Thesis for The Degree of Licentiate of Engineering

Qualitative and Quantitative Assessment of Integration Testing for Model-Based Software in the Automotive Industry

Jan Schroeder

UNIVERSITY OF GOTHENBURG

Division of Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Göteborg, Sweden, 2016

Qualitative and Quantitative Assessment of Integration Testing for Model-Based Software in the Automotive Industry

Jan Schroeder

Technical Report No. 139D
Division of Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Göteborg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Göteborg, Sweden 2016.

Abstract

Background: Integration testing of vehicle software in the automotive industry relies heavily on simulation models. As they replicate actual vehicle functions in the testing process, they increase in size and amount of interconnectivity as rapidly as the actual functions. Valid simulation models are a precondition for valid integration testing. Hence, assessment of the models is of high importance in industry. At the same time, assessment approaches for model-based software validated in industry are scarce.
Objective: The goal of this thesis is to assess current integration testing in the automotive industry and extend the validation of simulation models. Accordingly, we aim to collect insights from the practitioners in the field, including elicitation of actual challenges in the industry, state-of-the-practice processes, and assessments of applicability for validation approaches.
Method: To achieve the objectives we combine quantitative and qualitative research methods including interviews, workshops, surveys, literature reviews, software measurements, correlation analysis, and statistical tests. In five studies attached with this thesis we combine multiple research methods to achieve high validity and to ensure all presented approaches are applicable in industry.
Results: We elicited and categorized challenges from practitioners in practice, particularly in the field of integration testing for automotive software and analyze the current software development process. We present measurement results from complexity and size metrics, as a first assessment of the models. In addition to single measurements, we show how to evaluate software measurement results collected over time and how they can be related to model quality. We show that outlier analysis can help detecting impactful observations in the model development process. Furthermore, we found five approaches for the prediction of software model growth data and elaborate on their strengths and weaknesses, in practice. Next to providing actual approaches, we present practitioners expectations towards maintainability measurements and measurement predictions.
Conclusion: In this work we contribute to the understanding of concrete challenges in industry, we describe current processes, and provide approaches applicable in industry to address elicited challenges. With our work we improve the current assessment of validity of simulation models in integration testing in the automotive industry.

Keywords

Software Engineering, Industrial Case Studies, Empirical Research, Validation, Software Quality, Technical Debt, Software Metrics, Maintainability

Acknowledgment

My biggest thanks go to my supervisor Christian Berger, for his tireless encouragement and guidance. Without his support and feedback I might not have pursued my PhD studies and definitely would not have made it that far.

I also want to thank my second supervisor Thomas Herpel, who constantly shared his technical knowledge and experience. I could always rely on his talent to show me new ways to my work where I could not see them.

Special thanks go to my co-supervisor Miroslaw Staron, for his constructive advice at any time I needed it. His constant endeavor to strengthen my work with scientific rigor is highly appreciated.

I am very grateful to my colleagues at the Software Engineering Division. Particularly, Ivica Crnkovic, Michel Chaudron, Alessia Knauss, Hang Yin, Abdullah-Al Mamun, Antonio Martini, Vard Antinyan, Hugo Sica de Andrade, Federico Giaimo, Darko Đurišić, Hiva Alahyari, Lukas Gren, Salome Maro, Emil Algroth, and Grischa Liebel, as they were always available for constructive discussions and valuable reviews. Further, Ann-Britt Karlsson deserves gratitude, for all the hours she has spent to make this project work and her ever positive attitude.

I also want to thank my colleagues at the Automotive Safety Technologies GmbH, Christoph Funda, Okan Ecin, Ibrahim Alagöz, Herman Abt, Christian Baar, Martin Schalau, Radhakrishna Kothamasu, and Veit Steinhfel, for their willingness to share their technical knowledge, their patient availability to answer all my questions, and for always making me feel welcomed on site.

I owe a special thanks to Bára, for being an indefatigable source of inspiration and motivation for improvement, but also for her patience in stressful times and her constant support.

Finally, I want to express my gratitude to my family. First, to my parents Frank and Felicitas, who supported me with all means available and never stopped believing in me, even in difficult times. Second, to my sister Julia for her encouragement and decision support in times of doubt.

List of Publications

Appended Papers

This thesis is based on the following papers:

[A] J. Schroeder, C. Berger, T. Herpel "Challenges from Integration Testing using Interconnected Hardware-in-the-Loop Test Rigs at an Automotive OEM"
In Proceedings of the First International Workshop on Automotive Software Architecture (WASA), Montréal, QC, Canada, May 4, 2015.

[B] J. Schroeder, C. Berger, T. Herpel "Simulation and Validation of a Safety-Critical Electronic Control Unit for Integration Testing in Connected Hardware-in-the-Loop Environments"
Proceedings of the 3rd International Symposium on Future Active Safety Technology Towards Zero Traffic Accidents (FASTzero), Gothenburg, Sweden, September 9-11, 2015.

[C] J. Schroeder, C. Berger, T. Herpel, M. Staron "Comparing the Applicability of Complexity Measurements for Simulink Models during Integration Testing – An Industrial Case Study"
Proceedings of the 2nd International Workshop on Software Architecture and Metrics (SAM), Florence, Italy, May 16, 2015.

[D] J. Schroeder, C. Berger, M. Staron, T. Herpel, A. Knauss "Unveiling Anomalies and their Impact on Software Quality in Model-Based Automotive Software Revisions with Software Metrics and Domain Experts"
Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA), Saarbrücken, Germany, July 18-22, 2016.

[E] J. Schroeder, C. Berger, A. Knauss, H. Preenja, M. Ali, M. Staron, T. Herpel "Prediction of Software Model Growth in Practice"
In submission to the 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, May, 20-28, 2017.

Other Papers

The following papers are published but not appended to this thesis.

[A] J. Schroeder, D. Holzner, C. Berger, C. J. Hoel, L. Laine, A. Magnusson "Design and Evaluation of a Customizable Multi-Domain Reference Architecture on top of Product Lines of Self-Driving Heavy Vehicles An Industrial Case Study"
Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy, May 16-24, 2015.

[B] T. Herpel, T. Hoiss, J. Schroeder "Enhanced Simulation-Based Verification and Validation of Automotive Electronic Control Units"
Proceedings of the 5th International Conference on Electronics, Communications and Networks (CECNet), Shanghai, China, December 12-15, 2015.

Contents

Chapter 1

Introduction

In modern vehicles tremendous amounts of software fulfill multiple roles and control various vehicle features. Those features are ranging from engine management, to safety features like emergency braking and airbag control, to entertainment and user interaction. Software in vehicles today is embedded in so-called ECUs (electronic control units). ECUs comprise hardware and software parts and modern vehicles contain over 100 of these control units. The amount of software in cars is growing steadily. According to Broy [1], already in 2006, cars ran on up to ten millions lines of code. Today's vehicles contain already more than 100 million lines of code, which is about five times as much as in modern airplanes and twice as much as in current Windows operating systems (cf. [2] and [3]).

In cars, the amount of interconnections between the systems increases together with the size of the software systems. The anonymized graph to the left in Figure 1.1 is extracted from the actual architecture in a modern vehicle at a German premium car manufacturer and shows the amount of existing ECUs as black nodes and the connections between them. Some functionalities require the creation of big interconnected clusters of ECUs working together. Another example for high interconnectivity among functions in vehicles at a Swedish premium vehicle manufacturer is shown on the right side of the figure.

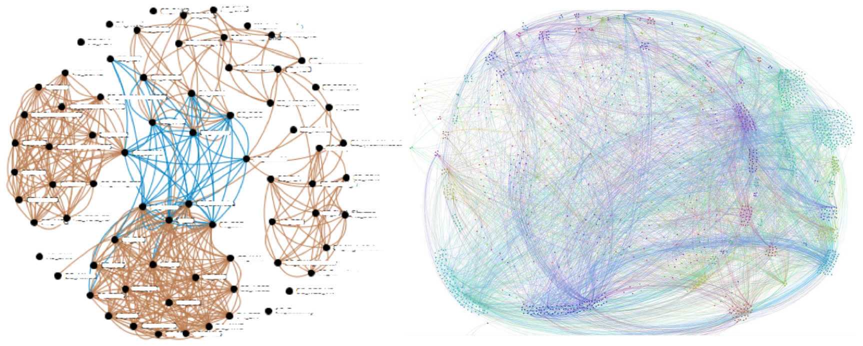


Figure 1.1: Examples for interconnectivity in practice. Signals shared between ECUs at a German automotive OEM are shown to the left and connected functionality at a Swedish automotive OEM to the right (cf. [4]).

A concrete example highlighting how interconnected and complex car software became is a case from 2008 where customers experienced in rare cases the engine control causing the engine to unexpectedly increase the engine rotation speed as soon as the air conditioning was turned on (cf. [5]). The case of Toyotas unintended acceleration as outlined by Koopman [6] highlights importance and difficulty of proper validation of complex embedded systems. A small error in the code caused the vehicles to accelerate unintentionally, causing numerous fatalities. Mössinger [7] lists more challenges particularly apparent for automotive software: reliability, functional safety, real-time behavior, resource constraints, playing key roles in the field. Hence, there is a need for high software quality in automotive software.

This increasing amount of interconnected functions and the need for high quality make extensive testing in the automotive field essential. Tests of ECU software and hardware are to a large extent conducted in so-called in-the-loop environments. In-the-loop refers to the test environment which reads the outputs of the system under test and feeds a respective reaction of the environment back into the system. Figure 1.2 shows the in-the-loop testing in detail.

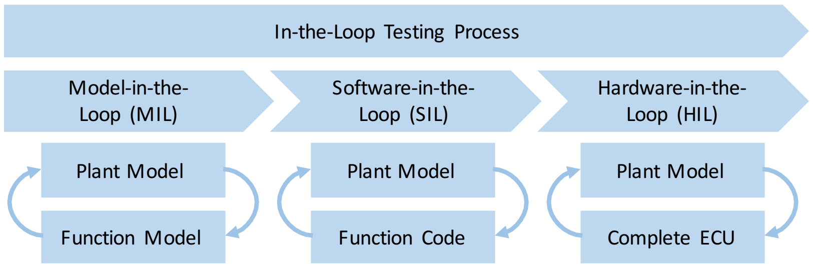


Figure 1.2: In-the-loop processes, adapted from Brückner and Weitl [8].

Testing steps iteratively build on each other. Model-in-the-loop (MIL) and software-in-the-loop (SIL) first address only the software part of the ECU. Hardware-in-the-loop (HIL) tests address the combination of ECU software integrated in the ECU hardware. All phases of this process rely on plant models. Plant models are used to simulate physical behavior. They replicate a real environment for the ECUs to be tested in. As ECUs are also interconnected with each other, an integration test follows as soon as the single ECUs are successfully tested in the above process. This additional test is performed in connected HIL tests. Figure 1.3 shows a typical hardware-in-the-loop environment for integration testing at a German premium car manufacturer.

In these connected HIL tests, vehicle ECUs are connected with each other and placed on frames with real vehicle hardware. A plant model simulates the complete vehicle environment so that all vehicle function can be tested in a realistic environment before it is used in a real vehicle. Next to the plant models, at connected HILs another kind of models are used. Models simulating real ECUs to enable fast replacement in case of failure or unavailability and easy debugging. Hence, next to plan models, simulation models also play a significant role in testing automotive software.

