b), we proposed a lightweight approach that allows engineers to annotate natural language requirements and link the respective part of a requirement to a signal or an event in a test execution. By analyzing logs of test executions for the linked events, we can feed back information regarding the execution to the level of the annotations and thereby to the level of requirements. The available annotations represent the markup language. A distinct feature of our markup language is that it contains annotations on different levels of detail. An engineer can decide how detailed he or she wants to annotate a requirement. The more detailed a requirement is annotated, the more information can be retrieved from a test execution.

In this article, we present how the markup language can be used to analyze test suites and compare different test stages with each other. More specifically, we explore the following three points of leverage:

- **Test Stage Similarity:** Given sets of test execution runs from different test stages, we want to know how similar the test executions on the test stages are.
- **Test Stage Compliance:** Given a test stage, we want to know how well the test stage complies with the implications of a specific set of requirements.
- **Test Case Alignment:** Given a set of test cases, we want to know how well these test cases are aligned with a specific set of requirements.

We evaluate these aspects by applying our approach to a real-world case from an automotive context. We have annotated 443 natural language requirements of a driver assistance system and linked the annotations to 1300 test executions from a simulation (SiL) environment and 53 test executions from test drives with human drivers (Vehicle-in-the-Loop). Further test levels like HiL will be included in future work. Based on the annotations and the mappings, we were able to compare the test executions on the two test stages with respect to the leverage points stated above.

Our analysis reveals that simulation scenarios show very different characteristics in terms of velocity and acceleration behavior compared with data from real drives resulting in different coverage of requirements. Moreover, our analysis reveals that the simulation is at a coarser level of detail compared to the data from real drives. Finally, we are able to show

that in the real test drives, 86 % of the requirements are at least partially or even completely fulfilled, and, in contrast, within the simulation, 65 % of the requirements are fulfilled.

Relation to Previous Publications This paper is based on our previous publication (Pudlitz et al. 2019b). This paper extends the previous publication by incorporating an extensive experimental evaluation. In this paper, the previously presented markup language is applied to a large-scale requirements specification from the automotive domain. The applicability of the approach and possibilities of analyzing different test stages with the annotation-augmented requirements is extensively evaluated. In particular, a comparison of driving scenarios, a comparison of the differences in expressiveness, and a comparison in terms of requirements verification potential is done on simulation and real driving data. Therefore, this paper highlights the substantial benefits of comprehensively adopting the lightweight multilevel markup language to connect software requirements and simulations.

2 Background and Related Work

This section provides an overview of tasks and research advances related to our approach of aligning requirements with explicit test executions at different stages of development. The first subsection presents an overview of approaches to requirements modeling as a comparison to our lightweight multi-level markup language. Section 2.2 discusses the subject of software testing and the differentiation to and potentials of extensive simulations. Section 2.3 provides an overview of existing efforts in aligning requirements specifications with testing environments. However, the presented approaches only provide means for static relations between requirements and corresponding tests. Therefore, Section 2.4 showcases existing tooling attempting to explicitly evaluate requirements with test execution in an end-to-end fashion. We distinctly show where our approach improves upon existing solutions.

2.1 Lightweight Requirements Modeling

Any kind of (semi)automated analysis profits from software requirements that exhibit a certain structure or adhere to a certain model type. However, expressing requirements as models usually comes with additional efforts for creating the models and the results may be less suitable for communication with non-technical stakeholders (especially with formal models). Therefore, different approaches have been suggested to make requirements modeling more lightweight. Some focus on supporting engineers in creating graphical models (Wüest et al. 2019), while others aim to maintain the appearance of natural language while still allowing for automated reasoning with so-called constrained natural language. The use of constrained natural language is an approach to create requirements models while keeping the appearance of natural language. Several authors propose different sets of sentence patterns that can be used to formulate requirements (Mavin et al. 2009; Eckhardt et al. 2016). Besides the advantage that requirements are uniformly formulated, the requirements patterns enrich parts of the requirement with information about the semantics. This information can be used to extract information from the requirements. Lucassen et al. (2017), for example, use the structure of user stories to automatically derive conceptual models of the domain. With our approach, we combine the strength of lightweight requirements annotations with the potential to be enriched with behavioral information collected in simulations.

2.2 Testing and Simulation

Software Testing is the verification that a software product provides the expected behavior, as specified in its requirements (Barmi et al. 2011). The conventional development and testing process for embedded systems is based on the V-model, which structures the development process into phases of decomposition of the system elements and their subsequent integration. Each requirement being specified on a certain level of abstraction is reflected by a test case on the same level, which determines whether the requirement has been implemented correctly. The increasing complexity of the systems, the many possible test cases, and the uncertainty about the system's context challenge this conventional testing process. Therefore, the use of simulations is becoming more and more popular to facilitate the testing of software in diverse, complex settings. This way, various behaviors of the software can be examined automatically.

Simulation is the imitation of the operation of a real-world process or system (Banks et al. 2000). The act of simulating something first requires that a model is developed; this model incorporates the key characteristics, behavior, and functions of the selected physical or abstract system or process. A simulator is a program that is able to run a simulation. Each simulation run is one execution of the simulation.

When simulation is used in a system development process, the model usually consists of a submodel that describes the system-under-development (SuD) and one or several submodels that describe the operational environment of the SuD. The simulation represents the operation of the SuD within its operational context over time.

A simulation scenario defines the initial characteristics and preliminaries of a simulation run and spans a certain amount of time. The scenario defines the global parameters of the operational context model. The model of the SuD is not affected by the definition of the simulation scenario. Therefore, a simulation scenario can be compared to a test case in a conventional testing process. The expectation is that the SuD performs according to its desired behavior in a set of representative simulation scenarios.

2.3 Requirements and Test Alignment

Alignment of requirements and test cases is a well-established field of research and several solutions exist. Barmi et al. (2011) found that most studies of the subject were on model-based testing including a variety of formal methods for describing requirements with models or languages. Another possibility to align requirements and test cases is by establishing trace links between the artifacts (Ramesh and Jarke 2001). Alignment practices that improve the link between requirements and tests are reported by Uusitalo et al. (2008) based on six interviews, mainly with test roles, from the same number of companies. The results include a number of practices that increase the communication and interaction between requirements and testing roles, namely early tester participation, traceability policies, considering feature requests from testers, and linking testing and requirements experts. Linking people or artifacts was seen as equally important by the interviewees.

2.3.1 Model-based Testing

In model-based testing, a behavioral model of the system, called test model, is developed based on informal requirements of the system (Utting et al. 2012). This test model is used to generate test cases automatically. One problem in this area is that the generated tests cannot

be executed directly against an implementation under test because they are on different levels of abstraction. Additionally, the formal representation of requirements often results in difficulties both in requiring special competence to produce (Nebut et al. 2006), but also for non-specialist (e.g. business people) in understanding the requirements. The generation of test cases directly from the requirements implicitly links the two without any need for manually creating (or maintaining) traces (Bjarnason et al. 2014). However, depending on the level of abstraction of the model and the generated test cases, the value of the traces might vary. For example, for use cases and system test cases, the tracing was reported as being more natural in comparison to using state machines (Hasling et al. 2008). Errors in the models are an additional issue to consider when applying model-based testing.

2.3.2 Traceability between Requirements and Tests

Alignment of requirements and test cases can also be achieved by setting up traceability links between artifacts (Gotel and Finkelstein 1994; Uusitalo et al. 2008). These trace links provide a means to enrich requirements artifacts with the information, that there exists a test artifact encompassing the functionality specified by the requirement. Trace links can be associated with different meanings to help engineers perform impact analysis or increase test coverage (Uusitalo et al. 2008). Traceability challenges include volatility of the traced artifacts, informal processes with lack of clear responsibilities for tracing, communication gaps, insufficient time and resources for maintaining traces in combination with the practice being seen as non-cost efficient, and a lack of training (Cleland-Huang et al. 2003). In addition, traceability links are inherently static artifacts that either link two artifacts or not. We are more interested in linking information of dynamic executions to the level of requirements. This facilitates the evaluation of requirements in a more detailed way by not only showing that there is a test case for a requirement, but actually present the evidence, that said requirement is successfully verified within a test.

2.3.3 Test Coverage

An important measure to assess the adequacy of a test suite is the notion of its *coverage* (Zhu et al. 1997). Most prominently, test cases and test suites are assessed based on their code coverage, i.e., the ratio of code that is executed when a test case or a test suite is executed. Especially for black-box testing, the concept of requirements coverage has been defined to assess the ratio of requirements that are “executed” by a test suite (Whalen et al. 2006). Similar to approaches for requirements and test alignment, requirements coverage can be computed based on (formal) requirement models (Whalen et al. 2006) or based on static traceability links between test cases and requirements artifacts (Ramesh and Jarke 2001). In our work, we focus on black-box testing and compute a measure of requirements coverage. However, we do not rely on formally defined requirements. Yet, we are interested in a more detailed assessment of requirements and test relationship than just a static relation indicated by a trace link.

2.4 Tooling for End-to-end Testing

A comparable approach to verify requirements within a testing and simulation environment in an end-to-end fashion is presented by the tool Stimulus by software company Argosim.¹

¹ www.argosim.com

Stimulus lets the user define formalized requirements and enrich the system under development with state machines and block diagrams to include behavioral and architectural information, respectively. With the help of a build-in test suite, signals from the environment on which the systems depends and reacts can be simulated. The system behavior within these simulations is evaluated with regards to its constraints specified by the requirements and violations are detected. The main features include the detection of contradicting and missing requirements.

This tooling approach however exhibits some major differences to the methodology proposed in this paper. First and foremost, the form in which requirements are drafted in Stimulus is in a highly formalized manner from which this approach is to be differentiated. While there are many efforts within the research community to explicitly formalize requirements to improve on their verification possibilities (Barmi et al. 2011), this markup language aims to provide the requirements engineer with a means to intuitively annotate natural language requirements in order to unfold the implicitly contained information in a way it can be used for verification purposes within a simulation. Secondly, the testing capability provided by Stimulus depends on the user to define inputs to the system and assign a range of values to them for test execution. This step however shall be automated with the proposed approach. From the data provided by the markups, a scenario for the simulation environment will be constructed, which evaluates the underlying constraints.

3 Approach

Our approach is schematically shown in Fig. 1. The starting point of our approach are natural language requirements, as seen in Fig. 1 as R1, R2 and R3. The requirements contain the software specifications and are written without patterns or constraints. Therefore, our approach needs to deal with spelling errors, grammatical inaccuracies, or synonyms. The approach of applying markups to natural language requirements is designed to abstract from these inconsistencies.

