Constructive Requirements Modeling – More Reliable Implementations in a Shorter Time

Christian Berger[1], Sebastian Siegl[2]

[1]Chalmers | University of Gothenburg, Sweden
[2]Audi Electronics Venture GmbH, Ingolstadt, Germany

christian.berger@chalmers.se
sebastian.siegl@audi.de

Abstract: Requirements engineering is nowadays the broadly accepted method to manage customer's requirements. The result is a specification from which a solution is implemented and which is used to validate the realization in terms of their fulfillment. However, today's tools assist in organizing and tracking the requirements but reliable criteria about their completeness, consistency, and realizability are missing. Furthermore, the resulting artifact is a document, which must be read and understood by humans, which itself is error-prone. It is obvious that errors and ambiguities result in an unwanted solution which is often and in the worst case only discovered in the final stage: Testing. This paper outlines an approach for constructive requirements modeling, which describes completely customer's demands in a formal manner so that already during the requirements' elicitation inconsistencies are eliminated, completeness is assessed, realizability is ensured, and all valid test cases can be derived by using a model-based testing approach. Therefore, we propose adaptions to the traditional V-model to not only save valuable development and testing time but also to achieve better results. The applicability is shown on the example of the software for an auxiliary heating system at a large German OEM.

1 Introduction and Motivation

Today's comfort and safety systems are not limited to the premium segment anymore. Instead, they are available to a cost-elastic and very competitive market. Driven by competitors, which equip their vehicles for the mass-market with more and more software-intense vehicle functions on the one hand but focused to limit the increasing development costs on the other hand, OEMs demand appropriate methods to master the overall costs while focusing on the quality of a vehicle function. According to [KBP01, Sel07], the later a manufacturer focuses on quality and correctness of a system the higher are the costs – often orders of magnitudes depending on the current development stage. Regarding the widely and often in iteration-cycles used V-model, an OEM applies

quality methods during or after the implementation stage first to determine the fulfillment of the specified requirements.

Thus, today's requirements engineering tools provide sophisticated assistance for managing a customer's requirements on the one hand [GNA+12]; furthermore, a specific requirement can be associated with one or many test cases to testify and track its fulfillment. Thus, the quality manager is always and at any time informed about the current nominal quality of the implementation.

However on the other hand, the consistency, completeness, realizability, and thus the actual quality of requirements is not assessed at all. Considering today's increasingly complex software systems, which demand a more elaborated software engineering [PBKS07] and therefore methods for quality assurance of these software-intense vehicle functions [RBK06, BBHR07, BBKR09, Ber10, BR12], the risk also increases to oversee possible situations with which a vehicle function could be faced and which could threat participants within the system context like pedestrians or bicyclists. However, this fact is not acceptable for safety systems in a customer product like vehicles!

To cope with these risks, OEMs and automotive suppliers are required to develop according to the ISO-26262 [ISO11], which defines obligatory processes and methods regarding the defined Automotive Safety Integrity Level (A-SIL); the ISO-26262 explicitly demands for software safety requirements that they must be "comprehensible, precise, unambiguous, verifiable, and realizable" and states furthermore that "traceability and adequateness regarding the SIL level and architecture" [p.99, ISO11] must be ensured. However, only semi-formal methods are "highly recommended" and thus, rather mandatory w.r.t. a specific A-SIL level.

In contrast, the avionics sector [DO11] recommends explicitly formal methods depending on the effect of a possible failure to extend classical testing approaches. And only in the railway sector [EN01], formal methods are highly recommended for using as a relevant technique even during the requirements specification.

Advantages of formal models are for example their mathematical proof of consistency, completeness, or realizability. Thus, the application of formal models to a requirements specification could significantly improve their quality due to these characteristics before its actual implementation is started. Furthermore, available methods from the requirements engineering management can be reused, for example traceability for single requirements or exchangeability with suppliers. However, there are still some concerns about formal methods, like additional costs, increased time for their application, and the lack of adequate tooling.

In this contribution, the method of sequence-based specification (SBS) [PP03, Car09] is described as a formal method to support the process of requirements analysis and formalization to achieve the aforementioned advantages. Therefore, we propose to create an additional artifact besides the traditional requirements document during the first stage in the development, which is verified and serves all later stages of the development process. Furthermore, we suggest to rethink the traditional V-model based development process to explicitly rely on this additionally created artifact to shorten development and

testing time on the one hand; on the other hand, the aforementioned characteristics of formal models yield to more reliable results.