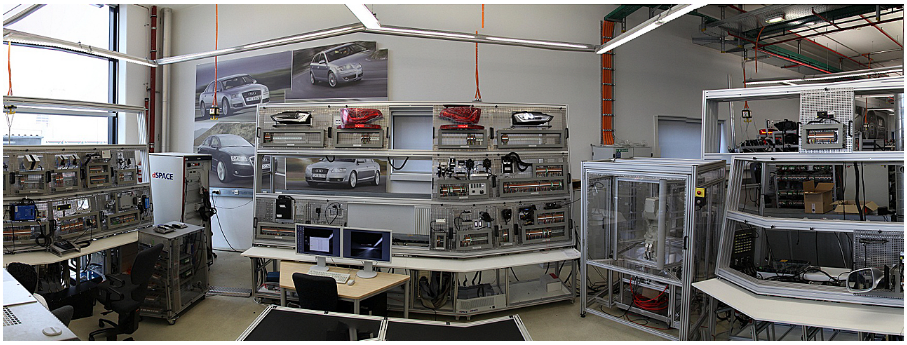


Figure 1.3: Example of a hardware-in-the-loop environment for integration testing.

Accordingly, in integration testing in the automotive industry, models are required to successfully test the real functionality. As they grow in size and complexity together with the real ECUs, ensuring their validity is an important step towards compliance with functional and qualitative requirements. In this thesis we investigate validity of simulation models used for integration testing in practice.

1.1 Background

In this section we first explain software validation and related it to the concepts of software quality and technical debt. As a means to conduct validation, software measurement is introduced, because it is used particularly throughout the thesis. Lastly, the specifics of software engineering in the automotive industry are presented, as the research presented in this thesis is conducted in this field.

1.1.1 Software Validation, Quality, and Technical Debt

According to the IEEE Standard Glossary of Software Engineering Terminology [9], validation is “The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.” Boehm [10] extended this definition by adding “the fitness or worth of a software product for its operational mission”. In the same study, Boehm also phrased the two informal distinctions between

- **Verification -** “Am I building the product right?”
- **Validation -** “Am I building the right product?”

Hence, during software validation the software engineer tries to ensure the functionality of a system as a whole in a realistic environment before it is released to the customer. This is in contrast with verification, where single or few encapsulated features are tested in environments adjusted to the respective functions.

As mentioned in the definition above, validation is usually conducted by comparing software with its requirement specification. Requirements may refer to functionality but also to quality attributes, or constraints (cf. [11]). Next to testing functionality, ensuring software quality is an important part of validation. Validating software quality is not seldom failed to address and misinterpreted in practice. In 2005, Glinz [12] found terminological and conceptual discrepancies among the existing definitions used for software quality. Eckhardt et al. [13] have recently shown that in practice qualities are still misunderstood and that quantification is rare. Hence, software quality requires clear definitions and approaches for quantification.

Multiple guidelines for classification of software quality exist. In their literature review, Giraldo et al. [14] focus on quality in model-driven engineering and answer the question how quality for models is addressed. They found a general lack of studies with clear definitions for quality characteristics. They further show that the guidelines provided with the OMG MDA framework [15] and the ISO 9126 standard [16] as the two most used.

We conclude that definition, interpretation, and application of software quality differ. There is no universal model. We decided to have one consistent quality definition for all work conducted in this thesis and chose the one established by ISO/IEC. In the standard ISO/IEC 25010 [17], which replaced the ISO 9126 standard in 2011, two quality models are defined and many quality terms are explained. The standard defines the following characteristics of product quality, used throughout the thesis:

- Functional suitability
- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

The standard does not come without critique. In 2005 Al-Kilidar et al. [18] particularly mention problems with the ISO 9126 standard regarding ambiguities, incompleteness, overlapping definitions, and its applicability in practice. Still, it proposes a uniform taxonomy to describe and categorize software quality. Additionally, the standard also provides a framework for quality measurement in ISO/IEC 25020.

In their literature survey, Mohagheghi et al. [19] present an alternative list of classes for quality, specifically for model-based software. They list the following six classes:

- Correctness
- Completeness
- Consistency
- Comprehensibility
- Confinement
- Changeability

The classes mostly overlap with the ISO 25010 standard but addresses challenges in modeling more specifically.

Table 1.1: Technical debt categories according to three recent literature studies.

2013: Tom et al. [24]	2015: Li et al. [21]	2016: Alves et al. [25]
Knowledge Distribution and Documentation Debt	Documentation Debt	Documentation Debt
Defect	Defect	Defect
Infrastructure Debt	Infrastructure Debt	Infrastructure Debt
Requirements Debt	Requirements Debt	Requirements Debt
Test Debt	Test Debt	Test Debt
Code Debt	**Code Debt**	**Code Debt**
Architecture and Design Debt	**Architectural Debt**	**Architecture Debt**
	Design Debt	**Design Debt**
	Build Debt	Build Debt
	Versioning Debt	Versioning Debt
		People Debt
		Test Automation Debt
		Process Debt
		Service Debt
		Usability Debt

In this thesis, maintainability has a special focus. In Schroeder et al. [20], we found that maintainability is important for practitioners and underrepresented by current validation approaches in the domain of integration testing. Additionally, maintainability has another interesting aspect. Li et al. [21] found that in current research, technical debt is strongly related to maintainability. By definition of Cunningham [22], technical debt denotes not quite right code that is accepted for the sake of reaching a deadline, for example. This causes an interest in terms of working effort, every time it has to be maintained or evolved. Finally, a principal might have to be paid as the developers repair the flawed code.

Technical debt is a research topic which finds high resonance with industry practitioners. In our study Schroeder et al. [23], we observed an overlap in the practitioners understanding of maintainability and technical debt. Therefore, this concept is carried on in the course of this thesis.

The concept of technical debt can be transferred to other software artifacts apart from plain code, for example debt in software architecture or tests. Different types of technical debt have been identified throughout the last twenty years. Table 1.1 shows what different kinds of debt have been identified by the three most recent comprehensive literature studies on the topic.

In the course of this thesis, we understand technical debt purely in regard to the piece of software under development, the simulation models. This includes architecture debt, design debt, and code debt. Debt related to processes, infrastructure, requirements, or people is not considered. Li et al. [21]

also list different activities involved in technical debt management. They are: identification, measurement, prioritization, monitoring, repayment, representation/documentation, communication, and prevention. In this thesis, we focus on the first activities, including identification, measurement, prioritization, and monitoring in the context of automotive software engineering.

1.1.2 Software Measurement

In the previous section, we introduced validation as testing a software product against its specification. Manual testing becomes infeasible when amounts of functionality, specifications, and complexity increase. Furthermore, manual testing becomes insufficient if specifications are not complete, accessible, or understandable. Accordingly, an alternative validation approach like software measurement can improve the validation process and complement testing. Additionally, software measurement provides a means of quantification, to make concepts like software quality and technical debt comparable.

Fenton and Bieman [26] describe measurement as the evaluation of software artifacts and the act of making attributes of software quantifiable by assigning numbers or symbols to them. This quantification happens according to a predefined set of rules, like the scale the measurement can be used in. The five scale types currently used are *nominal*, *ordinal*, *interval*, *ratio*, and *absolute* (cf. [27]). Scales describe the set of values they contain and the relations possible within the set. That means, not all operations are meaningful in on certain scale. For example, it is not meaningful to perform additions on nominal data or calculate the mean on interval data. Hence, scales contribute to correct handling of the quantitative data. Quantifying qualitative attributes enables the comparison of those attributes and the underlying artifacts. As attributes are comparable, statistical evidence can be collected on the performance of artifacts and their attributes, respectively.

The intention behind performing software measurements is to explain previously unclear concepts, increase their understandability, and controllability (cf. [26]). For qualities like performance, often simple evaluations of measurements are sufficient, for example, by measuring execution time. For software quality characteristics like usability, reliability, maintainability and related technical debt might not be derived by measuring single software attributes but by combining multiple attributes mathematically. A metric or measure for the size of a software artifact can be expressed by using "number of lines of code" but it can also be expressed by dividing the "number of lines of code" by the "number of components". Hence, plain measurements are combined or adjusted to improve the model of the behavior in the real world. To ensure a measure's applicability, evidence is needed on how strong a collected measurement result is correlated with the software attribute intended to be assessed.

Four specific code and architecture metrics used in this thesis are assessing size and complexity. They are listed in the following and detailed descriptions can be found in Section 5.2.1.

- **Size: Lines of Code**
  This metric is created by counting lines of code of the source files generated by Simulink. The code in the source files resembles an XML-like structure and contains all information also contained in the model.

- **Size: Block Count**
  For the block count metric we use a function provided by Matlab, called *sldiagnostics*. The function counts all blocks contained in a Simulink model, including all blocks nested in others (cf. [28]).

- **Complexity: Structural Complexity**
  This metric is a combination of the count of the amount of blocks each block is connected to (fanout) and the total amount of blocks in the model.

- **Complexity: Data Complexity**
  This metric uses the same attributes as the Structural Complexity metric but additionally considers the number of inputs and outputs of each block.

To improve the performance of the metrics used in this thesis, we adjust them to fit the field. Each metric used in this thesis is assessed for its performance in practice, using stakeholder interviews, workshops, and surveys. Not only the performance is of interest, but also the stakeholder needs are elicited to ensure that metrics applied in practice are needed and useful.

Software metrics can target different aspects of software like development, testing, or management and can be applied in all steps of the software development process to monitor products or their artifacts. In this thesis we focus on code and architecture assessment. Other activities within the software development process are not investigated.

1.1.3 Integration Testing and Model-based Software

Current software development processes in the automotive industry follow the V-model, as shown in Figure 1.4. As the process depends on hardware development, a development process with consecutive steps like the V-model proves useful and is still widely used in this domain. Nevertheless, single process steps, like independent software function development might be conducted in a more iterative and incremental way following agile methodologies. Testing activities are depicted on the right side of the figure, with the integration testing phase highlighted with dashed lines.

Integration testing in the automotive domain is performed using connected hardware-in-the-loop (HIL) platforms. At connected HIL platforms the ECUs part of the vehicle are coupled while the physical environment is simulated using real-time simulation models. In this environment interconnected ECU functions can be tested before the ECUs are brought into the actual vehicle. To ensure a realistic testing environment, the coupling is performed using the actual vehicle buses and real vehicle parts are included as feasible.

During integration testing, simulation models play a major role. They have two responsibilities: They serve as plant models, simulating the physical behavior of the vehicle and as ECU models, simulating actual ECU behavior to replace missing or faulty ECUs. Software models are an abstraction (cf. [29]), either of plain code or even of other models. Thereby, information that is currently not needed or not available are hidden to achieve simplification. Models for simulation is different from static models mostly used for visualization and

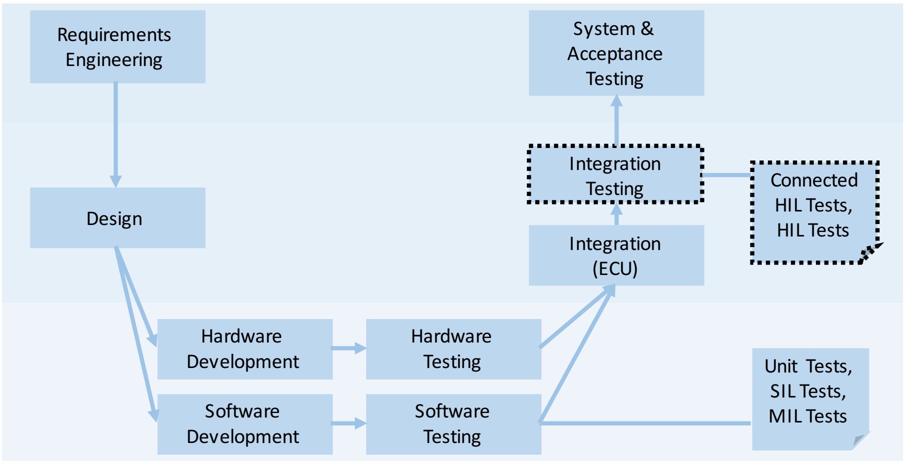


Figure 1.4: General V-Model as it is used in automotive software development

design. Static models usually do not require a time dimension, while time is the key aspect of simulation models (cf. Zeigler [30]). Either on a continuous or discrete time scale, the output of simulation models is calculated using a function of the input to the models.