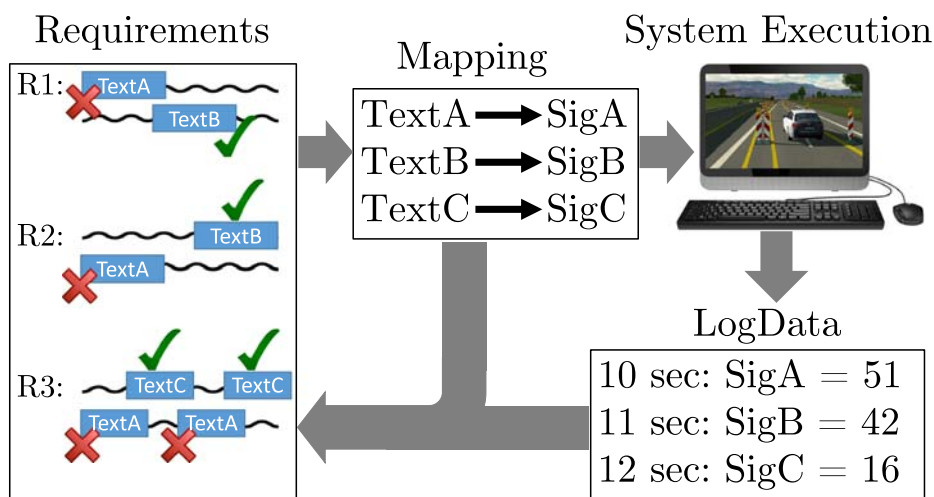


Fig. 1 Schematic representation of a requirements specification linked to a system execution with influencing intermediate steps

With elements of our markup language, the engineer can mark key phrases. The markings of the text passages are called annotations. These annotations are shown schematically as *TextA*, *TextB* and *TextC* in the requirements R1, R2 and R3. For example, system-relevant text phrases such as *motor*, *door*, *speed* or *acceleration* can be significant in requirements from the automotive context. Each annotation is extracted and then linked to specific signals of the functional implementation. This means that the natural language annotation *TextA* is assigned to the signal *SigA* (analogously *TextB* to *SigB* and *TextC* to *SigC*). Annotations contain important information regarding the system to observe the behavior of the system in the simulation or other test stages. Further, annotations can contain information about the environment (e.g., weather, roads, environment details) to benefit the creation of scenarios for simulations or other test stages.

By analyzing log data generated by some test stage, the annotations can be evaluated. Figure 1 shows a simplified, exemplary extract of the log data from seconds 10 to 12 with the occurring signal values. The results are then graphically displayed in the software specification. This allows the engineer to determine what information is important for the evaluation. The log data represents the behavior of the software at any time. Due to the increasing complexity of the vehicle functions, the complexity of the test stages evaluating these systems increases as well.

With our approach, the engineer can pursue two main goals. On the one hand, the fulfillment of requirements can be checked. On the other hand, test evaluations can provide valuable information regarding the behavior of the software - even at an early stage of development. Our proposed approach is independent of the chosen test stage and allows to use complex simulations as an evaluation tool for every stage of development.

Another advantage of our method is the presentation of the results in the natural language requirements. This allows easy access to the evaluations. The results can be understood directly in the context of the requirements without loss of content through translations. Especially with the growing importance of autonomous driving and more complex assistance systems, our lightweight markup language enables us to better understand complex scenarios and to make natural language requirements evaluable.

3.1 Markup Language

For the annotation of software functions and environment conditions, we developed a lightweight multilevel markup language to connect requirements specifications and log data from test runs. We developed our markup language to meet five demands.

1. A lightweight, intuitive approach for annotating objects in natural language software requirements.
2. A possibility to observe single objects as well as complex relations between elements in software tests without a formal translation.
3. An extraction of important simulation environment properties that must occur in the simulation.
4. A possibility to evaluate software behavior already during the development process.
5. A lightweight way to extract conditions to compare scenarios and test stages.

Based on these goals, we developed a language with four distinct levels. The idea to differentiate requirements regarding their context - system or environment - is adopted from Gunter et al. (2000). This differentiation enables the extraction of information about the test environment directly from the system specification. The work of Mavin et al. (2009) also introduces guidelines with the goal of formalizing as many requirements as possible.

Their work inspires our differentiation between triggers and pre-conditions and the association with actions. However, both works are created to serve as a template for creating requirements in a more formalized way. Our approach on the other hand focuses on enriching pre-existing natural language requirements. In addition, our language was refined by consulting and exploring the peculiarities of freely available natural language requirements (Dalpiaz 2018; Murukannaiah et al. 2017) and the requirements of an industrial partner.

The resulting language consists of elements, which are assigned to phrases in the natural language requirements documents with defined content characteristics.

The annotation in the present texts is carried out by the engineer and is deliberately a manual part of the approach. This ensures the flexibility and independence of the requirements. During the annotation process, the engineer chooses an element of the language depending on the context and the desired evaluation. The level is chosen implicitly and does not need to be selected explicitly.

3.1.1 Elements

Elements are the basic component of our markup language. Available elements and their description are shown in Table 1. It also shows, how the elements are strictly associated with different levels of detail. The correct understanding of the elements by the engineer is crucial, since the manually performed labeling effects the type of automated simulation evaluation. Each element is assigned to one of four levels, which define the level of detail of the evaluation.

3.1.2 Levels

All elements are assigned to one of the four levels. The levels build on one another and describe the level of detail of the elements. The properties as well as the limits of the levels are explained in the following.

The *Scope-Level (Level 1)* is used to differentiate between information on the system and on the simulation environment. As a result, the appearance of the objects in the simulation is displayed. However, no further information is available.

The *Type-Level (Level 2)* distinguishes the phrase of Level 1 into different types of text phrases depending on the behavior in the system. The different Level 2-types influence the type of evaluation and are the basis for the definition of conditions in Level 3.

The *Condition-Level (Level 3)* connects a type of Level 2 with a specific value via comparison operators to create condition statements. However, the formulated conditions have no connection among each other.

The *Causality-Level (Level 4)* establishes a relationship between the conditions of Level 3 and creates causal relationships. This requires detailed knowledge of the system and the necessary work process performed by the user is time consuming. The result however is an in-depth evaluation.

3.2 Annotating Requirements

There are three main goals to use our approach. First, extraction of information to select appropriate driving scenarios. Second, reviewing and monitoring software functions, and third, comparing different test runs and test stages. In all three cases, natural language requirements are the starting point of the investigation. Our markup language facilitates the

Table 1 Overview of all elements per Level

Level	Element	Description
Level 1: Scope	System	describes all information concerning the system, including any property perceptible from the outside as well as internal information. Result: link to signal available or not available.
	Environment	describes information on the simulation environment (e.g., weather) and simulation properties (e.g., simulation duration), and checks fulfillment of scenarios before a simulation run. Result: link to signal available or not available.
Level 2: Type	Value _{L1}	characterized by a value-continuous range and linked to system or environment. Result: progression over simulation time.
	State _{L1}	describes objects with multiple possible, but exclusive states (e.g., door - open/closed). Result: all appearing states.
	Event _{L1}	once or sporadically occurring object, often associated with signals. Result: number of appearances and average intermediate time.
	Time	concrete time specifications; automatically linked to simulation time. Result: not presented.
Level 3: Condition	Value _{L1} -Condition	values of Level 2 linked by <; ≤; =; >; ≥; ≠ with a number or parameter. Result: duration of the fulfilled condition.
	State _{L1} -Condition	states of Level 2 linked by = or ≠ with a possible state. Result: frequency, and duration in percent of the fulfilled condition.
	Event _{L1} -Condition	event from Level 2 with the values 1 or 0 for appearance and non-appearance. Result: number of appearances and average intermediate time.
	Time-Condition	time statements from Level 2 linked by <; ≤; =; >; ≥ or by natural language expressions such as <i>longer</i> , <i>shorter</i> , or <i>within</i> ; must be linked to other conditions as an extension of other Level 3 conditions. Result: not presented.
Level 4: Causality	{L3}-Trigger	Level 3 statements linked by AND, OR; if condition is fulfilled, {L3}-Action is triggered. Result: number of appearances.
	{L3}-Pre-Condition	Level 3 statements linked by AND, OR; pre-condition must be fulfilled in order to start a {L3}-Action. Result: number of appearances in total and as pre-condition with percentage.
	{L3}-Action	Level 3 statements linked together; following a {L3}-Trigger or {L3}-Pre-Condition. Result: number of appearance.

highlighting of necessary information and the observation in tests with an adaptable level of detail.

Figure 2 shows three examples of requirements for a Blind Spot Monitor [B-1, B-2, B-3]. The system warns the driver of vehicles in the blind spot by a red warning light in the outer mirror. If a vehicle is in the blind spot and the driver activates the turn signal lever, an additional warning tone is emitted. The red warning light flashes meanwhile.

B-1 contains Level 1 and Level 2 annotations. The annotated text phrases are each underlined in the original requirement. All detailed information about the annotation (level and type) are explicitly shown below. The *system* annotation of BSM checks if a corresponding signal is present in the test runs. This can vary depending on the simulation used or the level

Requirement Specification: Blind Spot Monitor	
ID	Object Text
B-1 L1/L2	The <u>BSM</u> is available for <u>velocities</u> between 0 km/h and 150 km/h.
	<div><div>L1BSM: System</div><div>L2velocities: Values</div></div>
B-2 L3	If a vehicle approaches the system vehicle from the rear and is within a <u>distance</u> of <u>3m</u> , the <u>BSM</u> reaches <u>warning level 1</u> .
	<div><div>L3Valuesdistance ≤ 3m</div><div>L3StatesBSM = warning level 1</div></div>
B-3 L3/L4	If the <u>BSM</u> is on <u>warning level 2</u> and the <u>turn signal</u> is <u>activated</u> on the same side another <u>vehicle</u> is <u>detected</u> , an <u>acoustic warning</u> signal is triggered.
	<div><div>L4Trigger: BSM = warning level 2 && turn signal = activated Action: acoustic warning = 1</div><div>L3Statesvehicle = detected</div></div>

Fig. 2 Examples of requirements of a Blind Spot Monitor with initial annotations on all Levels

of development of the system. To represent or compare the speed over the entire test period, the Level 2 annotation of *Value* is necessary.

B-2 contains only Level 3 annotations. The terms *distance* and *BSM* are each linked to a specific value from the requirement. For this purpose, an operator is selected, which is defined depending on the type. For states it is possible to use = and ≠, for values the common comparison operators.

B-3 contains one Level 3 and one level 4 annotation. The Level 3 annotation consists of the condition name *vehicle* and the associated condition value *detected*. They are linked with an equality operator. B-3, like B-1, shows that different levels can occur within a requirement. Level 4 annotations are made up of several Level 3 annotations. The Level 4 annotation in B-3 shows that two Level 3 annotations (*BSM = warning level 2* and *turn signal = activated*) are incorporated as triggers. The term *acoustic warning* is also a Level 3 annotation but in contrast to the other two an event. Therefore, the value 1 was added manually. Triggers, Pre-Conditions and Actions can always consist of a list of Level 3 annotations. These are linked by the engineer by *AND* or *OR* and later evaluated accordingly.

The example in Fig. 2 shows an excerpt from a software requirement specification. In B-2 and B-3, the requirements are each annotated at a higher level. This makes it clear that the engineer can start directly with the annotation on Level 3. It is not necessary to annotate all text phrases successively from Level 1 to Level 3. An exception is Level 4 (B-3), which requires Level 3 annotations to link them together. At the same time, this also means that several levels within a requirement, as in B-1, are possible. The engineer can receive evaluations flexibly and depending on the context. The flexibility of the engineer is further increased by the possibility to create annotations on Level 3 and 4 over several requirements.

In practice, the importance of requirements together with test runs in general, especially simulation runs, is a challenge for many companies. On the one hand, the requirement engineers lack the detailed understanding of simulations, on the other hand the test engineers are not familiar with the natural language requirements. Our presented markup language

together with our approach of connecting this language to test executions provide a tool for requirements engineers to make requirements more testable and to get feedback on the verification of the requirements. Test engineers have the opportunity to better coordinate test scenarios and develop them more specifically to meet requirements. This leads to a harmonization of both sides and promotes the exchange, understanding and cooperation of both sides.