The paper is structured as follows: First, the applicability of formal methods during the development of automotive software-intense systems is discussed. Next, theoretical principles of the SBS are presented. Afterwards, a revised V-model based development process that explicitly embraces the resulting artifacts from the SBS is proposed, which addresses the mentioned caveats. The process itself was successfully applied during the development of an auxiliary heating system, which is briefly outlined in the end.

2 Related Work

Applying formal methods during the requirements specification of software systems is known for more than two decades. Already in [GCR94], the application of formal methods is described for four different systems: First, the use of formal methods to specify a shutdown system for a nuclear plant; next, their usage for verification and validation of a safety-critical signaling system for the subway system of Paris to reduce the time gap between two trains from 150s to 120s; then, formal methods in terms of graphical notations to specify subsystems of a traffic alert and collision avoidance system for planes; and finally, formal methods were required within a project from the US Department of Defense for security-critical networks.

Thus, already at that time, the importance of formal methods was recognized to achieve the common goal: To increase the customer's confidence in the resulting implementations for these different, complex, and safety-critical systems. Nothing less than the same goal from the customer's perspective applies for modern safety-critical systems that are built into a mass-product like vehicles! However yet today, the aforementioned concerns about formal methods were basically confirmed by [Kon11] who carried out interviews to gather information about software testing and practical application of formal methods in automotive projects.

Although the ISO-26262 is in effect since November 2011, discussions about adequate formal methods, which comply with this standard, are still insufficient. One reason could be their limited recommendation within the standard as already mentioned. Moreover, recent publications like [CJL+08] focus only on annotation support for requirements as well as externally carried out verifications on top of AUTOSAR to support the requirements' traceability.

The authors of [PHP11] propose an approach to check automatically the vacuity of requirements. Compared to the SBS, it is a subsequent processing approach to ensure the validity of the resulting requirements specification, while SBS is constructing this specification at first.

Previous work to this paper which also bases on SBS was outlined in [SHG10] and [SHGB11]. While the former outlines the successful application of SBS during Hardware-in-the-Loop (HiL) testing on the example of an energy management system,

the latter deals with the successful application of SBS already during Model-in-the-Loop (MiL) testing of an automotive safety system in pre-series development project already at the early development stages.

The results of that previous work encouraged us to focus on the development process itself and to rethink the usage of formally modeled requirements already at the very earliest stage instead of their usage during MiL- or even HiL-testing only. Thus, as an extension to our previous work, this contribution focuses on revising the traditional and widely accepted V-model based development process to tackle the increasing complexity of software-intense systems by relying on constructive requirements modeling using SBS.

3 Constructive Requirements Modeling

The fundamental idea behind the SBS and its method to create an executable specification is described in the following. It is a constructive method, which means that properties like completeness of the resulting work products do not need to be verified by additional activities. They are correct w.r.t. the initially given preconditions and assumptions because of the rules that were followed to build the resulting artifacts. Hence, the method must be strict and rigid if properties such as completeness, correctness, realizability, and consistency are to be assured in the construction process. SBS itself is a rigid procedure to create a formal software requirement specification (SRS) [Car09, PP03]. Thus, the resulting artifact is consistent, complete, and traceably correct due the rules used for its creation. However, due to the fact that human error may occur during the application of SBS, an additional review afterwards helps to reduce this potential risk.

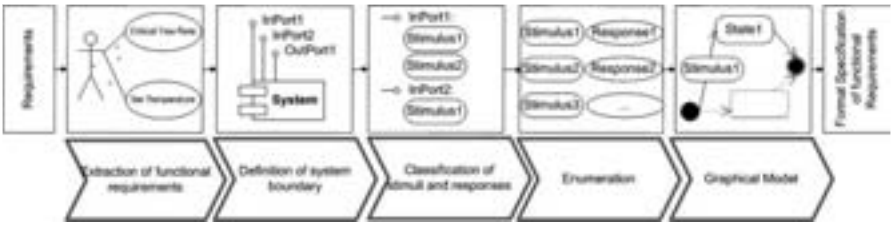

Figure 1: The manual steps in the Sequence-Based Specification process: Activities and artifacts.