At the case company, all simulation models are created in Simulink. They are constructed from blocks and connectors, using a layered architecture as shown in Figure 1.5. In the figure, the simulation model to the left contains two blocks, one being an ECU model and one a plant model. Within those models, functions are again visualized as blocks and are implemented in different ways. They can be combinations of logical operators, state machines, or written in programming or scripting languages. All three implementations have in common that they operate on the input signals and generate an output signal.

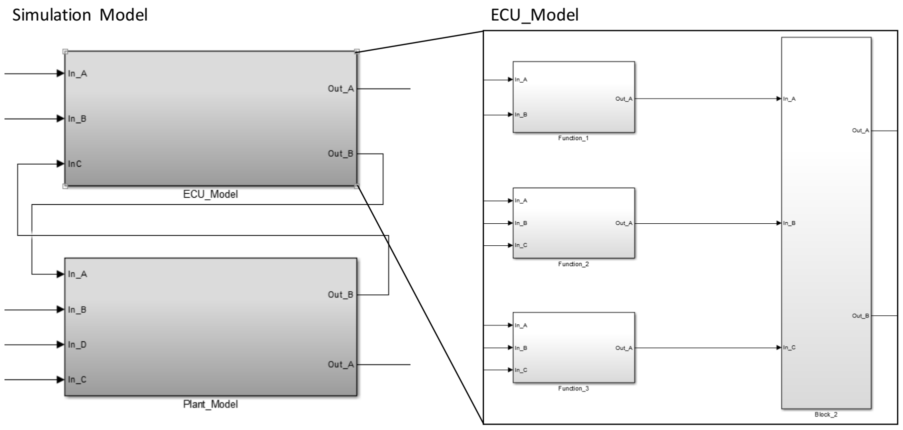


Figure 1.5: Example of a model created in Simulink.

The models are designed to be executable. At compile time they are translated into executable C code, which is mainly hidden to the developer.

This way of realizing model-driven development should be separated from other realizations like the usage of models as a means of visualization. This distinction is also mentioned in the model-driven development spectrum, presented in Staron [31]. Where one end of the spectrum covers the models solely used to visualize the code, Simulink and the related model-driven development is located on the other side of the spectrum where models are used as the main development artifact. This distinction is important, as existing assessment approaches created for plain code might not be directly applicable to generated code from simulation models. Particularly, code maintainability and usability metrics are not meaningful as engineers work directly on the models. Literature on model-based software has to be assessed using the same criteria, to avoid misconceptions.

1.2 Motivation and Problem Domain

We highlighted earlier, that automotive software is complex, constantly growing, and that contained functionalities are highly interconnected. Accordingly, approaches for identification, monitoring, removal, and prevention of complexity are needed. At the same time, there is a lack of research for the assessment of model-based software in the automotive domain. Hence, there is a need for assessment approaches applicable in this domain.

Integration testing in the domain depends on valid simulation models. As size and complexity of simulation models grow along with the real functions, manual validation is infeasible. Furthermore, validating these models solely by tests against written specifications might suffer from a lack of completeness, accessibility, or understandability of these specifications. Using indirect assessment of model properties with measurements provides additional means of model validity assessment and thereby contributes to increasing software quality in the automotive domain.

Software measurement is used widely in software engineering to assess software projects and products [26]. Evidence on the applicability of metrics in the domain of integration testing in automotive software engineering is still scarce. Furthermore, even though simulation models are widely used in industry and an important part of the integration testing process, there is only few literature on actual metrics for model-based software, particularly Simulink models.

1.3 Research Goal and Questions

In this thesis we are addressing current challenges in the field of integration testing in the automotive industry and investigate how simulation models can be assessed for validity. Quantitative assessment using software metrics and statistics is combined with qualitative approaches like interviews and surveys to ensure applicable improvements of current model validation. From this main goal, we derive sub-goals expressed as research questions which are going to be addressed in this thesis.

RQ1 What are challenges and processes currently existing in integration testing in the automotive domain?

RQ2 How can software quality assessment be customized to simulation models in the automotive domain?

- **RQ2.1** How can metrics of model complexity and size be applied to assess maintainability of simulation models in the domain?
- **RQ2.2** How can existing measurements be extended by outlier assessment to provide insights on correlations with quality characteristics?
- **RQ2.3** How can model growth be predicted reliably in the automotive domain?

1.4 Methodology

The research methods used in this study are shown in Figure 1.6. In the figure, the methods are also related to the previously presented goals and research questions. It can be seen that a strong focus is put on combining qualitative

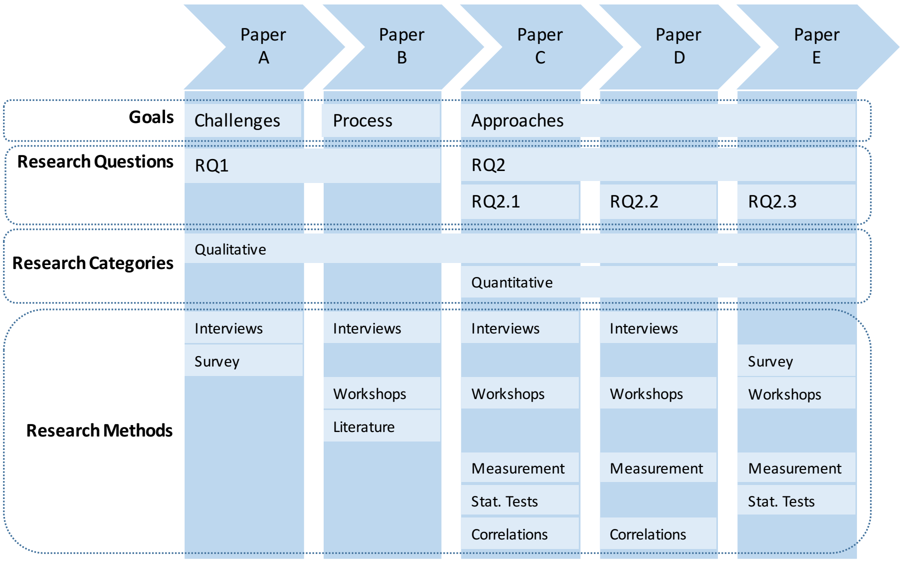


Figure 1.6: Overview of the research methods used in this thesis.

and quantitative research methods. Malhotra [32] describe quantitative studies as based on mathematical and statistical methods, while qualitative focuses on human aspects. In the appended papers, insights from literature and quantitative results from software measurements are always compared with practitioners' assessments, either in interviews, workshops, or surveys. By combining multiple research methods and sources of information we achieve method and data triangulation in all our studies. All studies contain more than one method of data collection. Except for study A, concerning the prominent challenges in the field, we always rely on more than one source of information. While triangulation increased research validity, the continuous involvement of practitioners ensured a benefit for industry, as well.

In the studies presented in this thesis we perform empirical research. Following Malhotra's [32] definition, empirical research analyzes real-life data to compare theories and observations. Different research methods are used within empirical research. Using five examples, Stol and Fitzgerald [33] show that the levels of detail of the research method definitions differ. All studies in this thesis project can be described as case studies, as they all address the case of integration testing of automotive software engineering. According to Malhotra [32] case studies are solely qualitative and commonly use observations, interviews, and group discussion for data collection. Other authors are less restrictive in their definitions. Runeson et al. [34], for example, argue that data collection in case studies should be based on triangulation of multiple data sources, including quantitative data as well. Hence, clear definitions of the research methods used is important.

We refer to the literature, to clearly define the research methods used in this thesis. The findings are summarized in Figure 1.7 and explained in the following.

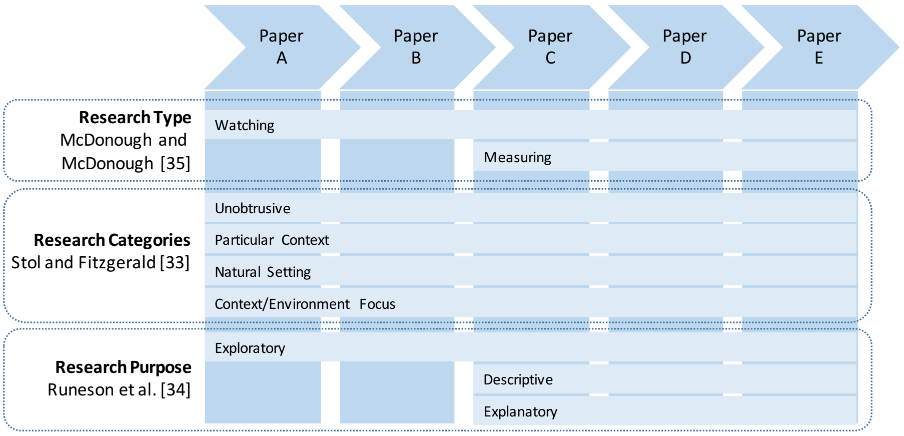


Figure 1.7: Research Categories from literature.

McDonough and McDonough [35] differentiate four types of research: controlling, asking/doing, watching, and measuring. In their categorization, case studies are placed in the "watching" category, exhibiting low amount of intervention and low selectivity. Study A and B fit this description. Studies C, D, and E also employ a degree of "measuring", which makes them more selective in nature and labels them as systematic observation rather than as case study, according to McDonough and McDonough.

Stol and Fitzgerald [33] collected multiple scales to categorize research methods:

- from obtrusive to unobtrusive
- from universal context to particular context
- natural setting, contrived setting, setting independent, and not empirical
- actor focus, behavior focus, or context focus

While conducting the studies presented in this thesis, we always tried to be unobtrusive. Following the case study definition of Zelkowitz et al. [36], we always avoided researcher intervention with the case. The context of our studies is always integration testing in the automotive domain. Respectively, our setting is natural and not contrived. The research focus is strongest on the context dimension. That means that the studies achieve a high amount of realism at the cost of control as it would be achieved in experiments. At the same time, the general high incorporation of practitioners and the use of surveys increase the actor focus and generalizability (cf. [33]).

Runeson et al. [34] distinguish different purposes of research:

- Exploratory
- Descriptive
- Explanatory
- Improving

All five studies are to some extent exploratory, as they explore phenomena and seek to create new research questions (cf. [37]). Particularly the first two studies fit this category, where we explore the field of integration testing in automotive software engineering by investigating challenges and processes. Although, casual relationships cannot be proven in case studies (cf. [38]), the later three studies have to some extent a descriptive and explanatory purpose. In those studies, we start with a theory of existing association of size/complexity measure with complexity/maintainability attributes of the software. Thereby, we try to describe existing phenomena and to find causal relationships between model attributes and quality attributes. The notion of study purpose can be linked to the distinction between inductive and deductive empirical research (cf. [34]). Inductive research creates theory from observations, while deductive research confirms a theory using observations.