The annotation process is intentionally left manual because the engineer shall decide on which level of detail he or she is interested in runtime information. However, we have also worked on NLP-based approaches to support the engineer by presenting words and phrases that are candidates for certain annotations (e.g., *States* (Pudlitz et al. 2019a)). In addition, our prototypical implementation distributes and applies all Level 2 and 3 annotations to all occurrences of the annotated phrase across the specification to avoid duplication and to reuse already created annotations.

3.3 Simulation Execution and Representation

Before a test run starts, the extracted annotations are linked with concrete signals and their associated values, like shown in Table 2. The signal names are internal signals of the system. The linking of natural language expressions with signal names of the system is called mapping. Depending on the level, an annotation can be interpreted as successfully mapped. In Level 1, the sole presence is sufficient for successful mapping. Level 2 requires a correct range of values in the available log data. For example, annotated *States* can not be mapped to continuous signals. Similarly, the same applies to Values, Events and Time Expressions. In order to achieve correct mapping for Level 3 expressions, it is necessary that signal names and signal values can be properly assigned. Since Level 4 annotations consist exclusively of Level 3 conditions, a Level 4 annotation is only successfully mapped if all contained Level 3 conditions are correctly assigned.

A mapping to specific signal names is not always possible. Table 2 contains an incomplete mapping for the Level 3 annotation `vehicle = detected`. This could be a first indicator that the test environment does not provide the signal or a poorly fitting scenario may have been chosen. Another possibility could be that the signal for the current state of development is not yet available. Nonetheless, it is still possible to start the test runs and just verify a subset of the system requirements. Further, the annotations from the text and the signals from the test are not necessarily a one-to-one but can also be established as

Table 2 Mapping from natural language expressions to signal names and values

NL expression		Signal
BSM	→	<bsm.Stat>
velocities	→	<veh.velocity>
distance ≤ 3m	→	<veh.distance> ≤ 300
acoustic warning	→	<bsm.acoustWarn.left> = 1 <bsm.acoustWarn.right> = 1
vehicle = detected	→	N/A
turn signal = activated	→	<veh.turnSig.left> = 1 <veh.turnSig.right> = 1
BSM = warning level 1	→	<bsm.Stat> = 1
BSM = warning level 2	→	<bsm.Stat> = 2

a one-to-many mapping. The expressions `acoustic warning` and `turn signal` in Table 2 demonstrate such a mapping. The signals are each mapped with two concrete system signals. The mapped signals are evaluated by default with an OR-operation. This allows a mapping even if general formulations are used in the natural language requirements.

After the extraction of the annotations and the preparation of the mapping, it is possible to carry out arbitrary tests. The presented approach is independent of selected simulators, test tools and test procedures. The evaluation is carried out exclusively on the log data of the respective test.

Log files may be structured differently depending on the test performed or the simulators available. However, at each time step, these data represent the state of the system. A big advantage in the analysis of the log data is the evaluation at each time step independent of the frequency. The included signal names differ essentially by two different types of recording. On the one hand, continuous signals, such as speed, acceleration or temperature, are recorded in each time step. These signals are annotated by the engineer in the specifications as *Value* and can therefore also be evaluated in the form of signal curves. On the other hand, signals with discrete states are only recorded when the state changes. The engineer can select *States* for these types of signals in the markup language.

For complex vehicle functions, increasingly complex vehicle tests are carried out. That means on the one hand the test runs are longer and with several vehicles at the same time. On the other hand, the scenarios are becoming increasingly complex and realistic. Both aspects contribute to the creation of a large amount of log data. The evaluation of individual signals is therefore not very meaningful. Our approach allows the evaluation of a significant number of vehicles over any period of time. By analyzing the log data, the annotations are analyzed at each time step. For better accessibility of the results, these are graphically displayed in the original requirements.

Figure 3 shows the three example requirements from Fig. 2 showing the results from the test run. In B-1, the text phrase BSM on Level 1 is mapped to the signal `bsm.state`. The

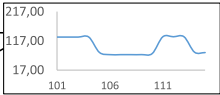
Requirement Specification: Blind Spot Monitor	
ID	Object Text
B-1 L1/L2	The <u>BSM</u> is available for <u>velocities</u> between 0 km/h and 150 km/h.
	<div><div>L1 BSM: System <bsm_state></div><div>L2 velocities: Values</div><div></div></div>
B-2 L3	If a vehicle approaches the system vehicle from the rear and is within a <u>distance</u> of <u>3m</u> , the <u>BSM</u> reaches <u>warning level 1</u> .
	<div><div>L3 Values distance ≤ 3m 5s 4.2 %</div><div>L3 States BSM = warning level 1 3x 15s 12.5%</div></div>
B-3 L3/L4	If the <u>BSM</u> is on <u>warning level 2</u> and the <u>turn signal</u> is <u>activated</u> on the same side another <u>vehicle</u> is <u>detected</u> , an <u>acoustic warning</u> signal is triggered.
	<div><div>L4 Trigger: BSM = warning level 2 && turn signal = activated Action: acoustic warning = 1</div><div>L4 Trigger: 3x Fulfilled: 100%</div><div>L3 States vehicle = detected <no mapping></div></div>

Fig. 3 Results of a test evaluation for exemplary annotations

speed signal can also be mapped. Since this annotation has already been created at Level 2 as *Value*, the continuous progress is displayed. Between minute 101 and 115 of the test run, a maximum speed of 131.1 km/h and a minimum speed of 69.9 km/h was reached. The concrete values of all evaluated *Value* texts are also available for further evaluations outside of the tool.

In B-2 all evaluations of the existing Level 3 annotations are shown. The $\text{Value}_{\{S\}}$ -Condition ($\text{distance} \leq 3\text{m}$) was fulfilled within the simulation for a period of five seconds. In addition, the ratio of this time in relation to the duration of the simulation is given in percent. In contrast, the number of conditions that have occurred can also be displayed for the $\text{State}_{\{S\}}$ -Condition. For this example, the BSM reached warning level 1 three times for a total time of 15 seconds, which corresponds to 12.5 % of the total test runtime.

In B-3 a Level 3 condition and a complex level 4 annotation is evaluated. As no suitable signals for the Level 3 annotation can be assigned in the mapping, the engineer will see the evaluation no mapping in the requirements specification. In contrast, the complex Level 4 annotation can be evaluated. The two Level 3 annotations annotated as triggers by the engineer were triggered three times in the log data. The event ($\text{acoustic warning} = 1$) also occurred in all cases, which means that the total annotation was 100 % fulfilled.

The example shows the influence of the different annotations on the type of evaluation. Simple evaluations are particularly suitable for early development of the software. With more time investment by the engineer, the annotations can become more complex and the evaluations gain in significance. In the most complex case, the Level 4 annotations, almost complete statements about the entire requirement can be made. The representation in the original requirements allows the engineer to evaluate all evaluations in the overall context.

The example shows the evaluations of a test run of an explicit test stage. The annotations of the engineer continue to serve as a starting point for comprehensive comparisons and evaluations. As our experiment shows, statements about the distribution of annotations and comparisons of several log data are also possible.

Our experiment shows a variety of evaluations. The detailed graphic representations in the natural language requirements can be domain dependent. In future work, the approach will be evaluated with test and requirements engineers in order to be able to make detailed statements about the benefits of the graphical presentation of results. In addition to the use and applicability of the language, the study will focus on making the simulation results easier to understand.

4 Experimental Evaluation

This section describes the experimental evaluation illustrating the potential of the presented markup language. We follow the reporting guideline for experiments in software engineering proposed by Jedlitschka and Pfahl (2005). We start by describing our research goals followed by the general experimental design and the context in which we conducted the study. Then, the data collection and analysis procedures are described, followed by the results of the experiment. Finally, these results are critically discussed.

4.1 Objective

This experimental evaluation aims at validating the applicability of the presented lightweight markup language on an exemplary, large-scale dataset from the automotive

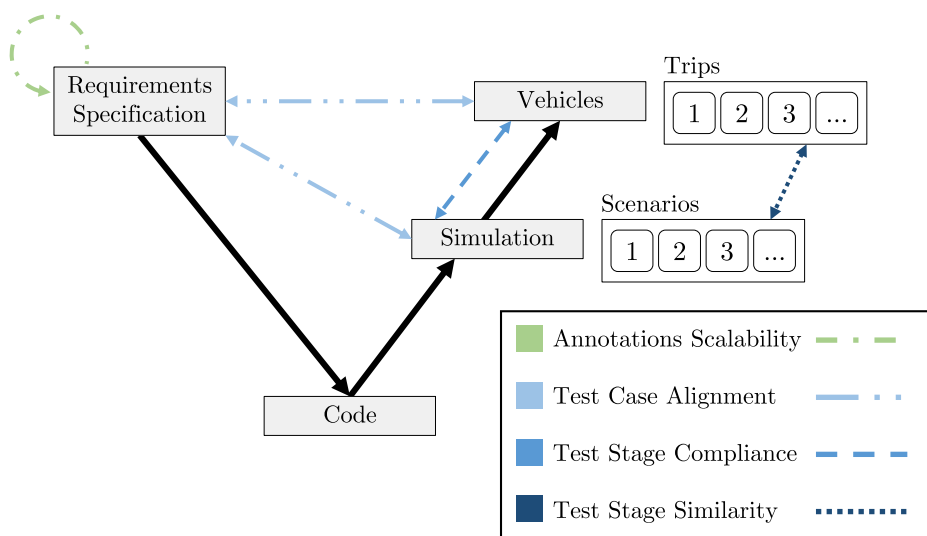


Fig. 4 Schematics of the experimental setup

domain. Further, additional benefits in establishing a comprehensive workflow with multiple test stages and extensive analysis are demonstrated.

The two different test stages under review in this paper signify that the approach as presented can be used to evaluate the system specification at different development stages. The aim is to show and discuss the significance of the results that can be obtained with this requirements verification approach.

It should be explicitly noted at this point that the presented analysis is executed for demonstrative purposes only, highlighting the possibilities enabled by the requirements annotation approach. This paper does not aim at optimizing the testing procedure itself or the specific function.

4.2 Experimental Design

The experimental evaluation is set up according to the schematics in Fig. 4. The starting point and base for the evaluation is the requirements specification which is extensively annotated with the markup language as presented above. Therefore, the first part of the evaluation concerns the investigation of the specifics of the application of the annotations to the extensive requirements specification.

The annotations in the specification are evaluated with respect to the test stages in two different ways: first, by analyzing log data from an extensive simulation, and second, by analyzing log data from actual, real test drives. We used VSimRTI² as simulator as it offers an extensive framework incorporating different simulators (Protzmann et al. 2017). The real driving data is provided by an automotive industry partner and is comprised of customer-oriented test drives executed by company employees.

The experiment setup defines four different research questions corresponding to the dashed arrows in Fig. 4.

²<https://www.dcaiti.tu-berlin.de/research/simulation/>

- **RQ1:** How are annotations distributed in realistic requirements specifications and how much effort is related to the annotation process?
- **RQ2:** Can the annotations be used to assess the similarity of different test stages with respect to the characteristics of the log data?
- **RQ3:** Can the annotations be used to assess the compliance of different test stages with the characteristics of a requirements specification?
- **RQ4:** Can the annotations be used to assess the alignment of actual test cases with the characteristics of a requirements specification?