The first step is the definition of the system boundary to derive all interfaces for input and output signals. Based on the requirements, all stimuli from these interfaces are determined. This comprises stimuli S that represent operations and responses R that are received as reactions from the system under development (SUD). A special stimulus is the empty event λ, which is also the starting point of the enumeration. In the automotive

domain, stimuli can originate in other electronic control units (ECU), in sensors on the same ECU, or in the system's environment.

In Figure 1, all required steps of the SBS process are shown:

- The functional requirements are identified and extracted. They are tagged for their traceability through the SBS and the following entire development.

- The SUD is considered as a Black Box. The SUD's system boundary is defined by its interfaces for information exchange.

- Stimuli (i.e. inputs to the SUD) and responses (i.e. information received from the SUD) are collected and classified. The classification is done on basis of the requirements w.r.t. the expected equivalent responses. The result is a set of classified inputs and outputs of the SUD.

- All possible stimulus sequences are completely and systematically enumerated and elongated. The method starts with the empty sequence λ of length zero, continuing with length one, two, and so on. Thus, it is assured that all possible combinations and permutations from the input domain are considered and related with the expected SUD's behavior. This enumeration process follows strict rules:

    - The stimulus extension is checked whether it is legal. Under certain conditions (e.g. when it is physically impossible to occur), an extension is considered as *illegal*. Then this sequence does not need to be considered anymore in the following iteration.

    - At each stimulus extension, the expected responses from the SUD are identified from the requirements and assigned to this stimulus sequence for later traceability.

    - After each stimulus extension, it is checked whether the expected behavior of the SUD w.r.t. all possible further extensions is equivalent to an already existing sequence. If such a sequence can be found, the current sequence can be mapped to the other one and the enumeration is continued for both in a common manner. This is called *reduction*.

- The process stops if all sequences are enumerated and elongated systematically, the corresponding responses are assigned for all sequences, and no further extension is possible.

It is to mention that each step – from the definition of the system boundary to the end of the enumeration – requires a reference to a requirement. Thus, the traceable correctness of the final SRS is assured. Furthermore, ambiguous or missing requirements are identified and all of them are validated w.r.t. their functional realizability.

All sequences that could not be reduced to another sequence are called *canonical sequences*. These sequences define the resulting state space for the SUD, and hence for the software implementation to realize the final system.

On the top layer, the states are considered as components of the SUD, as the states are on a high abstraction layer. Hence, the result is a first functional partitioning, from which the architecture and the deployment of components is derived. In the next step, the SBS is applied to each component, providing a design und implementation for it. The process of refinement is carried out recursively until a layer is reached on which the implementation is supplemented by manual coding or by model-based code generation e.g. from MATLAB/Simulink.

The first two steps are presented in Figure 2. SBS is used to derive an initial architecture of the SUD. On each component SBS is applied once again to generate a first design for each component. With the application of the SBS in each development step, properties such as completeness and traceable correctness can be assured down to an implementation artifact. Therefore, it is reasonable to adjust the development process to rely on the artifact SBS, which is described in the following section.

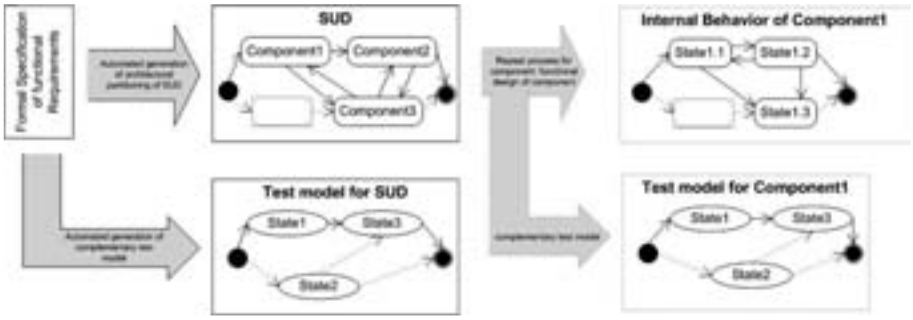

Figure 2: Recursive process for the usage of the SBS for the architecture and design.

From the artifact's point of view, the SBS process creates an executable state machine for the implementation as well as its complementary environmental model. The environmental model serves as a test model, from which any possible usage sequence can be derived, which serves as a valid and meaningful test case for using within MiL- or HiL-testing. In [Sie11], the development process for the test model with a systematic integration of time and timing requirements is described in greater detail. The executable state machine for the implementation can be simulated for validation and to assess the system's realizability. The executable state machine is equivalent to a Mealy automaton with further annotations.