As mentioned above, all studies in this thesis are conducted as case studies. Hence, we followed the concrete guideline on planning and conducting case studies by Runeson and Höst [38]:

- **Objective**
  Every study performed in this thesis has a clear goal.
- **Definition of the Case**
  We ensured a clear definition of what is studied in all attached papers.
- **Theoretical Background**
  Part of every study is a rigorous investigation of similar studies and necessary background.
- **Research Questions**
  We always defined explicit and relevant research questions to be answered in the course of our studies.
- **Methodology**
  Data collection and analysis in our studies is always clearly described.
- **Selection Strategy**
  The selecting of case and subjects was systematic and is clearly outline in each study.

Additionally, Runeson and Höst provide recommendation on the style for reporting case studies. We always followed their guidelines as close as possible, to provide clear and concise descriptions to enable replicability.

1.4.1 Interviews and Workshops

Interviews are an important means for data collection in case studies (cf. [38]). In interviews data is collected in a dialog between the researcher and one or more interviewees. Usually, the dialog is guided by the researcher using interview questions addressing previously defined research questions. Runeson and Höst [38] distinguish open and closed interview questions, formulated in a unstructured, a semi-structured, and a fully-structured manner. Unstructured interviews are mostly exploratory, containing open questions, and focus on qualitative data. Structured interviews on the other hand are more descriptive and explanatory, contain closed questions, and focus on quantitative data (cf. [38] and [39]). Semi-structured interviews are less restrictive and can contain attributes of both other categories. Hence, even though interviews are usually categorized as qualitative research method (cf. [32] and [39]), their usefulness to elicit quantitative data is mentioned, as well (cf. [38] and [39]). In this thesis, we define workshops as group-interviews, meaning an interview with at least two interviewees, guided by the researcher and interview questions.

We used interviews and workshops for different purposes. While interviews are a main source for data collection from practitioners, workshops additionally had a confirmatory purpose and served as additional source of input. In workshops we presented ideas which might have been proposed by single individuals in an interview to validate it, trigger discussions, and develop ideas further.

As shown in Figure 1.6, all of the five studies in this theses contain either interviews, workshops, or both. Hence, interviews are a major means of data collection in this thesis. We used semi-structured interviews and extracted both, qualitative and quantitative information. Open and closed questions were used and the interviewer was free to adjust and create questions based on the answers received (cf. [40]). We followed the guidelines from Seaman [39] on how to structure interview questions and transcripts. The data collected in transcripts was evaluated using qualitative methods like coding (cf. [39]) and quantitative methods like grouping and counting the numbers of answers.

Interviews and workshops have the advantage of close interaction between researcher and interviewee. Semi-structured interviews enabled flexibility to extend on unclear topics and to verify collected insights. On the other hand, interviews had the disadvantage of being limited in the amount of opinions collected, compared to studies like surveys where more subjects can be reached with less effort. Additionally, interviewees might be biased. For example, they might answer in their favor in case questions address deficiencies. A save environment was created in the beginning of the interviews to avoid this bias. Interviews were always treated anonymously.

1.4.2 Surveys

Surveys or questionnaires are often considered as a research method by itself, separate from case studies (cf. [33] and [32]). According to Stol and Fitzgerald [33], they are more universal than case studies, as they address a broader spectrum of subjects. They are also more focused on the subjects' opinions than on the context. Surveys are usually considered as a quantitative method (cf. [32]) but using open questions, they can elicit qualitative data as well (cf. [39]). Singer et al. [40] mention surveys as the most commonly used field study technique, due to the low amount of time and resources they require. They further consider interviews and surveys to be related, as the goal of both is the collection of information on processes, products, or personal knowledge.

The two studies in the Chapters 2 and 6 use surveys as data collection technique. We chose surveys if large numbers of subjects had to be addressed, if questions were mostly closed, and intended to be evaluated quantitatively. Surveys have the advantage of being time and cost effective while being asynchronous in a way that the researcher doesn't have to be present when they are answered. However, surveys have a risk of achieving only low response rates which weakens the strengths of the achieved evidence. They also have the disadvantage of possible misunderstandings when questions are ambiguous. We could mitigate the disadvantages by not only relying on surveys, but complementing them with interviews and workshops. As shown in Figure 1.6, we always used at least one additional research method together with surveys.

1.4.3 Quantitative Analysis

In literature quantitative research methods address hypotheses and collect generalizable results (cf. [32]). Quantitative analysis in this thesis refers to data collection and analysis approaches relying on numerical data and mathematical functions. Next to statistical tests and correlation analysis we also mention software measurements as quantitative approach. We thereby follow Runeson and Höst [38], who include the usage of correlation analysis and hypothesis testing to collected data in their definition of quantitative data analysis.

Software measurements are solely used for data collection and have been described previously in Section 1.1.2. Statistical tests and correlation analysis are used to analyze collected data and to confirm theories. In the Chapters 4 and 5 quantitative analysis is used to collect evidence to answer the research questions. In Chapter 6 we additionally provide a concrete hypothesis to be tested. The strong emphasis on quantitative research in the Chapters 4, 5, and 6 also requires rigorous methods. The area of software engineering experimentation provides required guidelines, for example provided by Basili et al. [41] or Wohlin et al. [42]. As it is near to impossible to control all variables in a real world case, we never perform a controlled experiment. Still, the guidelines enabled a structured research process. From their extensive descriptions we adopted the following guidelines for our quantitative analyses:

[A] **Research Questions**
Establishing clear goals and research questions before planning the data collection avoids researcher bias during execution. As applicable, we additionally defined independent/dependent variables and used hypotheses describing the research questions.

[B] **Empirical data collection**
We collected data from a large amount of data sources to increase validity. Additionally, to avoid bias, we used randomization of collected data sets, for example, when splitting data into groups for comparison. Automated data collection increased efficiency and replicability.

[C] **Empirical Analysis**
Careful selection of appropriate analysis methods ensured validity of the analysis. The analysis has to fit the collected data and the established research questions. In our studies, this included decisions between parametric or non-parametric analysis depending on the data. Furthermore, it included decisions for appropriate statistical tests and correlation analysis, fitting the data and its behavior.

In general, quantitative data collection and analysis has the advantage of providing generalizable and unbias results (cf. [32]). Additionally, results are replicable as they originate from numerical and mathematical functions. Still, quantitative results usually require further interpretation. An outcome of a statistical test is often limited to showing a difference in the performance of two or more compared approaches. The combination of quantitative research with qualitative research, as presented in this thesis, enables interpretation of quantitative results to receive further insights into the results and the applicability in practice.

1.5 Study Summaries

In this section the content of all included studies is summarized. We describe the goal, methodology, results, and contributions of each of the studies, to provide an overview over the work performed in the course of this thesis project.

1.5.1 Chapter 2: Challenges from Integration Testing using Interconnected Hardware-in-the-Loop Test Rigs at an Automotive OEM

In the first study, we elicited prominent challenges and their root-causes, in the real-world environment of integration testing at an automotive OEM. The elicitation is done using semi-structured interviews with 13 practitioners in the field. The interviews were based on a structured questionnaire but open for discussions. By coding the transcripts, challenges were extracted and grouped into eight different categories. Additionally, root-causes like time and work force constraint and suggested measures for improvement were obtained from the codes. Furthermore, technical debt items were extracted from the challenges. Next to the typical code, test, and architectural debt items, also requirements and documentation debt were mentioned by the practitioners. Accordingly, this study contributes to the thesis by outlining the field of integration testing. Major challenges are identified and compared to those found in similar studies.

1.5.2 Chapter 3: Simulation and Validation of a Safety-Critical Electronic Control Unit for Integration Testing in Connected Hardware-in-the-Loop Environments

The goal of the second study was the assessment of the current modeling and simulation process at the case company. We analyzed how existing knowledge from practice and literature can be combined to improve the current development process. Therefore, we performed interviews with seven stakeholders involved in the current development process to extract the current state-of-the-practice. Complementary literature studies provided the current state-of-the-art. Using knowledge from literature, we provide recommendations on how to improve the current development process. A systematic development process for the creation of simulation models for integration testing is proposed. The process is split in three steps performed in iterative cycles; requirements elicitation, development, and verification and validation. Detailed information supported with current literature are provided for each step. Hence, this study contributes with insights into the current development process. This provides a clear understanding of the activities involved in developing simulation models for integration testing. Additionally, we propose its extension in form of recommendations on improvements for each step.

1.5.3 Chapter 4: Comparing the Applicability of Complexity Measurements for Simulink Models during Integration Testing – An Industrial Case Study

In order to reveal specific quality deficiencies among the simulation models, we introduced software measurements in the third study. The research objective was to assess the applicability of size and complexity measurements to determine software maintainability and complexity in practice. We applied two size and two complexity metrics to 65 simulation models. These quantitative measurement results are then enriched with qualitative data elicited in interviews and a workshop with practitioners. Findings show that the metrics are not strongly correlated and that they assess different properties. The assessment of the qualitative data revealed a preference among the practitioners towards size metrics for the assessment of model complexity and maintainability. Hence, this study contributes with the evidence that the used size metrics are applicable to assess maintainability in the field. Furthermore, from the interview, we elicited requirements practitioners have towards maintainability metrics.

1.5.4 Chapter 5: Unveiling Anomalies and their Impact on Software Quality in Model-Based Automotive Software Revisions with Software Metrics and Domain Experts

In this study we aimed to extend the previous model validation using software metrics by extending it with measurements over four years of past software revision data. The goal was an efficient approach to highlight revisions with high impact on model quality by investigating outlying observations among the measurements. Four metrics were applied to the past revisions of 71 simulation models. Two different approaches revealed outliers among the measurement data. These qualitative information were then used to distinguish between meaningful observations (anomalies) and random outliers. This was done by interviewing eight engineers, each responsible for a subset of the 71 models. The findings were validated in consecutive workshops. Results show that outliers within the measurements actually correlate with the impact assessment of the practitioners for four quality characteristics. We showed that the quantitative approach using outlier detection is applicable in practice to reveal revisions with high impact. The approach can be used to assess the present development of the simulation models and highlight problematic revisions.

1.5.5 Chapter 6: Prediction of Software Model Growth in Practice

The goal of the most recent study was to predict previously received measurement data to assess their future development. Thereby, we aimed for predicting which of the simulation models grow beyond acceptable thresholds and evaluated the applicability of five prediction approaches in practice. Hence, we measured model size for 4,668 revisions of 48 simulation models and ran five different prediction approaches on the resulting time series. To make an informed decision on the applicability in practice, we collected practitioners expectations towards predictions in a survey. The prediction results are then assessed for how well they fulfill those expectations. We found that accuracy of long term predictions ($\geq$ 30 days) is most important to the stakeholders in the field. We show that there are significant differences between five approaches regarding prediction accuracy. Furthermore, we found that statistical approaches perform equally well as machine learning approaches while requiring less run time. Regarding machine learning approaches, we show that simple artificial neural networks perform best on the tested data. We contribute with applicable prediction approaches in practice and an assessment of their strengths and weaknesses.