The first research question elaborates on the general applicability of our approach by applying the markup language to a large-scale requirements document. The remaining research questions analyze the additional benefits resulting from the application of our markup language.

The difference between RQ3 and RQ4 is that in RQ3, we are interested in assessing the general ability of a test stage to trigger and observe certain phenomena described in the requirements specification. For example, if a requirement mentions *rain* as an important part of a requirement, a test stage must be able to stimulate and monitor the occurrence of rain. In RQ4, we are interested in the characteristics of the actually executed test cases (e.g., if a test case has been executed where it actually rained.) The detailed analysis process is explicitly explained in Section 4.5.

4.3 Context

This experimental evaluation is based on a requirement specification from the automotive domain provided to us by an industry partner. It is therefore a realistic application and evaluation as it would be carried out by industry experts. The requirements specification defines the behavior and setup of a blind spot monitoring (BSM) system used in modern passenger vehicles. The system exhibits a reasonable complexity to be able to draw generalizable conclusions while still maintaining an intuitively comprehensible overview.

The system visually warns the driver with a light in the side-view mirror when an object is within the blind spot of the driver. If this scenario occurs and the driver does not notice this warning and sets the indicator to signify the desire to change lanes, the light begins to flash. Additionally, an acoustic warning is triggered to keep the driver from changing lanes and potentially hitting the object in the blind spot.

Internally, the BSM has three very important states. As default, the system is in warning level 0. When an object enters the blind spot of the vehicle, but is situated behind the system vehicle, warning level 1 is triggered which enables the described visual warning. As soon as the approaching vehicle is next to the system vehicle but still in the blind spot, warning level 2 is triggered. Only if the system is set to warning level 2, the above mentioned acoustic warning is triggered.

An additional feature of the BSM system is the exit warning capability. This feature is only enabled when the vehicle is standing still (e.g., in a parking situation). This part of the system aims to prevent the occupants of the car hitting the car doors against objects next to the doors. If there is a small object next to one of the car doors that is detected by the sensors, the respective door is blocked to prevent damage to the car door or, potentially, the detected object.

The requirements specification that describes the BSM system consists of 443 artifacts written in natural language. Since the specification is written in unstructured, unrestricted

natural language, a multitude of different formatting styles like enumerations is present. Of the artifacts, 292 are actual requirements, 60 are additional notes - referred to as information - and 91 are headings.

4.4 Data Collection Procedures

This section provides an overview of the different data sources and how they were augmented to serve as a basis for the experimental evaluation.

4.4.1 Requirements and Annotation Data

To perform the planned analyses, the requirements are annotated manually. For the purpose of annotating large-scale requirements documents, we developed a tool to assist this process. The tool integrates the objects from the requirements specification and presents them in a structured way (similar to a requirements management tool). It is then possible to select text passages for annotations and subsequently select the level and type of annotation that should be applied to the text passage. The existing annotations of the currently selected requirement are highlighted in the text and additionally displayed in a designated section. The tool also supports the import of requirements from different sources and the export of annotated requirements in an XML format for further processing.

The first two authors of the paper annotated the requirements by splitting up the specification into two parts, annotating one part, and then validating the annotations of the other author in the second part. By this, both authors finally had a look at all requirements of the systems. Deviations and differences in the annotations were discussed and resolved. Both annotators aimed for annotations with the highest possible level in each requirement. Section 4.6.1 provides a detailed analysis of these annotations. The authors have been involved in requirements engineering from a research perspective for over two years and have dealt with the automated extraction of information from requirements specifications in the past. During their research, they work with requirements from the automotive context and are therefore able to incorporate their domain knowledge into the annotation process.

4.4.2 Annotation Mapping

For linking the annotated text passages to signals of a test stage, we manually created a *mapping* for each of the two test stages considered in our study. Table 2 shows an example of such a mapping. The mapping process of both test stages is carried out by two authors.

For the real driving data, consultations with domain experts ensure a high quality and correct mapping of annotations. When mapping the text phrases to the signals of the simulations, the signal names are coordinated in direct exchange with the developers of the BSM functionality within the simulator.

The basis for the mapping are all Level 3 annotations of the specification. Therefore, all annotations on Level 1 and 2 that are not implicitly contained within the Level 3 annotations are not considered in the mapping and consequently, in the subsequent evaluation of the test stages. Further, since the annotations on Level 4 are aggregated exclusively of annotations from Level 3, these are inherently included in the mapping process.

As described in Section 3.3, an annotation on Level 3 consists of a name and an assigned value. Therefore, not only the signal needs to be mapped but also the corresponding value needs to exist for this signal in order to assign it in the mapping.

4.4.3 Log Data Generation and Selection

As explained before, the log data is collected from two sources: a simulation environment and a database of test drive data from an automotive industry partner.

Simulation Data The simulator - VSimRTI - is a framework integrating different simulators. The traffic is simulated with SUMO (Lopez et al. 2018). This provides Simulated Urban Mobility (SUMO) within a network simulator. In VSimRTI, it is easy to write custom functions and map them to the vehicles in the scenario. We implemented a BSM system and equipped all passenger vehicles in the scenario with the implemented BSM module.

We implemented the BSM module according to the system specification as presented by the requirements. The implementation has been provided an affiliate expert of the research facility (i.e., none of the authors have been involved in the implementation of the BSM functionality).

To make the simulated scenario as realistic as possible, the LuST - Luxembourg SUMO Traffic - scenario is used (Codeca et al. 2015). This publicly available³ scenario aims at realistically representing the daily traffic in Luxembourg for a whole day. The layout of Luxembourg is translated exactly into the simulator, with city highways, main and residential roads. The traffic demands are simulated by analyzing the official demographics of Luxembourg in regards to the distribution of population and the age. In addition to this meticulously procured individual traffic, the exact bus routes and stops of the public transport in Luxembourg are incorporated in the scenario. Therefore, although the actual verification of the BSM is not in focus of this paper, the used simulation is highly realistic and a sophisticated means to test the system.

The simulation is started at 8 o'clock in the morning in simulation time, which is the peak of the number of simulated vehicles. In our simulation, this results in 1300 vehicles, which all drive on the virtual map for about 10 minutes. To collect the data from this 10-minute simulation, the simulator was executed for 5 hours on a standard laptop. We collected data of 13 signals from the 1300 vehicles with a rate of 50 Hz.

Test Drive Data The real driving data is provided by our industry partner and is comprised of test drives by employees with company-owned vehicles. These drives therefore represent the behavior of the vehicles in real life scenarios with actual users in customer-oriented driving trials. The data was packaged in 10-minutes snippets.

We extracted snippets where the relevant signals of the BSM are activated and show changes in the behavior. This way, 53 trips of 10 minutes each are extracted and serve as basis for the evaluation. We were not able to infer the actual number of involved drivers or distinct vehicles from the data. We extracted data of 30 signals from the 53 trips with a rate of 50 Hz.

4.5 Data Analysis Procedures

The setup presented in Section 4.2 defines a fourfold analysis. Following, the specifics of these analysis approaches are detailed.

Large-scale Application of Annotations The evaluation of the annotation characteristics is done by reporting various statistics regarding the distribution of the annotations with respect

³<https://github.com/lcodeca/LuSTScenario>

to their level and the requirements they originate from. It is examined how the annotations on each level spread over the different types of objects in the requirements specification. Further, the specific structures of the annotations on each level and how they manifest in the application on the large-scale requirements specification, are examined. In addition, we report the effort in terms of working hours spent to annotate the requirements.

Test Stage Similarity The evaluation of test stage similarity is carried out by relating the trips of the vehicles in the simulation to the trips of the real driving data based on system-dependent and independent factors. For our experiment, we used *vehicle speed* and *vehicle acceleration* as representatives for system-independent factors and *warning level* as representative for a system-dependent factor. The distributions of the values of these signals will be observed and analyzed to yield insights on how well the simulation represents the actual system behavior in real life scenarios. Therefore, this quantitative analysis is used to derive a qualitative statement about the similarity.

Test Stage Compliance The two test stages are further compared with respect to their compliance with the requirements specification and the augmented annotations. This is done by conducting a twofold comparison: once regarding the annotations and additionally regarding the requirements which these annotations originate from.

Annotation coverage: The base for this evaluation is the mapping of annotations to signals. The annotation coverage metric defines the ratio of annotations that exhibit a mapping and can therefore potentially be examined within the tests. This analysis is done for every level of the annotations.

Requirements coverage: The requirements coverage metric presents the ratio of requirements that can entirely, partially, or not at all be analyzed within the tests because contained annotations cannot be mapped.

To give an example of the defined metrics, we apply them to our example from Fig. 3. Of the total of six annotations, five are mapped to signals. Therefore, the annotation coverage in this small example is 83 %. Two of the requirements (B-1 & B-2) contain only annotations with a mapping while B-3 has both mapped and unmapped annotations. Thus, the requirements coverage is 67 % for entirely mapped requirements, 33 % for partially mapped requirements, and 0 % for entirely unmapped requirements.

Test Case Alignment The fourth and final part of the analysis is conducted by evaluating the test case alignment.

Depending on the level, annotations can be fulfilled. On Level 1, an annotation is fulfilled if the natural language expression is mapped to a signal. On level 2, the annotations are divided into types. In the mapping, the value ranges must match the types. Level 2 annotations are fulfilled if the corresponding information is available in the log data. For Levels 1 and 2, the exemplary annotation of the vehicles velocity is fulfilled, if it is actually recorded in the test stage and conforms to the specified type on Level 2. In this example, the velocity would probably be tagged as a value and therefore coincide if it is actually recorded with a continuous⁴ range of values.

In contrast, the annotations on Levels 3 and 4 are enriched with additional information that can be tested for fulfillment. On Level 3, a condition can be fulfilled or not. A Level 4

⁴Although all signals are recorded digitally and therefore are discretized to a given resolution of the values, a continuous sampling can be assumed if the range of the signal is not bound to some restricted set of states.

annotation is considered fulfilled if all contained Level 3 conditions are fulfilled. Again referring to the example of a vehicles velocity, an annotation on Level 3 would be the assignment of a certain value, or range of values to this signal. If, for example, a requirement mentions a velocity of 50 km/h or lower as a precondition for some event, this annotation is fulfilled on Level 3 if, at any point in the simulation or test drive, the velocity actually is lower than 50 km/h. On Level 4, the evaluation of fulfillment would as well be connected to the event which the specified velocity is a precondition for. So only, if every time the specified event occurs, the velocity is below the specified limit, will the Level 4 annotation be regarded as fulfilled.

As in the test stage compliance section, this evaluation is also focused on the metrics of annotation and requirements coverage. However, the two metrics are interpreted differently. In this case, we only consider mapped annotations and analyze whether and how they have been fulfilled in the analyzed test data.

Annotation coverage: The annotation coverage now indicates the ratio of annotations that are actually evaluated, i.e. fulfilled, within the tests.

Requirements coverage: For the requirements coverage, this is reflected onto the requirements in the specification to gain an insight of how well the tests actually audit the specifications.

Applying these new metric definitions to the example from Fig. 3, we see that all mapped annotations can also be evaluated. Therefore, annotation coverage and requirements coverage are 83 % and 67 %, respectively, as in the previous chapter. The potential of the evaluation is thus fully exploited because 100 % of the mapped annotations are fulfilled.