The method of SBS can be applied to virtually any system that can finally be broken down to discrete signals and states. It demands skills to choose the appropriate level of abstraction, a suitable hierarchization, and to decide, what a state embraces. A decision has to be taken also on the question where SBS is applied to refine states and where

other methods, e.g. temporal logic, are used. This decision clearly depends also on the type of sub-functionality under consideration, which can be a reactive controller or logic.

4 Proposal for a Revised Development Process for Software-Intense Vehicle Functions

According to [Sel07] and which is also confirmed by countless projects, the later a failure is discovered the higher are the costs for identifying and fixing its fault. Moreover and also shown in the aforementioned publication, 40% of the top ten project risks ("4. requirements mismatch", "5. user interface[1] mismatch", "6. architecture", "7. requirements changes") are related to requirements, architecture, and design, and thus within the early stages during the development.

In Figure 3, the V-model is depicted which is annotated not only by the exponentially increasing costs to fix discovered faults but also by the mentioned project risks. Thus, it is obvious that the uniformly distributed effort over the project runtime that is suggested by the design of the V-model differs from the reality's facts; the risks are left-skewed but their fixing efforts and thus the related costs are right-skewed.

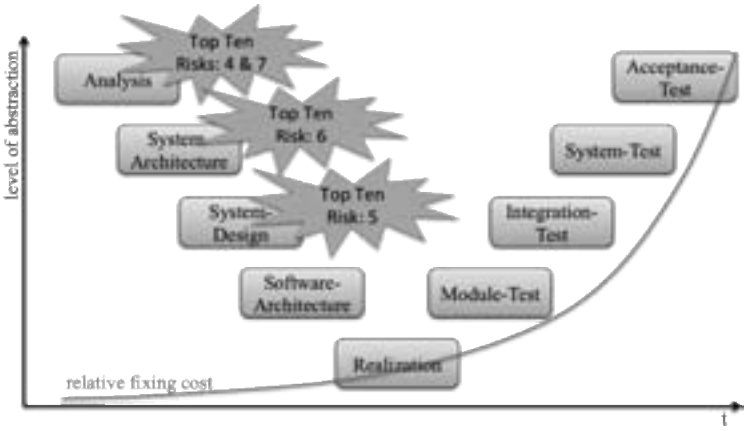

Figure 3: V-model, which is annotated by relative fixing costs and the 40% dominating project risks according to [Sel07].

However, any development approach, which is following this V-model, consists of consecutive steps, which depend on the previous one. This means in practice after finishing the requirements specification, the design and modeling of the architecture and the final implementation are started, which reveal sometimes inconsistencies or even realizability deficiencies in the specification. The subsequent step after the realization is the testing phase, which itself uses the specification again to unveil implementation errors. This step-wise and consecutive approach obviously moves the quality gate to the

[1] „user interface" includes not only system usage which is realized by an HMI but also technical interfaces which are used by neighbouring systems.

latest phase of the development where fixing inconsistencies and errors in a specification and implementation are most expensive. Furthermore, only implementation errors w.r.t. the specification are discovered but the quality of the specification itself like correctness, consistency, and realizability is hardly assessed.

But how can this widely realized [Cha05] but still accepted problem be overcome? Intuitionally, the quality gates must be moved to the earliest and reasonable phase during the development: For any system development project this is the point where the requirements specification is created and used to start the implementation. Obviously, any inconsistencies in this document will result in an erroneous implementation, which is only or even not in the worst case unveiled by test cases. Thus, the right-skewed distributed effort and costs must be corrected to result in a left-skewed distributed effort with a strong focus on the actual requirements specification in the best case.

Therefore, a combined method to act as quality gate, which serves both to validate the requirements specification and to support the implementation, is necessary. This method shall therefore provide consistent and complete test cases to reduce the test effort on the one hand; on the other hand, the method could also provide implementation artifacts directly. Thus, this method serves as so-called single-point-of-truth (SPoT) which must formally guide the process of scrutinizing and validating the requirements and their interrelations but which also must act as source model for generating implementation artifacts or test cases according to a project's specific needs.

At this point, the method of constructive requirements modeling as outlined in the aforementioned section comes into play. The constructed formal model, which fundamentally is an annotated Mealy automaton, contains by principle all valid usage scenarios for a system. As outlined, this is achieved by a systematic enumeration of the system's input signals combined with the expected system reaction according to the initial requirements specification. Thus, the requirements specification is validated iteratively in terms of consistency, correctness, and realizability during the analysis stage in a guided manner.