1.6 Discussion

This section will present a synthesis of the results and contributions from the individual papers and discuss how they connect to each other.

1.6.1 Contributions

In this thesis we have identified challenges and assessment approaches for quality characteristics of model-based software in practice. In detail, we provided the following contributions to the software engineering body of knowledge.

C1: Assessment of the field: challenges, root causes, development process
In Chapter 2 and 3 we contribute with insights into the real world case of integration testing within the automotive industry. First, in Chapter 2, we identify eight different categories of challenges in the field. Thereby we highlight that code, process, and communication challenges as recurring. Additionally to these challenges, in Chapter 3, we contribute with a description of a typical development process and recommendations for its improvements. These challenges and the assessment of existing processes contribute to establishing the state-of-the-practice of integration testing of automotive software engineering. It increases the understanding of the domain before we conducted more specific investigations and it provides other researchers with insights collected in a real environment.

C2: Practitioners expectations towards maintainability measurements in practice
Using the interviews presented in Chapter 4 we provided insights on the practitioners understanding of maintainability and its assessment. We showed that the interviewed practitioners from the field of integration testing in the automotive industry associate three major factors with maintainability of simulation models: communication between blocks, block size, and the structure of the model. As these findings are based on a single interview study, we validated them in the later studies described in the next contribution C3.

C3: Comparison of applicability of size and complexity measurements in the field
Based on the three attributes received in contribution C2, in Chapter 4, we showed how results from metrics based on attributes actually correlate with practitioners understanding of maintainability and model quality in general. Findings showed that correlations are only significant for one size metric. Concerning maintainability assessment, we revealed that the attributes mentioned by practitioners in interviews do not overlap with ratings received from measuring these attributes. We concluded that size measurements express maintainability best in the studied domain integration testing in the automotive industry.

C4: How to use outlier detection to highlight quality deficiencies in practice
In Chapter 5 we propose an automated approach to analyze anomalous behavior among time series of measurement data. We show that it is applicable in the domain of integration testing in automotive software engineering. Thereby, we establish an approach for automatic evaluation of measurement data based on historic values. Accordingly, we highlighted anomalous development of model quality which correlates with the size and complexity metrics used.

C5: Using outliers to detect correlations between measurements and model quality
Next to the applicability of the anomaly detection, in Chapter 5, we also showed which qualities correlate with detected anomalies. We contributed by showing that anomalous behavior in size and complexity measurements correlates with functionality, usability, efficiency, and maintainability. Hence, we provide an approach for automated assessment of these qualities, in practice.

C6: Comparison of prediction approaches for time series in practice
In our most recent study presented in Chapter 6, we show how to predict software model growth, which is one of the most suitable metrics for maintainability estimation in the domain, according to the interviewed practitioners. We found statistical significant differences between the of accuracy of the different prediction approaches. Thereby, we show that support vector regression performs worse than other machine learning and statistical approaches.

C7: Elicitation of practitioners expectations towards predictions
In the course of Chapter 6 we also show that practitioners in the field of integration testing in automotive software development expect predictions to be particular accurate for up to one month ahead. High short term predictions are not as important, as is maintenance and run time of the prediction.

1.6.2 Threats to Validity

Regarding the threats to the validity of the research presented in this thesis, we followed the largely overlapping guidelines of Runeson and Höst [38], Feldt and Magazinius [43], and Wohlin et al. [42]. The following threat assessment is based on the classifications presented in those studies.

- **Conclusion and Internal Validity**
  According to Feldt and Magazinius [43], conclusion validity concerns the significance of the conclusions drawn and internal validity concerns ensuring that the treatment actually caused the outcome and not possible confounding factors. Compared to experiments, in industrial case studies confounding factors can always influence the results as we cannot control all variables. Still, we ensured the statistical significance in all analyzes we conducted. Additionally, we mitigated threats to internal validity by not relying on one single data set but applied data and method triangulation and do not rely on single data sets or methods.

- **Construct Validity**
  Construct validity refers to the connection between study goal and the observations (cf. [43]). In the course of this thesis construct validity ensures that what we measure and assess actually fit our intentions. For example, does measuring model maintainability relate to model validity? We addressed this threat by constant involvement of industry practitioners when planning and conducting the studies as well as for evaluating the results.

- **External Validity**
  Generalizability is one of the major threats of all five studies part of this thesis. As mentioned in Section 1.4, the studies are always based on the single case of integration testing in the automotive domain. Hence, we cannot claim that findings hold for other domains, as well. We mitigated this threat by using large data sets and varieties of approaches. Additionally, within the limits of non-disclosure agreements, we describe our case, contained models, tools, and processes as detailed as possible. Thereby, we increase generalizability to domains with similar environments.

- **Credibility and Dependability**
  Credibility and Dependability are concerned with the correctness, consistency and repeatability of the results. Result credibility in this thesis is often threatened by the comparably small sample sizes available for the studies. Next to generalizability this can be seen a major threat of the studies conducted in this thesis. We mitigated this threat by involving participants with different roles and ensure correctness by applying rigorous methods based on current literature.

- **Confirmability**
  Ensuring that received findings are determined only by the respondents and not by the researcher (cf. [43]) is particularly important in the two qualitative studies in the Chapters 2 and 3. To ensure confirmability, we followed established guidelines on conducting interviews and extracting knowledge. Additionally, we always involved more than one researcher in the planning, decision, and evaluation processes.

1.7 Conclusions and Future Research

The goal of this thesis was to assess current challenges and development processes, and provide applicable assessment approaches for validity evaluation of simulation models for integration testing in automotive software engineering. Hence, in this thesis we covered two major topics. First, we outlined the field by analyzing challenges and state-of-the-practice processes. Second, we investigated the approaches software measurement, outlier analysis, and predictions for applicability in the field and their ability to support the validation of simulation models.

To address the first topic, we performed extensive literature research and conducted qualitative research methods like interviews, workshops, and surveys. We concluded that there are two prominent challenges in integration testing in the automotive industry. Challenges related to code and process and communication were found to be recurring. Additionally, we found that technical debt items are also challenging the integration testing process. This includes code, test, and architectural debt, as well as requirements and documentation debt. During our investigations we also found that the process of creating simulation models requires special attention to validation and verification. Hence, we showed that the field of integration testing in the automotive industry is facing multiple challenges. We decided to focus on challenges related to model quality. Particularly, maintainability and the related model size and complexity.

During our investigations of measurements, outlier analysis, and predictions for model validation we found that all used approaches are applicable in the field of integration testing in the automotive industry. We showed that size and complexity measurements results are not strongly correlated and therefore assess different aspects of the models. We showed that the engineers' understandings of what makes a model complex differ from their complexity assessment using anonymized models and measurements. Interviewed engineers significantly preferred a size metric counting blocks in Simulink models for assessment of complexity over more sophisticated metrics which combine structural and data attributes of the models.

By applying outlier analysis to measurement values retrieved over time we demonstrate an automated approach for model assessment. We found that outliers in the measurements are related to software quality characteristics and showed that our approach can differentiate revisions with low and high impact on the quality characteristics.

Lastly, we predicted model growth using the previously mentioned measurements collected over time. As a first step, we elicited requirements of practitioners towards predictions and found that prediction accuracy over long time periods are prioritized over short term accuracy, run time, and maintenance effort. We then showed that five different prediction approaches fulfill the requirements obtained from the practitioners. Regarding prediction accuracy, we could unveil significant statistical differences among the approaches. The statistical approach ARIMA showed best accuracy results and short run times, whereas the machine learning approach support vector regression required longer run times while achieving worse accuracy results. Furthermore, we showed that among the machine learning approaches simple feed-forward networks provides highest accuracy given our data.

Hence, we provide a set of quantitative measurement and analysis approaches applicable in the field of integration testing in the automotive industry.

The work conducted in this thesis provides multiple opportunities for future work. For example, the three presented approaches can be combined in an integrated assessment framework for simulation models. Software measurements and assessment could be integrated with the existing development process. The framework should then enable the incorporation of additional software metrics addressing further quality characteristics. The framework should also provide a means for visualization. It shall be possible to present the findings received in the studies in an efficient way to the practitioners, to make decisions on which models require further attention or refactoring.

Furthermore, generalizability, one of the biggest threats to this work could be addressed in future research. The applicability of the work in this thesis was validated using one single case. Although, we mitigated the threat regarding generalizability in our studies, part of the future work will include the validation of the used approaches in other domains using similar processes and models. Hence, another requirement towards a possible framework is the portability.

Lastly, further work will include a way to address deficiencies our approaches find in the models. We will conduct further literature work to discover ways to correct and prevent problems in the models, while continuing the close interactions with industry practitioners to ensure applicability.

Chapter 2

Paper A

Challenges from Integration Testing using Interconnected Hardware-in-the-Loop Test Rigs at an Automotive OEM

J. Schroeder, C. Berger, T. Herpel

In Proceedings of the First International Workshop on Automotive Software Architecture (WASA), Montréal, QC, Canada, May 4, 2015.

Abstract

Developing automotive functions involves complex software to a growing extent while still following a consecutive waterfall-like development process: Integrating and testing software with other software and with hardware components is conducted towards the final phases during the development. For example, ambiguous requirements or unclear semantics in system interfaces show up very late and mostly not before integration testing. In this article, we are reporting about results from conducting interviews with integration and test engineers at a large automotive OEM about today's most resource-intense challenges when dealing with software integration testing tasks at interconnected hardware-in-the-loop test rig environments. Challenges in processes, communication, and implementation were mentioned most often as emerging topics to be tackled over and over again during this phase in a vehicle series development project.

2.1 Introduction

Most of the software in modern cars is located on electronic control units (ECUs). Amongst others, they house safety features in the airbag control unit, driving-related features in engine and transmission control units, or driver assisting features like in the adaptive cruise control unit. This widespread use of ECUs makes testing an important and extensive task. Figure 2.1 shows a reduced V-model representing the stages of the ECU development process in the automotive industry. As ECUs contain a hardware and a software part,

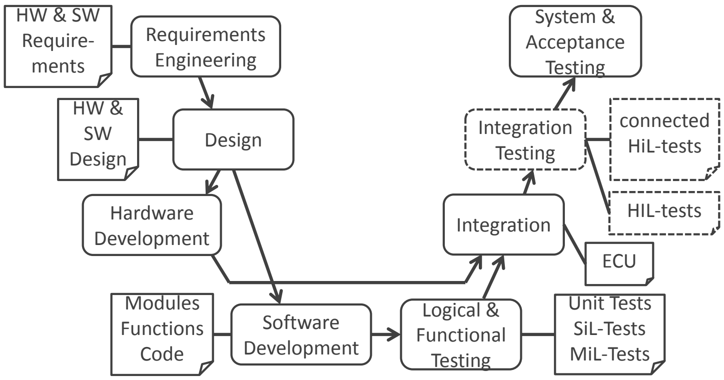


Figure 2.1: V-model for the ECU development process. The elements surrounded by dashed lines represent the integration testing stage and the related artifacts thereof, representing the context of this study.

integration testing plays a major role. At the studied company, integration testing is performed using different hardware-in-the-loop test rigs (HILs). In a first step the ECUs are tested in single HIL systems to test the logic and functionality, while in interconnected HIL systems multiple ECUs are tested together. Simulations are used in interconnected HILs to replace missing or not yet finished ECUs.