4.6 Results

This section provides the analysis of our experiment according to the strategies presented in the preceding chapters. The structure is set up to answer the research questions presented in Section 4.2.

4.6.1 Evaluation of the Large-scale Application of Annotations

The following section provides insights into the distribution of annotations to the requirements specification. For readability and clarity, in this section, the super-ordinate term for the three types of elements in the specification - namely requirement, information and heading - will be *artifacts*, as usual for the requirements engineering domain. In the rest of the paper, the term requirements includes all categories. The artifacts are annotated manually by the authors as described above.

Figure 5 shows the number and type of artifacts containing annotations for each level, as well as the number of unique annotations for each level. The latter is visualized by the pillars, which refer to the second axis.

In Level 1 there are 127 unique annotations, which split into 105 annotations tagged as part of the system and the remaining 22 as part of the environment. Since the Level 1 annotations are consistently used in the requirements, they spread over 294 artifacts with an average of 2.3 Level 1 annotations per annotated artifact. Breaking the artifacts down, 85.4 % of them are requirements, 12.2 % are informations and only about 2.4 % are headings. Therefore, about 86 % of all requirements contain an annotation on Level 1, 60 % of all informations and only about 7.7 % of all headings.

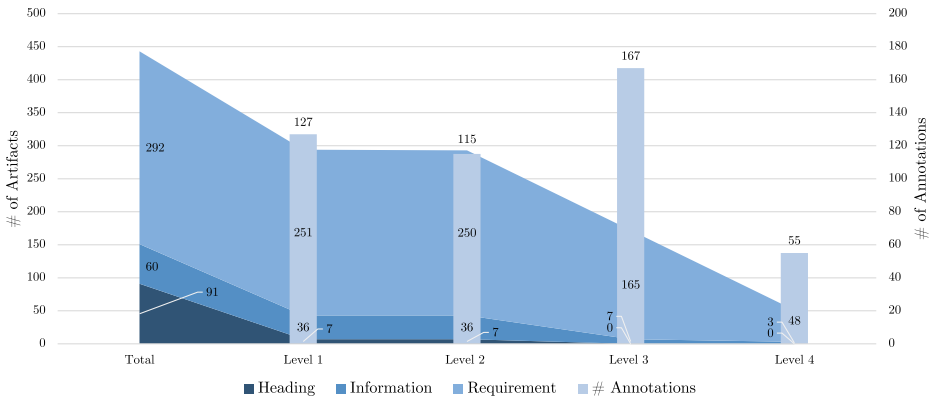


Fig. 5 Statistics of the annotations in the requirements

Elaborating on these annotations, 115 are refined on Level 2, resulting in 293 annotated artifacts. The majority of Level 2 annotations are states, as displayed in the distribution of Level 2 annotations to types in Fig. 6. In previous work, the extraction of these states from natural language requirements has been automated to drastically reduce manual labor (Pudlitz et al. 2019a). There is just one requirement containing only annotations on Level 1. Therefore, the number of Level 2 annotations per artifact is similar to the one reported for Level 1, amounting to 2.2.

The next step is the association of conditions with the Level 2 annotations to establish Level 3. In this experimental setup, 73 of the annotations on Level 2 are transferred to Level 3, accounting for 63.5 % of all Level 2 annotations. Due to the possibility of multiple associations per Level 2 annotation, this results in 167 annotations. Therefore, every annotation on Level 2 that is incorporated on Level 3, is assigned 2.3 conditions on average. Interestingly, although the overall number of annotations increases, these spread over significantly fewer artifacts with just 172 containing annotations on Level 3. About 4.1 % of these artifacts are informations, with the rest being actual requirements. This distribution is reasonable, since the requirement artifacts of the specification should - per definition - contain the critical technical specification for the system. On average, there are 2.1 Level 3 annotations per annotated artifact.

Of all Level 3 annotations, 61.1 % are used to establish causal relationships on Level 4, resulting in 55 unique annotations. Therefore, each Level 3 annotation occurs in roughly 1.5 annotations on Level 4. Interestingly, the annotation containing the abbreviation *BSM* (i.e. the systems name) is used far more often, occurring in 28 of the annotations on Level 4. On

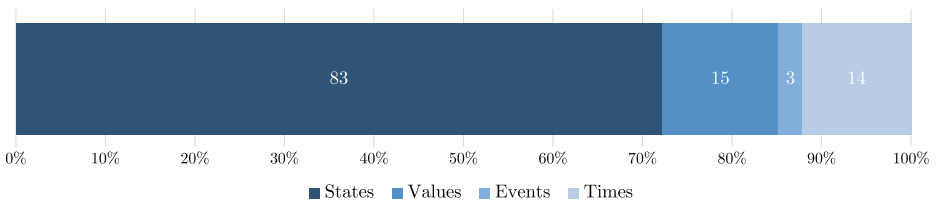


Fig. 6 Distribution of the Level 2 annotations to the different types

average, each Level 4 annotation contains roughly 1.6 Preconditions or Triggers and about 1.2 Actions. However, with 63.6 % of all Level 4 annotations, the majority of causal relations is established as Triggers. Further, 85.3 % of Level 4 annotations with multiple linked conditions or actions are connected with the logical operator *AND*, while the remaining minority is connected via an *OR*. Finally, only 51 artifacts contain these Level 4 annotations, with just 3 of them being informations and again, the majority being requirements.

The distribution of annotations to the different types of artifacts is rather intuitively reasonable. The very few occurring annotations within headings are due to some more generic annotations or specific system names, which of course occur in some of the headings. The ratio of informations and headings is similar in regards to the overall distribution and the distribution of annotations. Since the informations serve as additional content to further specify the actual requirements, they inherently contain some annotations. However, as observed in the analysis, especially in Levels 3 and 4, where the entropy of the annotations considerably increases, the vast majority of annotations originate from requirements artifacts.

The annotation process was carried out in several steps. Time was measured separately for the annotation process and the review process. The entire requirement specification was annotated by two authors in 5 hours and 47 minutes. The duration of the entire annotation review process was 83 minutes.

Interpretation of Results This analysis of the application of the markup language to a comprehensive requirements specification explores and answers the first research question. The presented results facilitate the assumption that the language scales well. It is demonstrated that it is possible to represent a substantial amount of the functional implication of the specification with the help of the structured annotations. In addition, all possible types of annotations are used, displaying the relevance of all constituent parts of the markup language. Further, the distribution of the annotations and their associated levels can be interpreted as a relative measure of complexity of the requirements.

4.6.2 Evaluation of Test Stage Similarity

Although the requirements and annotations are the focus of this work, the two test stages shall be compared independently as well. This section provides an analysis of how similar the two test stages, as presented by the log data they yield, are in general. This also influences the expressiveness of following evaluation steps. In particular, in the evaluation presented here, it is analyzed how well the simulation scenarios represent the ones encountered in the real driving data. The analysis is done regarding system-independent factors, namely the velocity and the acceleration of the vehicles and system-dependent factors, i.e. the warning level of the BSM system.

The factors, which the test stages are compared by, are defined based on domain knowledge and knowledge of the system which is established through the process of annotating the requirements specification. In this case, the system-independent factors are selected since they represent fundamental information regarding the characteristics of a trip with a vehicle. The system-dependent factor is chosen, since this system state turned out to be highly relevant within the annotation process. Therefore, the comparable factors can be chosen by a (domain) expert based on prior knowledge and are facilitated by the annotation of the system specifications.

Velocity Comparison Figure 7 shows the histograms of velocities and accelerations for the real data and the simulation as percentages of time for discretized values.

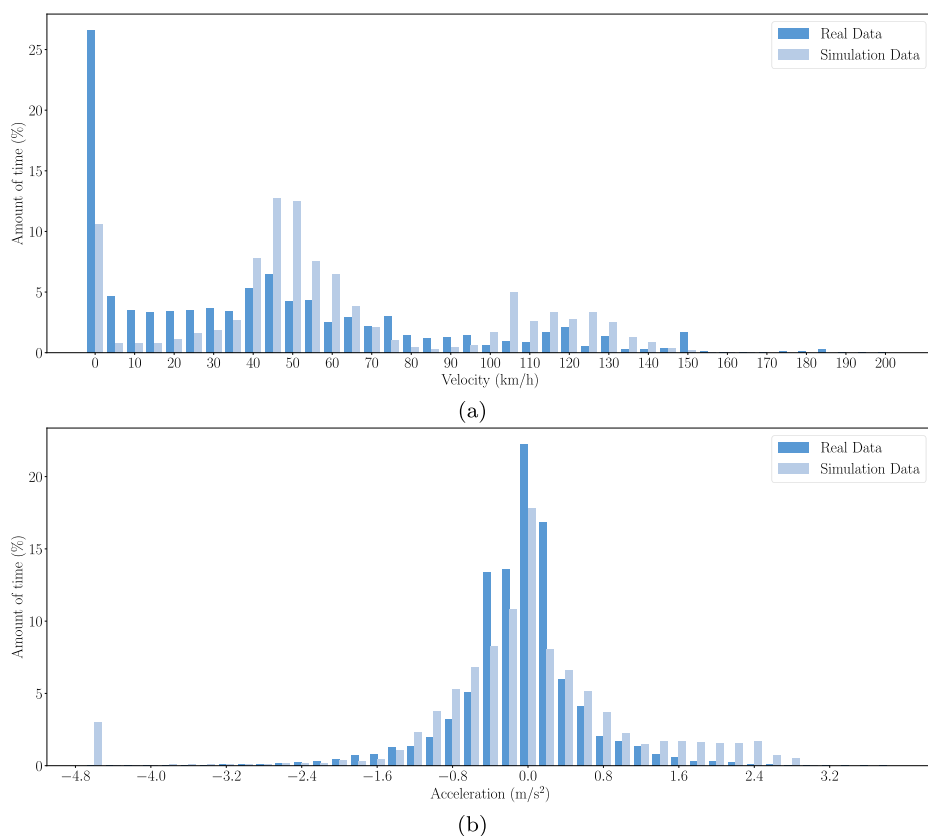


Fig. 7 Histograms of the **a** velocities and **b** accelerations within the simulation and real data

In Fig. 7a, the velocities are discretized into intervals of 5 km/h. It is very apparent that, in the real data, the vehicles spend more time in standstill since more than 25 % of the time the vehicle speed was between 0 and 5 km/h. The corresponding proportion for the simulated vehicles only amounts to around 10 %.

It can also be observed that the vehicles in the test drives generally travel at lower speeds compared to the simulation. The simulation exhibits approximately normally distributed velocities around the 50 km/h and 130 km/h marks, with a slight exception at 110 km/h. This suggests a deterministic driver behavior in the simulation, since these are the allowed speeds within the city, on the highway and on the highway when it is raining, respectively.

To further interpret the distributions of the velocities, Fig. 8a shows a Box-Whisker-Plot of the data. As conventionally applied, the box signifies the first and third quartiles, with the contained line being the median. The whiskers show the 5 and 95 percentiles. The descriptions as before are further supported by this plot, as the quartiles of the simulation data are shifted higher than the respective quartiles of the real data.