Due to the fact that each state in the resulting Mealy automaton also contains the expected system's output, the formal requirements model can serve either as source model to generate first implementation artifacts on the one hand, or to provide input signal sequences to derive only valid and meaningful test cases on the other hand. Their usage is shown in Figure 4, which results in a revised and slightly left-skewed variant of the V-model. These modifications are strongly encouraged by the results from [SHG10] and [SHGB11].

Even if the V-model is widely used in the automotive domain for the development of vehicle functions, the outlined method can also be used with modern agile methods in iterative contexts. This is especially interesting when providing test cases for MiL testing as described in [SHGB11]. Thus, prototypical implementation models can iteratively be improved.

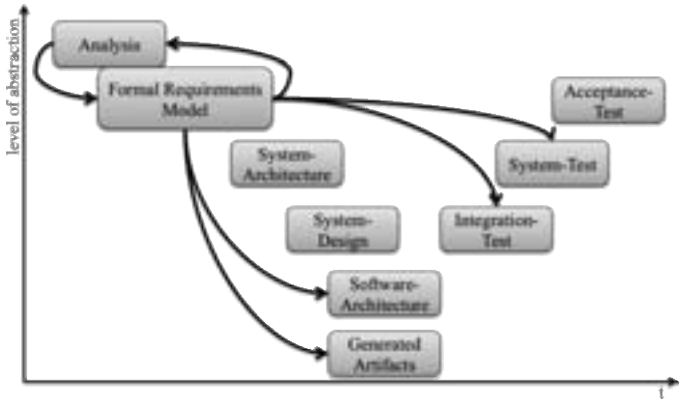

Figure 4: Revised V-model with a strong focus on a formal requirements specification model from which implementation and testing artifacts are generated automatically to reduce implementation and testing efforts.

In practice, each transition of the Mealy automaton can also be weighted with a probability to reflect more likely usage profiles and to direct the test case generation [SL10]. Furthermore, these generated test cases can be directly used during a manual prototyping phase to enable and support a "test first" approach for MiL-testing, while getting further insights from the prototype. In the end, these requirements models enable a quality gate at the very earliest time point to save valuable resources later on in the V-model development process. Furthermore, artifacts from suppliers can be validated objectively and even automatically.

5 Case Study: Auxiliary Heating System

In this section, the previously described revised development process was applied during the software development of an auxiliary heating system. These systems are mainly used in road vehicles to heat the passenger compartment and to defrost the windows. In hybrid or electric vehicles, their usage is extended for components of power transmission and power supplies.

In the following, the conventional system for the passenger compartment is considered. Depending on the customer's choice of a temperature, the auxiliary heating system determines whether it is necessary to heat or to cool. Based on the environmental conditions it starts at a calculated time with its activity. During the phase of heating or cooling it is monitoring the environment as well as the compartment w.r.t. changes of the temperature. If necessary, it adapts its activity to the changed conditions. The system contains both discrete and continuous behavior.

Furthermore, in recent vehicles there are a lot of possibilities to control the auxiliary heating system and air conditioning. It is possible to program timers via the key, the

Multimedia Interface (MMI), or directly at the climate control. Moreover, specific settings can be overridden by user actions that have higher priority.

5.1 Application of the SBS to the Auxiliary Heating System

As outlined in the revised V-model development process, we applied the SBS already at the very early stage to derive a formal requirement specification from initial requirements of the auxiliary heating system. A high functional level of abstraction was chosen as the purpose of the model was also the derivation of test cases for integration testing. Therefore, different sources had to be brought into the process to formalize the requirements, e.g. contents from a requirement management system, sketches and minutes from meetings, and also experts for the functionality who were in charge of the final product. After the analysis and clarification of the requirements, the models as described in Section 3 could be created.