2.1.1 Problem Domain & Motivation

Integration testing is a resource-intense and complex task. Multiple different parties like testers, developers, HIL-platform specialists, and tooling experts are involved, who are often employed by different suppliers. They provide various artifacts like requirements, test specifications, tests, test-environments, and test-tools. This study investigates the challenges emerging from this constellation.

In a previous study we found high complexities and low maintainability among the ECU simulations [23]. According to Li et al. [21], maintainability is the quality requirement which is affected most by technical debt. Hence, in the course of eliciting challenges we additionally investigate which types of technical debt can be identified and are related to them.

2.1.2 Goal & Research Questions

From the previous observations, the following two research questions are extracted:

RQ-1: What are today's challenges when integrating and testing deliverables at interconnected hardware-in-the-loop test rigs?

RQ-2: What are potential root causes for these challenges and how do they relate to technical debt?

2.1.3 Contributions

This study provides insights in the challenges currently present during integration testing at a large OEM in the automotive industry. The challenges that were mentioned most often by engineers are highlighted. This is intended to help other researchers and industrial practitioners to find important issues in their own context. The identified root causes and potential solutions help to better understand present challenges with a future goal to support their mitigation. Identified technical debt is listed and indicates correlations between current challenges and technical debt in the context.

2.1.4 Structure of the Article

The rest of the article is structured as follows: In Sec. 2.2, we are discussing related work to this study. The design of the study is described in Sec. 2.3; its results are reported in Sec. 2.4 and discussed in Sec. 2.5. The article is concluded in Sec. 2.6.

2.2 Related Work

Multiple studies investigate challenges in the software development process considering different contexts and granularity. The studies from Broy [1] and Grimm [44], for example, investigate challenges in the entire field of automotive software. These studies represent a more general scope, covering all areas of the software development process. In this study we focus on the very specific part of challenges during integration testing in the automotive field.

Furthermore, in the last 20 years numerous papers investigated technical debt in the software development process. Ninety-four relevant studies on technical debt have been analyzed by Li et al. [21], covering the years from 1992 to 2013. Martini et al. [45] specifically assess technical debt in industrial companies containing automotive industries as well. They, however, focus specifically on the architecture kind of technical debt. This study does not restrict the focus area for technical debt in our context of study.

2.3 Design of the Study

We based this study on a survey by using the interview technique as reported by Shull et al. [46]. Its design and structure are described in the following.

2.3.1 Design of the Survey

We reached out to 13 participants from the integration and testing team managed by a subsidiary of a large German premium OEM. These engineers are both working as resident collaborators on-site but also conduct testing services on their own premises. We selected three integration engineers, three test engineers, three developers for ECU simulations, the team leaders for each division respectively, and the leader of the whole department. All of them are involved in integration testing for current vehicle series development projects. Their responsibilities are even overlapping as the studied company expects the developers to have knowledge of the testing and integration steps as well. Additionally, the test automation tools are realized using a model-based approach. Including their team leaders, the engineers have working experience between one and ten years with an average of 5.42 years and a standard deviation of 3.06 with integration testing in the automotive context.

We conducted semi-structured interviews using the questionnaire as shown in Sec. 2.3.2 and recorded the responses by taking notes. Afterwards, we presented our notes to the interviewees for clarification purposes. The interviews took between 30 and 60 minutes and approximately 40 minutes on average.

The interview notes were then coded based on the guidelines provided by Seaman [47]. Firstly, the notes were scanned for actual challenges. By assigning tags, the challenges were subsequently categorized. The categories have been created by two researchers individually, were compared, and discussed in order to find a common base. The following rules were applied during coding:

- One challenge can have multiple categories.
- If a challenge is caused by another challenge in the same category, they are considered as one.
- If a challenge is split up by the experts in sub-challenges of the same category, they are considered as one.
- If the experts' perception of a challenge differ, both perceptions are considered.

Additionally, challenges related to technical debt were highlighted. The evaluation if a challenge represents a kind of technical debt is based on the definitions given in Cunningham [22], relating it to "not-quite-right code" and Brown et al. [48] extending the definition to other artifacts of the software development life cycle as well. Hence, in this study technical debt is defined as an action performed in order to meet short-term goals while accepting quality issues and delaying long term goals. Categories for debt are taken from Li et al. [21].

2.3.2 Questionnaire

We used the following questions in our survey to collect the data:

[A] What is your role?

[B] What are your responsibilities fulfilling this role?

[C] What are the most resource-intense challenges that you have to deal with during integration testing?

[D] What were the measures or counter-measures decided to be conducted to overcome these challenges?

[E] What was the root cause of the respective challenges?

2.4 Results

From the interviews, challenges, causes, and solutions as well as technical debt could be identified.

2.4.1 Challenges

After evaluating the results from the interviewees, we identified the following categories about challenges during system integration and validation:

Process and Communication: During the work, distributed responsibilities as well as long communication channels are considered as challenging to meet planning deadlines. This concerns the communication between OEM, suppliers for platform hardware, suppliers providing software and testing services, and suppliers responsible for the tooling involved. Interviewees mentioned challenges when information or deliveries are not received by the required parties or if they are incomplete.

Requirements: Challenges with requirements are mentioned twofold: Firstly, existing requirements specifications are partially considered being incomplete so that the implementation can be achieved properly; secondly, the corresponding behavioral models, for which the department has to develop corresponding plant models, are continuously evolving. Applying a common requirement format across all participating parties was mentioned as challenging as well.

Documentation: Incomplete or partially outdated documentation and code commenting was mentioned as a challenge particularly by newly involved engineers who are still familiarizing themselves with the system environment. In addition, scripts were created to automate tedious or potentially error-prone development and testing tasks; however, the documentation for such scripts is insufficient in some cases to understand scripts that are sometimes complex.

Architecture: The interviewees stated that the models, with which they have to deal, only have a prescribed structure on the higher levels. While this structuring was reported to be helpful for simplifying the overall modeling, some models tend to be very complex on deeper levels. Another major challenge was reported in the overall modularity of the entire software. It was mentioned as challenging to manage all the different ECU software libraries and their variants. If libraries depend on others, updates can have influencing effects on other parts of the system as well.

Code: The aforementioned incomplete requirements cause implementation models that may be based on assumptions; hence, such models need to continuously evolve to correspond to the changed and further elaborated requirements later. In addition, however, such models growing in size and complexity also cause further challenges as they need to be continuously updated and maintained. In some case, “dead code” was reported being present in simulation models that is actually not needed but its removal is considered to be too time consuming.

Tool: Regarding the tools in use, developers report mainly two major challenges: Firstly, the proprietary development environment is lacking sufficient support to address all the code and platform challenges faced by the engineers. In addition, upgrading to a newer version is experienced as challenging which partly leads to the use of older tool versions if the migration to the new version consumes to much time. Furthermore, tools are partly not flexible enough to handle large scale models where engineers are often experiencing lengthy loading and build times. Secondly, the commercial tools in use are extended by own scripts to automate tasks; here, developers face complex and dependent scripts that are in addition challenged by non-typical models or tasks during the development and thus, continuously require manual adaption.

Platform: As the implementation and plant models are used on specific HIL-hardware environments; the update to newer hardware environments as well as using different versions at the same time is challenging the model development and maintenance. Defects in the hardware environment are reported to be difficult to spot and being identified actually as a hardware problem.

Third Party Software The engineers are facing challenges regarding inconsistent type information among third party software and even when reusing software among the same company. They also report about them being faulty and outdated. Additionally, the need for a specialist for a specific part of the software was mentioned as a challenge.

2.4.1.1 Suggested Measures for Improvement

We additionally asked the engineers to outline potential solutions to the stated challenges; regarding process challenges, the engineers suggest to better align the individual tasks and to improve the communication of operational and management levels. Considering tool challenges, engineers suggest to better test new versions of proprietary and own tools before deploying them to the entire process; in addition, an appropriate visualization for the scripts is suggested. To address architectural challenges, developers propose the greater modularization and a decrease of internal dependencies. According to the engineers code challenges can be mitigated by automation. Regarding requirement challenges, the engineers wish mostly for improvements in traceability.

2.4.1.2 Identified Root Causes

Most of the collected challenges cause subsequent and dependent challenges themselves. For example, a lack in communication between parties can result in unclear requirements, nonspecific tests, and faulty code. General root causes, however, were mentioned lacking time and work force, and the inevitable distribution of responsibilities in the integration and integration testing process among different participating parties.

2.4.2 Technical Debt

According to the very broad specifications given on the categories of technical debt in Li et al. [21] or Alves et al. [49], in the integration and integration testing process alone, all possible technical debts could be identified. As, this does not provide a statement on how much effort it would take to repay the respective debt, neither about if it has an effect on the business value of the company, this is not considered as a severe problem. Next to the typical code, test, and architectural debt created by focusing on short-term goals over long term quality, the interviews revealed two apparent debts in the context:

Requirements Debt: According to Li et al. [21], this is one of the least studied kinds of technical debt but appearing repeatedly in the challenges mentioned by the engineers. Following the definition from Ernst [50], we interpret requirements debt as distance between the optimal specification and the actual one.

Documentation Debt: Taken from the interview results, documentation debt seems to be relevant in the automotive context as well. Li et al. [21] summarize it as "insufficient, incomplete, or outdated documentation". Together with the constant requirement change and a possible lack in communication, documentation debt might hinder awareness about who made which kind of change in code, tests, platform or tooling.

2.5 Analysis & Discussion

The coding resulted in two categories that stick out. Challenges related to process and communication were most frequently mentioned, followed by challenges regarding code. Even when grouping the engineers by their responsibilities as test engineers, integration engineers, developers and leaders, the same two categories appear most often in all four groups. The mentioned code challenges are not surprising as many of them are well known in any kind of software development. Still, some of them seem to be particularly prominent in the automotive field. The management of multiple interconnected ECU variants or the presence of several different hardware platforms at the same time might be an example. Those challenges are grounded in daily work on modeling tests or simulations, where requirements are changing continuously.

An indication for the high amount of process and communication challenges can be derived by grouping the engineers by experience. Process and communication challenges are mentioned less often by engineers with less than two years experience at the studied company. Two years ago, a major change in the collaboration model has been introduced to better reflect the company's compliance guidelines. This change seems to have a strong effect on the engineers work who are still adopting it. Still process and communication challenges do not come completely unexpected either, as integrating software and hardware in the given context comprises multiple parties and artifacts, which have to be synchronized.

When relating the challenges identified to existing research on technical debt and their causes in similar contexts we also found similarities. We compared causes for technical debt mentioned in Martini [45] with challenges identified in this study. They mention business factors, design and architecture documentation, reuse, parallel development, effects uncertainty, non-complete refactoring, technology evolution, and human factors. In their paper, they relate those causes to architectural technical debt. As these causes strongly overlap with the results received in this experience report, we see an indication that they are applicable to other kinds of debt as well.

There is a noticeable threat to the external validity of the study. Because the study was performed at one company and with 13 participants, the generalizability of the results is limited. We mitigated this threat by diversifying the selection of participants as much as possible by including developers, testers, integration engineers, and project leaders and expect to observe similar results in comparable contexts in the same domain.