Interestingly, the 95 % marks are very similar with a difference of under 3 km/h. The shown medians are at 36.9 km/h and 56.6 km/h for real and simulation data, respectively. This is a significant deviation, which is further supported by the means and standard deviations of the velocities of the data, amounting to 43.0 km/h \pm 42.03 km/h and

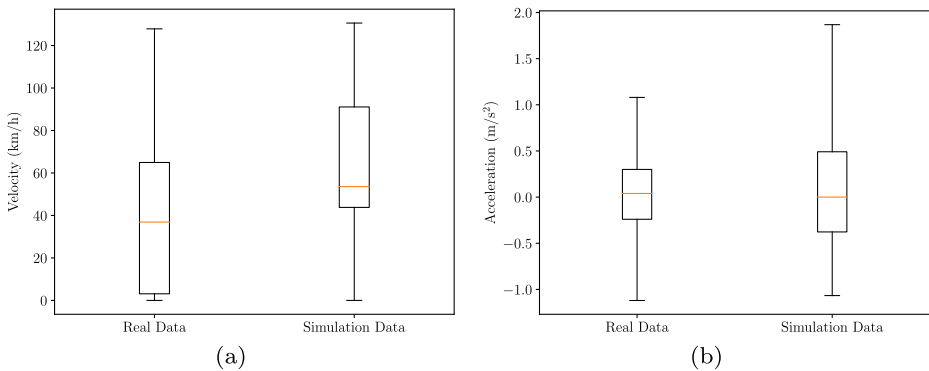


Fig. 8 Box-Whisker-Plot of the **a** velocities and **b** accelerations within the simulation and real data

62.33 km/h \pm 38.35 km/h for real and simulation data, respectively. The reported standard deviations further show how diverse and broadly spread the data is.

Eventually, the comparison of velocities yields some undeniable differences between real and simulation data, with each set of data exhibiting some peculiarities.

Acceleration Comparison Figure 7b shows the histogram of the vehicle accelerations for real data and simulation. The values are discretized into intervals of 0.2 m/s². Again, the simulated vehicles show a nearly normally distributed behavior around zero. An exception to this are the constant percentages for the values between 1.4 m/s² and 2.4 m/s², which also might exist due to deterministic behavior defined in the simulator. Additionally, there is a spike in occurrences at -4.6 m/s². This is most probably the defined value for a hard stop.

The acceleration values for the real data exhibit a similar distribution with a roughly normal distribution as well. The slight differences are apparent when observing the Box-Whisker-Plot in Fig. 8b. The quartiles of the simulation data cover a broader range of values, while the medians are very similar with 0.04 m/s² and 0.0 m/s² for real and simulation data, respectively. Further, the 95 % mark of the simulation data is higher than in the real data, which fits the observations from the histogram as described before.

The mean and standard deviation values further support this observation, with 0.021 ± 0.67 m/s² and 0.035 ± 1.24 m/s² for real and simulation data, respectively. The standard deviation of the simulation data is sensibly larger, as can be reproduced when looking at the quartiles in the Box-Whisker-Plot.

Warning Level Comparison The analysis so far is done by system-independent factors. Additionally, the scenarios can be compared by system-dependent factors. In this case, the exemplary comparison is done according to the warning level of the BSM.

A general analysis for every signal of a system which can be compared is the amount of time, which the signal attained a certain value. For the warning level, there are three possible values, 0 for no warning, 1 for a raised warning level and 2 for the highest warning level - as explained in Section 4.3. The share of total time that these signal values are assumed is visualized for real and simulation data in Fig. 9. It should be noted, that the scale of the axis starts at 90 % in order to highlight the small percentages for warning levels 1 and 2.

For the simulation as well as the real driving data, the vast majority of the time, the system is in warning level 0, which signifies an uncritical situation with no object in the blind spot of the vehicle. The warning levels 1 and 2 are differently distributed between

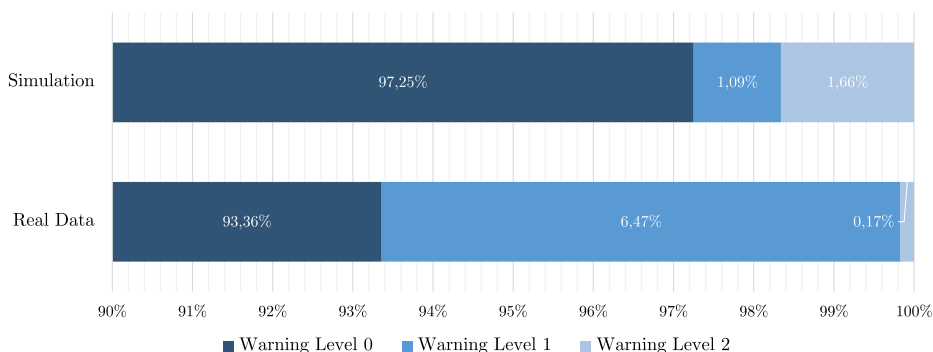


Fig. 9 Comparison of the different warning levels according to their activation time

the two data sources. In the real driving data, the warning level 1 is attained for just under 6.5 %, while in the simulation data, this level is only present in just above 1 % of the time. On warning level 2, the ratios are reversed. In the simulation, this warning level is present 1.66 % of the time, while in the real driving data, only 0.17 % of the time warning level 2 is reached.

To conclude, the warning level 1 is not as well covered in the simulation as it is in the real driving data. On warning level 2, the simulation shows more activation compared to the real data. This fact needs to be considered when analyzing the annotations and comparing the two test stages.

Interpretation of Results The presented results provide a baseline to answer the second research question, signifying the potential of the approach to assess the similarity of different test stages. Precisely, we conclude that there are not many similarities between the distribution of the analyzed signals of the real data and the simulation. However, this is not necessarily - and definitely not in this exemplary application - a disadvantage of one of the test stages. In a different setting, an analysis like this might yield very interesting insights into the quality of manually curated simulation scenarios or similar kinds of test stages.

4.6.3 Evaluation of Test Stage Compliance

To compare the two test stages with respect to the compliance with the specification, the annotations are used as a means of comparison. Therefore, the metrics of annotation and requirement coverage as defined in Section 4.5 are applied. This way, statements about the potential of the different test stages can be derived from the mapping.

The ratios of annotations that could be mapped to signals within the real and simulation data are shown with respect to their level in Fig. 10. As described in Section 4.4.2, the mapping is done based on the annotations on Level 3. Within the real driving data, almost 75 % of the annotations on Level 3 are mapped to signals with corresponding values. For the simulation, only about 47 % of the annotations could be mapped.

The presented ratios of Levels 1 and 2 in Fig. 10 are the direct results of the mapping on Level 3, since the Level 1 and 2 annotations are implicitly contained in the ones on Level 3. Therefore, the total number of mapped annotations on Levels 1 and 2 is the same, with 50 for the real data and 23 for the simulation data. Further, the total number of annotations on Level 1 and 2 is the same because this analysis is done in reference to the 73 annotations

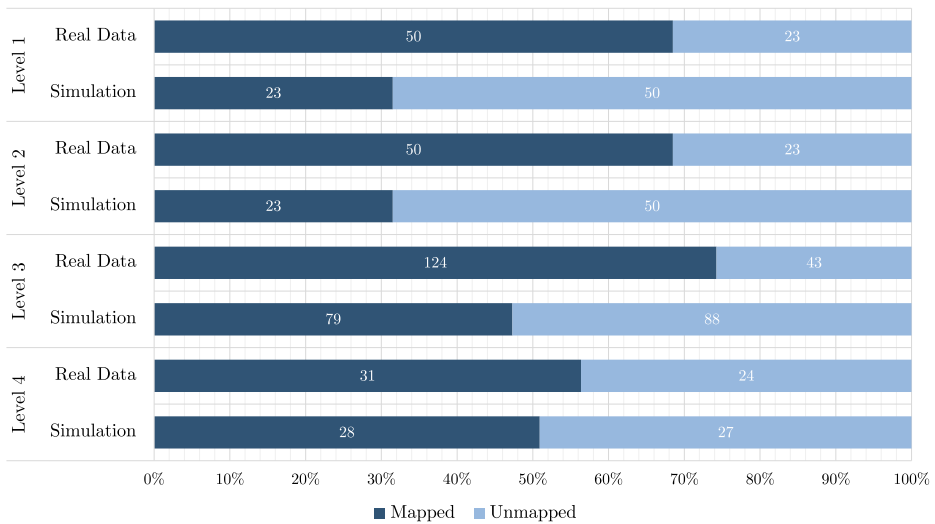


Fig. 10 Ratio of the mapped annotations for each level

on Level 1 and 2 which actually are refined on Level 3. Consequently, the ratio of mapped annotations is the same on Level 1 and 2.

The results on Level 4 also directly depend on the Level 3 mapping since any annotation on Level 4 is an aggregation of Level 3 annotations. Interestingly, despite the significant differences between real and simulation data on Level 3, the difference in the number of annotations with an associated mapping on Level 4 amounts to only three annotations.

Generally, the analysis shows that the ratio of testable annotations within the simulation for any level rarely exceeds the 50 % mark. The vehicle test stage shows substantially more potential.

The second metric to be reported is the requirements coverage, displayed in Fig. 11. The figure shows the ratios of requirements where either all contained annotations could be mapped (entirely mapped), only a subset of the annotations could be mapped (partially mapped) or no annotations could be mapped at all (entirely unmapped). As with the previous analysis, the results show a larger potential for the vehicle test stage, since more requirements can potentially be analyzed due an existing signal mapping. This is signified by the 62 % of the requirements showing the potential to be completely verified within the real data, and only 49 % within the simulation. The proportions of partially mapped requirements is very similar between the test stages, with a difference of just two annotations.

Interpretation of Results The presented results of the analysis signify the capability of this evaluation step in assessing the potential of different test stages - therefore answering the third research question. Explicitly, this evaluation shows, that the test stage where the requirements are verified with the real test drives shows more potential. In terms of annotation and requirement coverage, the vehicle test stage exceeds the simulation test stage.

The lack of potential in terms of annotations that could not be mapped to explicit signals or attributes within the simulation, can be further analyzed to identify improvements to make

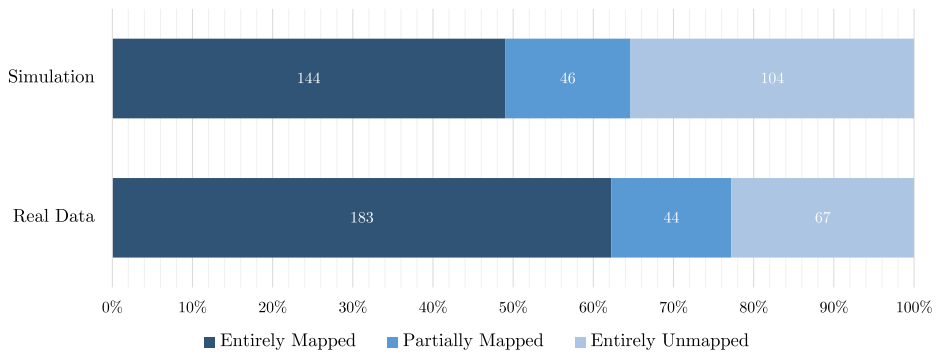


Fig. 11 Ratio of requirements with mapped annotations

the simulation more expressive and cover a larger part of the requirements specification. In our simulation for example, the status of the vehicle doors could not be simulated and, therefore, not be assessed to be reflected upon the annotations.

In terms of the real test drives, annotations that could not be mapped to specific signals signify that some aspects of the requirements specification are not explicitly represented as signals within the vehicle. In our application to the log data from an industry partner, this affected annotations like the detailed specification of construction measures like concrete piers or masts which the car is supposed to be recognizing with its camera. Abstract meta-information like this is not recorded as part of the vehicles internal signals and can therefore not be evaluated.