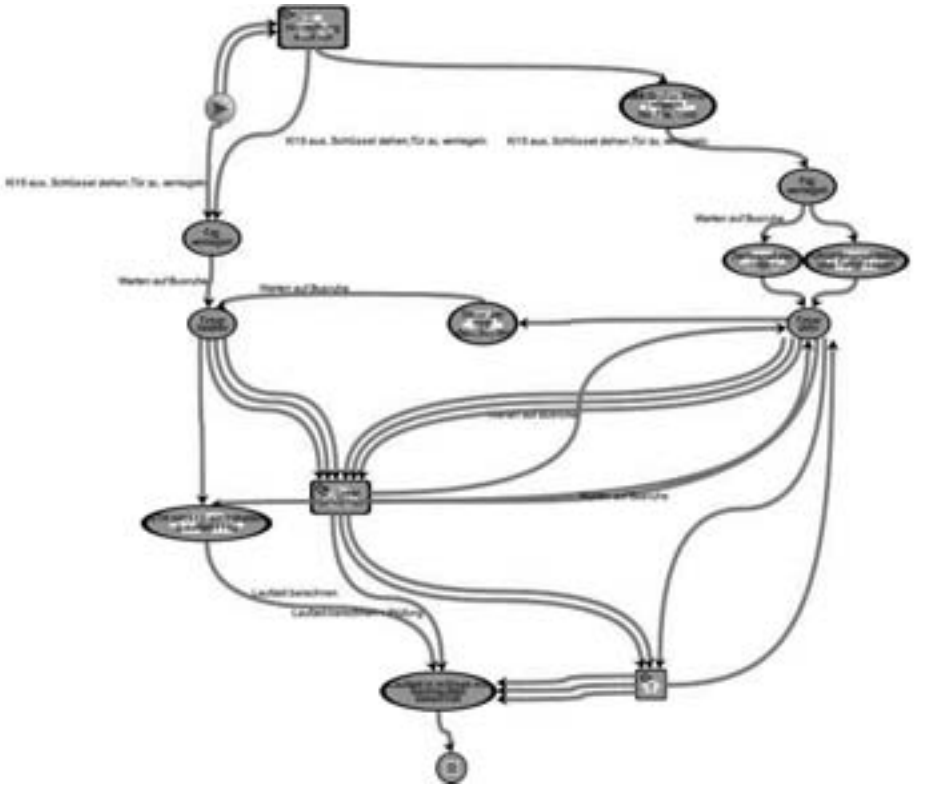

Figure 5: Detailed model for the auxiliary heating functionality. This model was used during integration testing on the ECU.

To give an idea of the size and complexity, the following figures of the model for the auxiliary heating system are given: On the top layer the model consists of five states. Three of them, i.e. all except the start and final state, are refined. As a result, the

implementation consists of three components that interact with each other. On the next layer the SBS process is applied to each component. The result consists of 39 states with 52 transitions. These figures provide quite a good idea of the range, in which the size of the model diagrams resulted.

As described in Section 3, the internal behavior and the appropriate test model are derived from the SRS model. In Figure 5, a detail of the test model is depicted, which was derived on a higher abstraction layer for integration testing on the final ECU. The abstraction on this layer is determined by the interfaces provided through the target ECU and its bus interfaces. For the sake of clarity, the states are grouped. The rectangular boxes in Figure 5 are grouped states, in which possible behavior with the same functional purpose is specified. Furthermore, the grey and white states are further refined. These states can be entered and the internal behavior of the state is specified in a new sub-diagram.

The process was applied until a level of abstraction was reached which was sufficiently detailed. The complete automaton, i.e. the top model including all states and transitions from the refinements, consists of 796 states and 1,019 transitions. The average length of one usage sequence is 66 transitions, which assumes an operational profile with uniformly distributed probabilities.