2.6 Summary & Conclusions

Integrating different software deliveries both, with other software components and with hardware is a growing complexity challenge in the final phases of a vehicle development project. In this article, we are reporting about experiences from a department working with interconnected hardware-in-the-loop test rigs. Therefore, we conducted semi-structured interviews with integration and test engineers to capture today's most resource-intense challenges and the probable root causes. Prevalent challenges in the automotive domain could be identified and backed up with contemporary data from industry elicited in the context of integration testing. After clustering the different responses, we identified challenges related to code and process and communication as reoccurring patterns challenging today's engineers.

Future work needs to further investigate the process changes in the company that led to the high number of process and communication challenges, by conducting an in-depth case study to unveil contributing factors. Furthermore, analyzing code and design of tests and simulations next to requirements and documentation is suggested in order to tackle potential cumulating technical debt.

Acknowledgments

We are grateful to all integration and test engineers who supported this experience report significantly with their feedback.

Chapter 3

Paper B

Simulation and Validation of a Safety-Critical Electronic Control Unit for Integration Testing in Connected Hardware-in-the-Loop Environments

J. Schroeder, C. Berger, T. Herpel

Proceedings of the 3rd International Symposium on Future Active Safety Technology Towards zero traffic accidents (FASTzero), Gothenburg, Sweden, September 9-11, 2015.

Abstract

Model-based software using Matlab & Simulink is indispensable in the automotive sector. Hence, the approaches for requirements engineering, development, verification, and validation in this area are deeply studied. This study focuses on their specific application for simulation models of safety-critical software and hardware components in the domain. A methodology for the above-mentioned software development steps is proposed. Each step is explained and considerations regarding safety are outlined. The study concludes with showing the feasibility of combining stakeholder knowledge with current literature on model-based development.

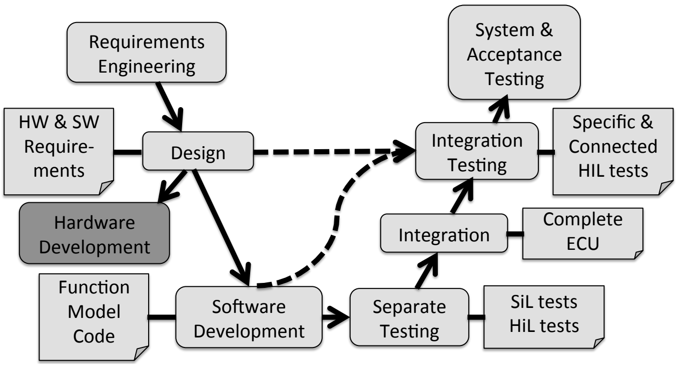


Figure 3.1: V-model for the ECU development process following the solid lines. The dashed lines show the approach for ECU testing using simulation investigated in this study. The hardware development step highlighted is not split up any further as it is out of the scope of this study.

3.1 Introduction

Electronic control units (ECUs) responsible for safety-critical functionality in vehicles become more and more common. As they undergo continuous evolution, permanent verification and validation are required in order to ensure their functionality and quality.

3.1.1 Problem Domain & Motivation

In industry, verification and validation of ECUs is done using highly specialized Hardware-in-the-loop (HIL) test systems during integration and testing phases. Additionally, to ensure correct operation of the ECUs in an interconnected system like a complete vehicle, connected HILs are used. To ensure an efficient utilization of the connected HIL as a resource, real ECUs are complemented with simulations realized in Simulink. Simulations enable testing of single or multiple ECUs in a real-car environment even if not all ECUs are fully developed.

The general development process for an ECU involves the steps depicted in Figure 3.1. After the design has been created from the requirements, the development process is split up into a hardware and a software part. For an ECU to be used on a HIL, the software part has to be integrated with the hardware after both parts have been tested separately. This dependency bears the risk of delays if either the hardware or the software development is not finished in time. Hence, a different approach for the connected HILs was introduced depicted by the dashed lines in Figure 3.1. Simulations of the ECU software and its physical surroundings called plant models are created and connected to real ECUs or other simulations at the connected HILs. These simulations enable testing functionality at connected HILs without the ECU?s hardware. The approach depends on existing interfaces for the bus communication. Hence, the design, containing clearly defined specifications of the communication from and to the ECU is required to apply this approach.

3.1.2 Goal and Research Question

The goal of this work is to analyze the process of realizing a simulation model for the safety-critical ECU, like the Airbag control. Experience from existing projects showed that lack of maintainability can lead to severe problems in later development phases. If a model is difficult to maintain, small flaws in code and structure can accumulate and high effort is required if solving the issues becomes inevitable. This kind of problems is also called Technical Debt (cf. [51]).

From the statements above, the following research question emerges.

RQ: How can best practice from literature and from stakeholders familiar with existing preconditions in industry be combined to a methodology for simulation and validation of a simulation model?

Hence, it is analyzed which approach for the simulation of model-based software results from an investigation in the given context. The methodology shall include a structured approach for requirements engineering, requirements prioritization, software development, verification, and validation. As the ECU to be created is a safety-critical component in the car, considerations in this regard are part of the study as well simulating safety-critical ECUs.

3.1.3 Contribution

This study focuses on a problem in industry. Hence, it contributes with a process ready to be applied to projects creating simulation models in industry. Additionally, each step in the process is examined attentively, to provide additional background on limitations and delimitations of a method. Those limitations and delimitations might be generalizable to other projects outside the domain as well.

3.1.4 Scope

This study covers simulation models of ECUs in the context of automotive software engineering. The models simulate software and also hardware parts of the ECUs, like sensors and actors. Still, the hardware and software development of the real ECUs are not in the scope of this work.

3.1.5 Structure of the Article

This article is structured as follows. In Section 3.2, the design and methodology of the study is outlined. Section 3.3 describes the results received from applying the methodology in the context described in Section 3.1. In Section 3.4, the results are analyzed and the study is concluded. The study is summarized in Section 3.5.

3.2 Design of the Study

To understand the process of the creation of simulation models, the relevant data-flows and its required behavioral models need to be identified, iteratively realized, and validated in the HIL environment. This includes requirements engineering, software development using Simulink, verification, and validation.

Continuous integration and incremental development are necessary as features are realized iteratively and evolve over time. This and the multiple different variants require the software to be maintainable. Hence, these considerations shall be part of the study as well.

The process of mapping the safety-critical ECU to a simulation model for the connected HIL is studied with a strong focus on incremental realization and continuous validation. Therefore, relevant stakeholders are involved and complemented by findings from similar safety-critical systems found in literature. In addition, the influence of the safety standard ISO-26262 on the simulation model is investigated. Each step is validated in stakeholder workshops and by a second researcher, to increase its applicability in the context of the study and to ensure method and data triangulation. Existing development processes mentioned in the literature are mostly meant for creating ECU software and not their accompanying simulations (cf. [52] and [53]). In this study, those processes will provide considerable input, as a gap was identified in literature for processes specifically designed for the creation of ECU simulations.

A stakeholder workshop in this case means a meeting in which at least one but mostly multiple engineers come together and discuss intermediate results with one of the researchers.

Next to other less frequently involved stakeholders, seven participated repeatedly in the workshops. They are three software developers, three software testers, and one department manager. On average they have a work experience of 4.5 years in the context of model-based software engineering for the automotive industry.

3.3 Results

The result of this study is a systematic process for creating a simulation model. It has to fulfill properties relevant for a safety-critical ECU used in connected HIL environments. All relevant stakeholders and current literature are considered. The investigations resulted in the following list of activities part of the process. The process is also summarized in Figure 3.2.

[A] Requirement Prioritization
The correct features have to be selected for implementation in the simulation. How to prioritize which feature to implement first will be decided based on:

- How relevant is a feature for the ECU in general and for the ECU in an interconnected environment?
- How urgent is the feature to be tested?

The following methodology is used in this step:

- Stakeholder interviews on best practices for feature selection and prioritization
- Stakeholder workshops to verify the lists of prioritized requirements

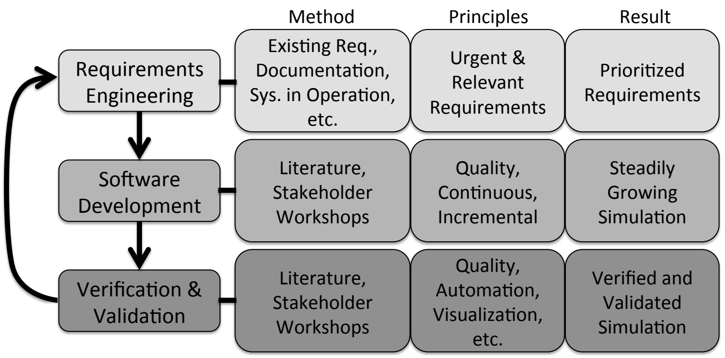

Method
Principles
Result
Requirements Engineering
Existing Req., Documentation, Sys. in Operation, etc.
Urgent & Relevant Requirements
Prioritized Requirements
Software Development
Literature, Stakeholder Workshops
Quality, Continuous, Incremental
Steadily Growing Simulation
Verification & Validation
Literature, Stakeholder Workshops
Quality, Automation, Visualization, etc.
Verified and Validated Simulation

Figure 3.2: Summary of the process resulting from the analysis.

- System in operation analysis, in order to compare which of the requirements other simulations realize

The outcome expected from this step are:

- Validated and prioritized requirements

[B] Development of the simulation
The development process shall follow these principles:

- The process has to be continuous and incremental
- The created simulation has to follow guidelines for maintainable software, elicited from the industry's best practices and from literature

The following methodology is used in this step:

- Stakeholder interviews on best practices for the creation of simulation models and to which extend they address maintainability.
- Literature review on designing and developing model-based software for maintainability
- Validation of the applicability of the elicited approaches from industry and literature using comparison in an experiment

The outcome expected from this step are:

- An incrementally growing simulation of the ECU developed while having focus on maintainability

[C] Verification and Validation
The validation has to ensure that the implemented simulation is a plausible representation of the original ECU and thus, functional and quality requirements have to be verified. This step consists of:

- Validation and verification of both, the simulated ECU alone and in combination with other modules on a connected HIL.

- Assessment of the simulation's maintainability
- Automation of the validation process
- Visualization of results received

The following methodology is used in this step:

- Stakeholder interviews on best practices for the model validation in industry
- Literature review on software engineering approaches for the validation of simulation models and model-based software in general
- Validation of the elicited approaches in an experiment comparing expert opinion and literature
- Validation of the resulting approaches for applicability in industry in stakeholder workshops

The outcome expected from this step are:

- A verified and validated simulation model for relevant properties of the Airbag ECU.

3.3.1 Requirements Elicitation and Prioritization

For requirements elicitation and prioritization, techniques are used that have proven as being useful in the automotive industry, as outlined by ([54]). The elicitation is based on triangulation of data gathered from existing requirements, documentation, systems in operation, and from stakeholder interviews. In this case existing requirements play a major role as the real ECUs are usually specified already. These existing specifications facilitate the elicitation process. Still, as the simulations are reduced versions of them, the other sources are used as well. Additionally, the prioritization becomes important. The set of requirements the final simulation has to fulfill will be ordered by urgency and relevancy.

3.3.2 Development of Simulation Models

The vastly observed approach for developing simulations at the studied company comes with the nature of its structure. The department for testing ECUs in connection receives test assignments from the departments responsible for the actual ECU development.

That means, that functionality is added to the simulations as needed. Either caused by software evolution or because of extension of the test requirements. Therefore, some simulations might not be changed for months, while the ones that are part of the current test assignment might experience multiple changes. Hence, in reality the development parts of the results vary.