4.6.4 Evaluation of Test Case Alignment

This last evaluation explores how well the potential as defined in the foregoing section is actually exploited within the test cases. The test case now regards the actual execution of the test and the qualitative analysis, if the annotations are fulfilled for the different tests. This evaluation therefore explores, how well the tests align with the specification and further, how the different test stages differ in terms of their annotation and requirements coverage.

The annotation and requirements coverage as reported in this section are slightly differently constructed than before, as described in the analysis procedures in Section 4.5. Here the coverage defines the ratio of annotations that are actually fulfilled within a test. For the requirements coverage, these annotation are reflected upon the specifications to show how well these are covered.

To conduct these metrics, the evaluation of the log data in relation to the annotations is done for each vehicle in the simulation and every trip of the driving data. The henceforth reported results can therefore be interpreted in two ways: either as a means to evaluate the correct system behavior or to evaluate how well the test cases of a test stage cover all the characteristics of the system. Which of the two should be evaluated is to be defined and further examined by the engineer carrying out the testing.

To examine the annotation coverage, Fig. 12 shows the ratios of fulfilled and unfulfilled annotations per level. Additionally, the potential is illustrated by incorporating the number of annotations that are not mapped. This again signifies how much more potential is offered by the annotations, if more of them could be mapped to signals in the test stages.

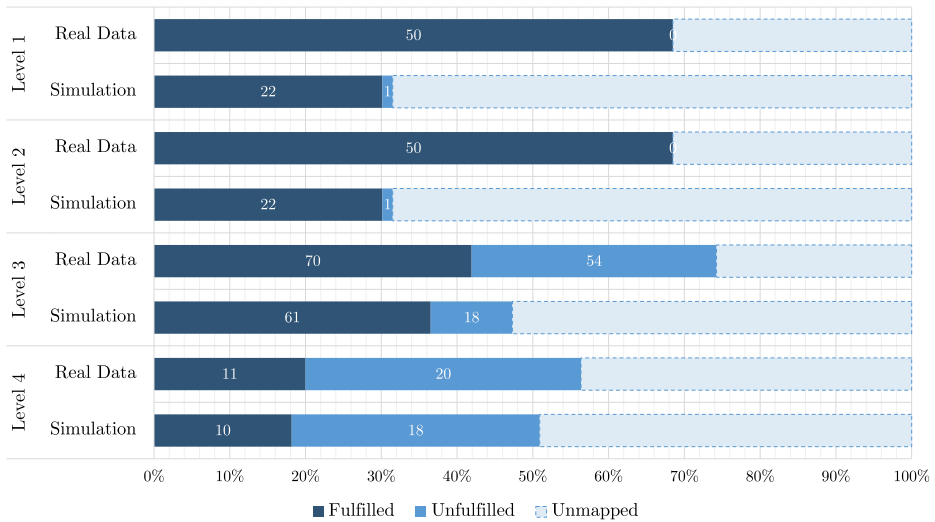


Fig. 12 Ratio of fulfilled annotations by level

On Levels 1 and 2, the test cases within the vehicle test stage accomplish their full potential. With regards to the simulated tests, just one of the annotations for the first two levels is not evaluated.

The analysis of the Level 3 annotations yields a different result. While in total, the number of fulfilled annotations within the real data is higher than in the simulation, the test cases within the vehicle test stage cover only 56 % of their theoretical potential. Within the simulation, the fulfilled annotations cover 77 % of the annotations that can actually be tested.

On Level 4 the two test stages show very similar fulfillment ratios, with 35 % and 36 % of the mapped annotations being fulfilled, signifying 11 and 10 actual annotations for real and simulation data.

Due to the results being referenced to the requirements specification, a statement about the fulfillment of the natural language requirements is facilitated. It should be noted that one requirement may contain fulfilled as well as unfulfilled annotations. Figure 13 reports

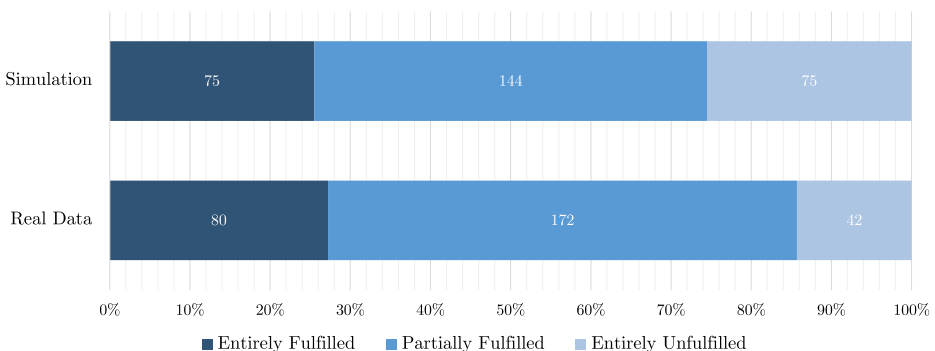


Fig. 13 Ratio of fulfilled requirements

the ratio of requirements with respect to the degree of fulfillment. It should be noted that as reference, only the requirements which actually contain annotations are used.

With regards to the performed simulation, 75 (25.5 %) of the 294 requirements exclusively contain fulfilled annotations. The evaluation of the real trips shows that 80 (27.2 %) of the considered requirements contain fulfilled annotations only. In contrast, 75 (25.5 %) requirements are entirely unfulfilled within the simulation with regards to the inherent annotations and 42 (14.3 %) with respect to real driving data. Therefore, the test drives cover more of the functional implication of the BSM, as the ratio of partially covered requirements is larger than in the simulation.

The annotations are further analyzed in terms of their fulfillment with respect to their level. Figure 12 shows the average percentage of fulfilled annotations per requirement for each level. It is apparent that when verifying the system requirements with the real driving data, the ratio of fulfilled annotations per requirement is higher than in the simulation - for every level.

Interpretation of Results The evaluation of test case alignment yields interesting insights into how extensive the tests within the different test stages verify the specification as presented by the annotations. These insights provide the answer to the fourth and final research question. The results show, that in this specific application, the test drives cover more of the peculiarities of the BSM system. These results are expected, as the previous analysis of potential already demonstrated an advantage of the vehicle test stage. This analysis is the ultimate objective of applying the markup language, since these are the results the engineer immediately reflects upon and uses for functional or test case optimization.

4.7 Discussion

We discuss the four different analysis approaches and the results in the following.

Large-Scale Application of Annotations The static analysis of the annotations, independent of the testing, demonstrates the general applicability of the lightweight multilevel markup language to large-scale requirements specifications. A significant proportion of the functional implications of the specification can be covered by means of the annotations. All possible specifications of the annotations on the different levels are applied in the process. Interestingly, the Level 4 annotations are only present in a small subset of requirements. An expert employee from the automotive industry partner, who is more familiar with the system, might have established more causal relations, but still, the trend is definitely identifiable. Additionally, it should be noted that this process is very subjective and every engineer will most probably produce different annotations. The final conclusion to be drawn is that the language scales to extensive specifications and provides a baseline for subsequent analyses as conducted in this study. The effort that we spend to annotate the requirements is acceptable (less than 6 hours) and presumably lower than the effort needed to create a formal model from a set of almost 300 requirements.

Test Stage Similarity The evaluation of general test stage similarity serves as a means to determine how well the simulation represents real driver behavior and traffic flow. In the presented application, the simulation differed from the real data to a considerable extent. Especially the velocities of the vehicles are distributed differently than in the real driving data. However, this does not necessarily impair the validity of the simulation. The fact that the top speed in the simulation is about 30 km/h lower than in real test drives has an impact.

No statement about the behavior of the system at these high speeds can be made when only testing the system with the simulation. Further, the comparison of system-dependent factors can yield important insights about the expressiveness and significance of the test stages. If, for example, a simulation shows a lot less activation of a certain signal in comparison to another test stage, the expressiveness of this simulation regarding this signal is to be questioned.

This analysis is a powerful means for assessing the validity of a test stage context, as the presented comparison illustrates. The focus for this paper does not lie in the actual, extensive comparison of the used simulation but aims at showing the potential of this approach. The presented extensive analysis signifies the applicability and importance of the evaluation of test stage similarity, as it serves as a relevant context for the subsequent analyses.

Test Stage Compliance From the results of the comparison of test stage compliance, two conclusions can be drawn. Firstly, it can be explained why the mapping of the real trips does not reach 100 %, even though the specifications, the communication matrix, and the implementation of the system come from one company. Since only internal signals are contained in the log data, annotations regarding the environment cannot be evaluated. For example, annotations such as `construction site` or `height of the cycle path $\geq 20\text{cm}$` could not be assigned. The analysis of annotations like these would need a more extensive database, where also statements from aggregated signals are present. This is a shortcoming of the test stage, which without the presented analysis, might not be as apparent to an engineer.

Secondly, the different levels of mapped text phrases make the test stage comparable. The simulation contains fewer signals because the simulator uses abstractions of vehicles. In our experiment, the vehicles are provided by the simulator SUMO. Annotations, for example, with the text `Doors`, `Exit warning` or `Bicycle = detected` cannot be mapped because the SUMO simulator does not provide such information. A second reason is the developmental stage of the system. The blind spot monitor in our simulation is a prototypical implementation and does not provide the full functionality of the original BSM.

In conclusion, the evaluation of test stage compliance yields an important statement about the system verification potential of a test stage.

Test Case Alignment As expected, the results of the test case alignment correlate with the results of the test stage compliance. As the vehicle test stage already showed a higher verification potential regarding the annotations, this manifests in the test case alignment. The tests in this test stage cover a larger number of the requirements and their contained annotations.

At this point, however, it is not the absolute numbers that are the focus, but the evaluation options that allow comparing two very complex test stages and their constituent test cases. The possibilities enabled by this test case alignment analysis are promising. Not only the comparison and support of a selection of a test stage is facilitated but also the in-depth optimization of test cases. The exploration of expressiveness of test cases in terms of requirements coverage is an important tool to systematically assess the functional verification power of the applied tests.

General Applicability of the Approach Our lightweight approach can be used to collect information regarding the testing procedure at every test stage. Given the annotations for a requirements specifications, the presented evaluations and analysis outputs are completely

automated. In addition to comprehensive evaluations based on natural language requirements, comprehensive evaluations of the test cases and stages are possible. This aligns with the verification approaches in the industry, where several test stages are necessary for a successful software development process. The evaluation of the test stage comparison enables the developers and testers to better compare development statuses and test options.

Not only can an engineer use the derived statements to address functional shortcomings of the system but also shortcomings within the testing itself.

Limitations Our approach is limited by some external aspects. The results displayed are highly dependent on the quality of the requirements. This includes ambiguous requirements that are misunderstood by the engineer resulting in annotations that cannot be mapped or results that are misinterpreted. Our approach only displays the results of the annotated text phrases. If important information is not documented as requirements, our approach may miss important aspects. Another limitation of the approach is the subjective annotation process. The lightweight language is deliberately a manual way for the engineer to select observable objects. For valuable results, domain knowledge is required during the annotation process. Since no check for annotation validity is provided, bad or false annotations will result in poor results.