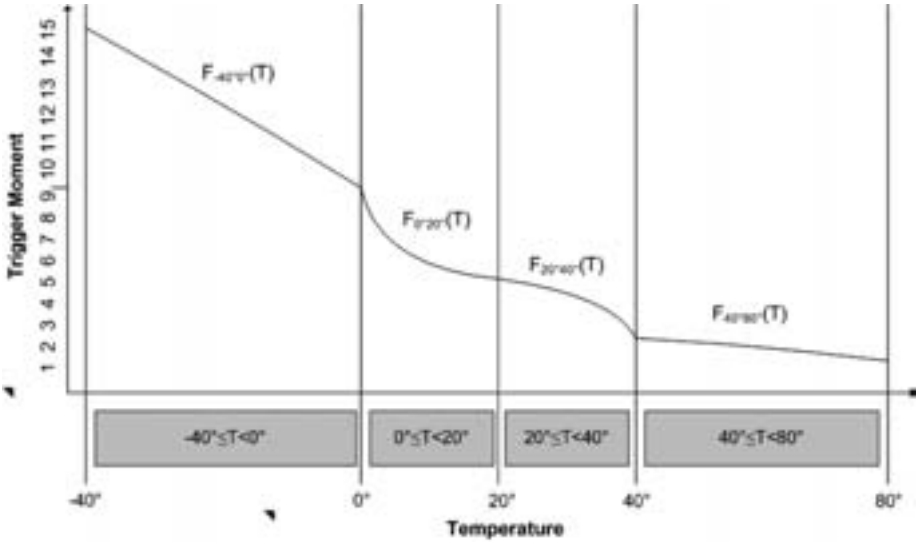

Figure 6: Example of continuous functions that are mapped to discrete states during the SBS process.

The system's continuous behavior is mapped to discrete states. Continuous functions are used to specify the values between discrete states. In Figure 6, the function e.g. F0°20°(T) provides all continuous values between temperatures including 0° and excluding 20°. The abscissa represents the temperature in degree Celsius and the ordinate describes the trigger momentum for an actuator in the heating system. As illustrated in that figure, the continuous transition between discrete states is specified

with functions, and the discrete states serve to span the state space of the automaton and for e.g. the calculation of metrics.

5.2 Results

As expected and described in Section 4, ambiguities in the requirement specification could be identified and solved during the formalization process. The deficiencies were apparently related to the negligence of usage scenarios, which were represented by systematically enumerating possible stimuli sequences that might occur in the field use. Thus, error-prone interpretations during the implementation could be avoided in a systematic manner and test engineers already had a high confidence in the realization. Furthermore, they could easily derive test cases for their model-based testing process directly from the SRS model.

As there are many phases of the development process between the construction of the models and the testing of the integrated system, a variety of potential errors can be detected. In between are tool changes, code processing, and additional basic software modules in the final ECU for e.g. the I/O communication to the SUT.

In contrast to the traditional V-model-based development process, informal specification documents did not provide clear, complete, and consistent answers to all expected usage scenarios for the auxiliary heating system. Thus, the responsible engineers for the functionality were involved to clarify these issues. Thus, the initial requirements could be revised thoroughly and systematically to get a complete and consistent formal model.

As already mentioned, these resulting SRS models also served to improve the testing of the implementation. Additional errors could be identified, that had not been discovered before. They were mainly related to neglected possible usage scenarios, which had not been tested and on the other hand, to deficiencies in the requirement specification, that led to vagueness in the interpretation on the implementation and test side. The acceptance of the users was high because finally the created model served as basis for discussion and communication. Encouraged by the results of this project, the revised V-model based development process shall be applied for new functionalities on a project for thermo-management.

6 Conclusion and Outlook

Formal methods are not the one and only Holy Grail to tackle the increasing complexity of upcoming software-intense systems; using a formally derived artifact at the very early stage does not automatically mean that it specifies the “right” requirements. However, and that is the important aspect, it is assured that all following stages rely on a consistent, complete, and realizable model, which serves as the single-point-of-truth in an artifact-driven development process, and thus, error-prone human interpretations can be significantly reduced or even avoided.

Nevertheless, formal methods contribute mainly to reduce the probability to discover unexpected errors during later stages in traditional development processes. As shown in [SHGB11], even manageable embedded systems with a sophisticated testing process at first sight did benefit from the SBS approach – already at early stages by supporting the MiL-testing. As a consequent step, this contribution outlines a revised V-model based development process, which explicitly embraces a method to constructively model requirements in a formally consistent, complete, and realizable manner at the earliest possible stage. Its applicability and benefits were demonstrated on the example of the requirements elicitation and development of an auxiliary heating system.

In the future, formal methods need to be rather mandatory and not only limitedly recommended as stated in the ISO-26262 standard due to two reasons: At first, the abandonment would provoke the risk of lately discovered errors which are very expensive in terms of product recalls for example; second, the customer expects a reliably and thoroughly engineered and operating software-driven system. However, future work must be carried out to investigate the expressiveness of applicable formal methods for requirements specification regarding the type of embedded systems in terms of continuous signal processing or event-driven system. Moreover, the underlying characteristics of a system's input signals should be investigated to provide guidelines for applicability and to derive best practices. Furthermore and as already mentioned as concerns in Section 1, appropriate and user-friendly tooling is necessary to increase the overall acceptance among developers and for test and requirements engineers.

References

[BBHR07] C. Basarke, C. Berger, K. Homeier, and B. Rumpe, "Design and Quality Assurance of Intelligent Vehicle Functions in the 'Virtual Vehicle'," *Proceedings of 11. Automobiltechnische Konferenz – Virtual Vehicle Creation, Stuttgart*, vol. 9, 2007.

[BBKR09] A. Bartels, C. Berger, H. Krahn, and B. Rumpe, "Qualitätsgesicherte Fahrentscheidungsunterstützung für automatisches Fahren auf Schnellstraßen und Autobahnen." In: *Proceedings des 10. Braunschweiger Symposiums "Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel,"* 2009, vol. 10, pp. 341-353.

[Ber10] C. Berger, "Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles." Aachen, Germany: Shaker Verlag, *Aachener Informatik-Berichte*, Software Engineering Band 6, 2010.

[BR12] C. Berger and B. Rumpe, "Engineering Autonomous Driving Software." In: *Experience from the DARPA Urban Challenge*, C. Rouff and M. Hinchey, Eds. London, UK: Springer, 2012, pp. 243-271.

[Car09] J. M. Carter, "Sequence-Based Specification of Embedded Systems." University of Tennessee, Knoxville, 2009.

[Cha05] R. N. Charette, *"Why Software Fails."* In: IEEE Spectrum, Vol. 42, no. 9 (INT), 2005, p. 36.

[CJL+08] D. Chen, R. Johansson, H. Lönn, Y. Papadopoulos, A. Sandberg, F. Törner, and M. Törngren, "Modelling Support for Design of Safety-Critical Automotive Embedded Systems," *SafeComp*, vol. 5219, Series: Computer Safety, Reliability, and Security (SAFECOMP), pp. 72-85, 2008.

[EN01] EN 50128 – Railway applications – Software for railway control and protection, November 2001.

[DO11] DO 178C – Software Considerations in Airborne Systems and Equipment Certification, January 2012.

[GCR94] S. Gerhart, D. Craigen, and T. Ralston, "Experience with formal methods in critical systems." In: *IEEE Software*, vol. 11, no. 1, pp. 21-28, 1994.

[GNA+12] J. M. Carrillo de Gea, J. Nicolás, J. L. Fernández Alemán, A. Toval, C. Ebert, A. Vizcaíno, "Requirements Engineering Tools: Capabilities, Survey and Assessment." In: *Information and Software Technology*, 2012.

[ISO11] ISO-26262 – Road Vehicles – Functional Safety, November 2011.

[KBP01] C. Kaner, J. Bach, and B. Pettichord, "Lessons Learned in Software Testing." Wiley-India, 2001.

[Kon11] B. B. Konka, "A case study on Software Testing Methods and Tools," University of Gothenburg & Chalmers University of Technology, July 2011.

[PP03] S. J. Prowell and J. H. Poore, "Foundations of sequence-based software specification." In: *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 417-429, 2003.

[PBKS07] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software Engineering for Automotive Systems: A Roadmap." In: *Proceedings of the 2007 Future of Software Engineering*, 2007, pp. 55-71.

[PHP11] A. Post, J. Hoenicke, and A. Podelski, "Vacuous real-time requirements," in 2011 IEEE 19th International Requirements Engineering Conference, 2011, pp. 153–162.

[RBK06] B. Rumpe, C. Berger, and H. Krahn, "Software Engineering Methods for Quality Management of Intelligent Automotive Systems." In: *Integrierte Sicherheit und Fahrerassistenzsysteme*, 2006, no. 1960, pp. 473-486.

[Sel07] R. Selby, "Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research (Practitioners)." Wiley-Blackwell, 2007.

[Sie11] S. Siegl, "Specification and verification of the embedded real-time systems in the automotive domain." Göttingen, Germany: Cuvillier Verlag, *Audi Dissertationsreihe*, Band 52, 2011.

[SHG10] S. Siegl, K.-S. Hielscher, and R. German, "Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models." In: *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10)*, 2010, pp. 345-350.

[SHGB11] S. Siegl, K.-S. Hielscher, R. German, and C. Berger, "Formal Specification and Systematic Model-Driven Testing of Embedded Automotive Systems." In: *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE 2011)*, 2011, pp. 1530-1591.

[SL10] S. Siegl and C. Lauer, "Design and Validation of Embedded Real-Time Applications." In: *Proceedings Of Embedded Real Time Software and Systems (ERTSS)*, Toulouse, France, 2010.