In addition, complex requirements may not be marked because the language has too few elements for suitable annotations. As described in Section 3.1, the language is based in part on existing, established syntax. In our experiment, we were able to extract all the information we wanted. However, it is still possible that the language does not cover all aspects of the requirements.

5 Threats to Validity

There are several threats to the validity of this experimental evaluation which are subsequently discussed.

Internal Threats to Validity The first internal threat to validity is the custom made BSM function used within the simulation. A more reasonable application would have been to use the actual implementation of the system from the automotive industry partner. However, due to the productive code being subject to a non-disclosure clause, it could not be obtained to be used in this study. To mitigate this threat, the custom BSM module is constructed in tight compliance with the specifications of the system, to represent the actual functionality as closely as possible.

The second threat to internal validity is presented by the mapping of signals to the annotations. For the mapping of the signals and values of the system as present in the test stages to the annotations, an extensive domain knowledge is needed. The authors are involved with this project and the systems specification in a high level of detail. However, no engineer from the automotive partner was available to validate the mapping. To counter this lack of developer involvement, the mapping was assessed by consulting experts from our research facility.

Lastly, the most obvious potential threat to validity are the annotations themselves. However, as these are always subjective to the engineer introducing them to some degree, these are not regarded as an actual threat to validity. As long as the person annotating the specification has some background and expert knowledge of the system, the annotations are

regarded as reasonable. To further diminish the potential threat, the annotations in this study are always reviewed by two experts.

Construct Threat to Validity Additionally, there is one threat to construct validity, presented by the scenario selection. Ideally, the scenarios of different test stages should be aligned to some extent by default. This lack of pre-alignment manifests in the choice of simulation scenario, which - for comparability reasons - should be the same as the one in the test drives. However, in practice, this alignment is not necessarily given either. Therefore, a highly realistic simulation scenario was chosen to represent as realistic traffic as possible and therefore examine a realistic system behavior as well.

6 Conclusion and Outlook

With the increasing complexity of software-intensive systems, the importance of high-quality requirements specifications and their verification increases as well. This paper presents a lightweight multilevel markup language to annotate critical causal relations and correlations within the requirements. It is further demonstrated how these annotations can be connected to different test stages within the development process. In an extensive evaluation experiment, the proposed approach for connecting a test stage with the annotations and evaluating these jointly, is carried out.

The experimental evaluation shows the potential benefits of the presented approach. By analyzing a simulation or real test drive data with respect to the coverage of requirements, the quality of the respective test stage can be evaluated and - to some extent - quantified. This does not only improve the functional refinement of system features but also the optimization of the testing procedure itself.

The presented approach further integrates well into every phase of development, since the test stage used for verification is interchangeable due to an adaptive setup of the markup language and analysis life-cycle.

In the future, this approach shall be validated in an industrial setting, potentially at an automotive industry partner. Usability and the benefits for the developer shall be highlighted and analyzed in an extensive study. Further, the automation of the process in some key areas is planned. As already mentioned, the automatic extraction of systems states to automatically annotate parts of the requirements specification is already possible. This automation will be expanded to reduce manual labor and support the engineer as comprehensive as possible.

Acknowledgements Open Access funding provided by Projekt DEAL. We would like to thank Patrick Ebel for helping us implement the BSM system and the VSimRTI team for helping us set up the simulation environment.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Banks J, Carson JS, Nelson BL, Nicol DM (2000) Discrete-event system simulation. Prentice Hall, Upper Saddle River
- Barmi ZA, Ebrahimi AH, Feldt R (2011) Alignment of requirements specification and testing: A systematic mapping study. In: IEEE International conference on software testing, verification and validation workshops. IEEE. <https://doi.org/10.1109/ICSTW.2011.58>
- Bjarnason E, Runeson P, Borg M, Unterkalmsteiner M, Engström E, Regnell B, Sabaliauskaite G, Loconsole A, Gorschek T, Feldt R (2014) Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empir Softw Eng* 19(6):1809–1855. <https://doi.org/10.1007/s10664-013-9263-y>
- Cleland-Huang J, Chang CK, Christensen M (2003) Event-based traceability for managing evolutionary change. *IEEE Trans Softw Eng* 29(9):796–810. <https://doi.org/10.1109/TSE.2003.1232285>
- Codeca L, Frank R, Engel T (2015) Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In: 2015 IEEE Vehicular Networking Conference (VNC), pp 1–8. <https://doi.org/10.1109/VNC.2015.7385539>
- Dalpiaz F (2018) Requirements data sets (user stories). Mendeley Data, vol v1. <https://doi.org/10.17632/7zbk8zsd8y.1>
- DIN (2012) Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme. DIN EN 50128 DIN Deutsches Institut für Normen e.V., Berlin, Germany
- Eckhardt J, Vogelsang A, Femmer H, Mager P (2016) Challenging incompleteness of performance requirements by sentence patterns. In: 2016 IEEE 24th International Requirements Engineering Conference (RE), pp 46–55. <https://doi.org/10.1109/RE.2016.24>
- Galko C, Rossi R, Savatier X (2014) Vehicle-hardware-in-the-loop system for adas prototyping and validation. In: 2014 International conference on embedded computer systems: Architectures, Modeling, and Simulation (SAMOS XIV), pp 329–334. <https://doi.org/10.1109/SAMOS.2014.6893229>
- Gotel OCZ, Finkelstein CW (1994) An analysis of the requirements traceability problem. In: Proceedings of IEEE International Conference on Requirements Engineering, pp 94–101. <https://doi.org/10.1109/ICRE.1994.292398>
- Gunter CA, Gunter EL, Jackson M, Zave P (2000) A reference model for requirements and specifications. *IEEE Softw* 17(3):37–43. <https://doi.org/10.1109/52.896248>
- Hasling B, Goetz H, Beetz K (2008) Model based testing of system requirements using uml use case models. In: 2008 1st international conference on software testing, verification, and validation, pp 367–376. <https://doi.org/10.1109/ICST.2008.9>
- Huang W, Wang K, Yisheng L, Zhu F (2016) Autonomous vehicles testing methods review, pp 163–168. <https://doi.org/10.1109/ITSC.2016.7795548>
- ISO (2011) Road vehicles – Functional safety. ISO 26262 International Organization for Standardization, Geneva, Switzerland
- Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: 2005 international symposium on empirical software engineering, 2005., pp 10 pp.– <https://doi.org/10.1109/ISESE.2005.1541818>
- Kassab M, Neill C, Laplante P (2014) State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering* 10(4):235–241. <https://doi.org/10.1007/s11334-014-0232-4>
- Lopez PA, Behrisch M, Bieker-Walz L, Erdmann J, Flötteröd Y, Hilbrich R, Lücken L, Rummel J, Wagner P, Wießner E (2018) Microscopic traffic simulation using sumo. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp 2575–2582. <https://doi.org/10.1109/ITSC.2018.8569938>
- Lucassen G, Robeer M, Dalpiaz F, van der Werf J, Brinkkemper S (2017) Extracting conceptual models from user stories with visual narrator. *Requir Eng*, pp 1–20. <https://doi.org/10.1007/s00766-017-0270-1>
- Mavin A, Wilkinson P, Harwood A, Novak M (2009) Easy approach to requirements syntax (ears). In: 2009 17th IEEE International Requirements Engineering Conference, pp 317–322. <https://doi.org/10.1109/RE.2009.9>
- Mich L, Franch M, Novi Inverardi PL (2004) Market research for requirements analysis using linguistic tools. *Requir Eng* 9:40–56. <https://doi.org/10.1007/s00766-003-0179-8>
- Murukannaiyah PK, Ajmeri N, Singh MP (2017) Toward automating crowd re. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp 512–515. <https://doi.org/10.1109/RE.2017.74>
- Nebut C, Fleurey F, Le Traon Y, Jezequel J (2006) Automatic test generation: a use case driven approach. *IEEE Trans Softw Eng* 32(3):140–155. <https://doi.org/10.1109/TSE.2006.22>

- Protzmann R, Schünemann B, Radusch I (2017) Simulation of convergent networks for intelligent transport systems with vsimrti. *Networking Simulation for Intelligent Transportation Systems: High Mobile Wireless Nodes*, pp 1–28
- Pudlitz F, Brokhausen F, Vogelsang A (2019a) Extraction of system states from natural language requirements. In: 27th IEEE International Requirements Engineering Conference (RE), <https://doi.org/10.14279/depositonce-8717>
- Pudlitz F, Vogelsang A, Brokhausen F (2019b) A lightweight multilevel markup language for connecting software requirements and simulations. In: Knauss E, Goedicke M (eds) *Requirements engineering: foundation for software quality*. Springer International Publishing, Cham, pp 151–166. https://doi.org/10.1007/978-3-030-15538-4_11
- Ramesh B, Jarke M (2001) Toward reference models for requirements traceability. *IEEE Trans Softw Eng* 27(1):58–93. <https://doi.org/10.1109/32.895989>
- RTCA (2012) Software considerations in airborne systems and equipment certification. DO- 178c, RTCA and EUROCAE
- Utting M, Pretschner A, Legeard B (2012) A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 22(5):297–312. <https://doi.org/10.1002/stvr.456>
- Uusitalo EJ, Komssi M, Kauppinen M, Davis AM (2008) Linking requirements and testing in practice. In: IEEE International Requirements Engineering Conference (RE), pp 265–270. <https://doi.org/10.1109/RE.2008.30>
- Whalen MW, Rajan A, Heimdahl MP, Miller SP (2006) Coverage metrics for requirements-based testing. In: *International Symposium on Software Testing and Analysis (ISTA)*, pp 25–36,
- Wüest D, Seyff N, Glinz M (2019) Flexisketch: a lightweight sketching and metamodeling approach for end-users. *Software & Systems Modeling* 18(2):1513–1541. <https://doi.org/10.1007/s10270-017-0623-8>
- Zhu H, Hall PAV, May JHR (1997) Software unit test coverage and adequacy. *ACM Comput Surv* 29(4):366–427. <https://doi.org/10.1145/267580.267590>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Florian Pudlitz is a research associate at the department of Automated Systems Engineering Technologies at TU Berlin. At the end of his university education (B.Sc. and M.Sc. at TU), he dealt with automated extraction of modes in natural language requirements. Continuing this approach, he now focusses on linking requirement specifications to simulations. During his research, he has also dealt with ML, automation in industrial testing, and cyber-physical systems. For international co-operation and research exchange, he visited Chalmers University in Gothenburg in May 2019. Research results have been published at REFSQ'19 and RE'19.



Florian Brokhausen is a research associate at the Technische Universität Berlin at the department of Automated Systems Engineering Technologies. He received his M.Sc. in Automotive Systems from the same university and his B.Eng. from the Baden-Wuerttemberg Cooperative State University (DHBW). During his integrated Bachelor studies, he has already worked for an international OEM in the automotive sector. His research interests include continuous experimentation, applications of ML in software engineering and (automotive) big data analysis.



Andreas Vogelsang is an assistant professor (junior professor) for software engineering at the Technische Universität Berlin. He leads the software engineering group at the Daimler Center for Automotive IT Innovations (DCAITI). He received a PhD from the Technical University of Munich. His research interests comprise requirements engineering, model-based systems engineering, and software architectures for embedded systems. He has published his research in international journals and conferences such as IEEE Software, ICSE, and RE. In 2018, he was appointed as Junior-Fellow of the German Society for Informatics (GI). Further information can be obtained from <https://www.aset.tuberlin.de>.