As in this case, the ECU simulation is not an extension of an existing one but a new development, the respective development steps have to be described as well.

3.3.3 Verification

Software verification can be defined as "The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase." [55]

If not done manually, testing and verification of model-based software systems is naturally based on model-based testing approaches. There is a wide range on literature regarding general approaches testing for different coverage criteria of the models. When it comes to testing of safety-relevant software, this is realized using formal methods and model checking. For example, there are methods applied in software for railway stations (cf. [56]) or systems in the healthcare, nuclear, and military industry (cf. [57]). They can be used to formally proof that the system works correctly.

Formal methods do not come without drawbacks, though. Amongst others, the literature mentions a lack of scalability, understandability and proper tools (cf. [58–60]). Earlier research has started to look into these problems. In their paper, [61] analyze the integration of formal methods into the industrial context. Additionally, for simulations as mentioned in this study, which are usually realizing only parts of the functionality and proportionally smaller compared to the real ECUs, scalability issues might only apply for simulations of high complexity.

3.3.4 Validation

Software validation can be defined as "The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements." [55].

The issue that the simulations implement only a subset of requirements distinguish the validation approaches from them applied to the real ECUs. It might actually occur, that next to different functional requirements also different quality are required in the simulation. Hence, the question arises, how can it be ensured, that the simulation actually reflects the reality?

In their literature study, [19] found six quality requirements, which current literature presents assessment strategies for. They are correctness, completeness, consistency, comprehensibility, confinement, and changeability. As mentioned earlier, maintainability is an important attribute of the simulations studied. In a previous study it was shown that maintainability can be assessed by measuring complexity [23]. Hence, literature provides existing approaches to support validation of model. In most cases, their applicability in industry still has to be shown.

3.3.5 Safety

The simulations are not going to be used in a real car. Still, to ensure that the final vehicle software fulfills the safety requirements and to increase amount of recognized safety-related issues as early as possible, it seems reasonable to apply methods common in the ECU development for the simulations, as well. [62] use fault injection and mutation testing for validation and verification early in the software development life cycle. They follow the recommendation of the ISO-

26262 standard. Their approach could be applied to simulations in integration testing. A mutated simulation could reveal faults in the interconnected system.

3.4 Analysis and Conclusions

This study presents one possible approach for development of simulations for safety-critical ECUs in the context of integration testing using hardware-in-the-loop test rigs. Analyzing the process showed that special attention has to be paid to validation and verification of simulation models as required functionality and quality might differ from the real ECU. Regarding safety, approaches from the regular ECU software development seem to be applicable for most of the verification and validation steps.

Regarding threats to validity, using findings from stakeholder workshops from a single company can bias the methodology, as those stakeholders are the ones using the current process that is thought to be improved. The workshops were necessary though, in order to ensure that the new process fits the context and the needs of the respective stakeholders. This threat of bias is mitigated by including a second independent researcher to discuss each finding received.

Generalizability is limited as stakeholder workshops have been performed in one company. According to [63], it could be achieved by repeating the study using different cases in similar contexts. As no similar studies could be identified, this threat is not negligible.

3.5 Summary and Future Work

In this study we combined stakeholder knowledge in the automotive domain with current literature to create an approach of developing simulation models of ECUs. In order to ensure the applicability of the process, it will be validated by simulating a real ECU in the same field as part of future work.

3.6 Acknowledgments

The authors wish to thank the developers, test engineers, and managers who supported this work with valuable comments and suggestions.

Chapter 4

Paper C

Comparing the Applicability of Complexity Measurements for Simulink Models during Integration Testing – An Industrial Case Study

J. Schroeder, C. Berger, T. Herpel, M. Staron

Proceedings of the 2nd International Workshop on Software Architecture and Metrics (SAM), Florence, Italy, May 16, 2015.

Abstract

Context: Simulink models are used during software integration testing in the automotive domain on hardware in the loop (HIL) rigs. As the amount of software in cars is increasing continuously, the number of Simulink models for control logic and plant models is growing at the same time.
Objective: The study aims for investigating the applicability of three approaches for evaluating model complexity in an industrial setting. Additionally, insights on the understanding of maintainability in industry are gathered.
Method: Simulink models from two vehicle projects at a German premium car manufacturer are evaluated by applying the following three approaches: Assessing a model's (a) size, (b) structure, and (c) signal routing. Afterwards, an interview study is conducted followed by an on-site workshop in order to validate the findings.
Results: The measurements of 65 models resulted in comparable data for the three measurement approaches. Together with the interview studies, conclusions were drawn on how well each approach reflects the experts' opinions. Additionally, it was possible to get insights on maintainability in an industrial setting.
Conclusion: By analyzing the results, differences between the three measurement approaches were revealed. The interviews showed that the expert opinion tends to favor the results of the simple size measurements over the measurement including the signal routing.

4.1 Introduction

Software takes over more and more responsibilities in current vehicles. The amount of software, realized as control units responsible for different functionalities like gear shifting, airbag control, or cruise control is increasing. In order to test the control units, the industry applies hardware-in-the-loop (HIL) testing. That means to connect a control unit to virtual versions of other units. In one further step, all virtual control units are connected with each other using virtual bus systems. In these compositions, real control units can be tested while the software simulates the rest of the complete car. In order to achieve the best results, the simulation has to represent the real vehicle as good as possible or at least as close as the test requires it to be.
The department where the study was conducted is responsible for creating plausible software models representing the complete vehicle on multiple HIL platforms. Simulation models for all control units used in the car and the respective physical plant models are created using Matlab and Simulink. The validation of the increasingly complex vehicle models is a more and more urgent challenge, as it has to be ensured that complete HIL simulations actually simulate the car properly.

4.1.1 Problem Statement

As control unit and plant model software are growing, integration and test engineers are continuously challenged to master the growing complexity as well. For example, the case of Toyota's unintended acceleration case outlined by Koopman [6] shows how important testing of control units in real or simulated environments is. Hence, in this domain, assessing the complexity of models is just as important as assessing code complexity.
Code complexity is well understood and there are many established measurements and metrics for example designed for object-oriented software (cf. Tegarden et al. [64] and Chidamber and Kemerer [65]). Models are created to get an abstraction from the code level, therefore code complexity metrics cannot be applied without a modification. Finding measurements designed for model-based software like Simulink is still a challenge. Additionally, it is not clear how applicable the measurements are in industry and if existing measurements are accepted among software engineers in the automotive field.

4.1.2 Research objectives

The assessment of model complexity measurements is chosen as a means to address this challenge. Complexity measurements are of interest as they assess amongst other things complexity, maintainability, understandability, and the probability of errors in the models (cf. Plaska [66]). The measurement results shall improve the integration testing phases. Hence, the study investigates to which extent existing metrics for complexity of Simulink models are applicable in the field of automotive software.
Complexity of software is correlated to software maintainability. For example, Kemerer [67] combined the existing evidence until 1995 about their relationship. Banker et al. [68] and Plaska [66] also describe a connection between them.

Consequently, maintainability investigations of software models provide a promising related research objective and shall be part of the study, too.

4.1.3 Context

The study takes place at an electrical development department of a large German premium car manufacturer producing more than 1.5 million vehicles per year. The department is responsible for about 80 Simulink models of control units for different car variants. In 2013, the department performed tests on almost 190 HIL systems of which around 15 are interconnected systems.

4.1.4 Document Structure

The structure of this study accords to the guidelines to case studies from Runeson and Höst [38]. Section 4.2 describes the related work to this study. In Section 4.3 the design of this study is outlined while Section 4.4 shows its results. Lastly, Section 4.5 summarizes the study, shows its impact, and lists possible follow-up studies.

4.2 Related Work

The complexity evaluation approach used in this study was first mentioned in 1988 by Card and Agresti [69]. Their approach aims for assessing the design of software. The measurement is defined as a combination of the complexity inside a block and the complexity of the relations between blocks referred to as local and structural complexity. The local complexity measure represents a value for the cohesion and the structural complexity for the coupling of a system. Additionally, the authors evaluate their metric in real projects and analyze ways to minimize complexity by adjusting design characteristics.
Plaska and Waldén [70] port the approach mentioned above to the Simulink setting in order to evaluate hydraulic control systems. One of the authors described the metric more thoroughly together with other Simulink-specific measurements by Plaska [66]. Boström et al. [71] created a tooling which enables automatic measurement of Simulink models using the metric defined in the two previous papers. In this study, their metrics are used to realize the measurement defined by Card and Agresti [69].
Apart from the mentioned work, the literature provides few studies applying rigorous research methods to investigate the complexity of model-based software. Dajsuren et al. [72] apply similar structural metrics to assess modularity of Simulink models. Feldmann [73] applies Cyclomatic Complexity metrics and counts lines of code to measure complexity. The company this study is conducted at uses the same two approaches. Antonio and Ferrari [74] use the metric based on the work of Card and Agresti [69] to relate complexity to the comprehensibility of models. None of the studies assesses the applicability of the complexity metrics in the respective fields.
In the field of object oriented software engineering comparable studies can be found. For example, as part of their study, Anda et al. [75] assess software maintainability using code size measurements and compare them to the judgment of experienced experts. Their results show, that the expert opinions agree

with size measurements. In this paper a similar approach is applied to reveal which of the three measurements is preferred by the experts in the context described in Section 4.1.3.
It can be concluded, that only few reported complexity measurements specific to model-based software exist. Furthermore, evidence to what extent the existing measurements are applicable in industry is missing in the existing literature, especially addressing the integration phase of software models.

4.3 Case Study Design

The goal of this research is to analyze how applicable complexity measurements are in the mentioned industrial context. This includes applying complexity measurements to the vehicle models. The results received from the complexity measurement mentioned by Card and Agresti [69] are going to be compared with two software size measurements. This enables a comparison of the three different approaches which are expected to have correlating results. Additionally, an interview study is performed to assess the expert opinion from software developers in the field and therefore to check if the model-complexity metrics are aligned with the designers' perception of complexity. The results received in the interviews are then validated in consecutive workshops.

4.3.1 Research questions

From the goal of the study, the following research questions were derived.

[A] How do the results from one composite complexity metric and two different size metrics correlate with each other in the industrial setting?

[B] To which extent are the complexity measurements used in this research applicable to support the integration testing of Simulink models in the context of HIL simulation for automotive software?

[C] What are further indicators on how to measure/evaluate maintainability of Simulink models extracted from the expert opinions of software engineers at the studied company?

4.3.2 Case and subjects selection

As mentioned before, the study takes place at a HIL testing department. The models created at the department represent a small collection, compared to the models created for the real control units of the cars. But as they simulate the whole connected vehicle and its environment they are a valid representation to the population of all models at the studied company.
The modeling is performed in multiple sub divisions of the department. The interviewees were chosen as they are the group of engineers responsible for performing a part of the development and the whole integration testing of the mentioned models. They are software developers with different engineering backgrounds.

4.3.3 Data collection procedure

The data in this study is collected in two ways. First, measurements are performed on the models of the vehicle control units. According to Card and Agresti [69] and Plaska and Waldén [70] the first complexity metric is defined as the sum of structural and local complexity. Whereas the structural complexity is a relation of the number of calls a block makes to other blocks (fan-out) and the whole number of blocks in the system. The local complexity is represented by the number of input and output variables of a block in relation to the fan-out and the number of blocks. In the following, the sum of structural and local complexity summed for all blocks in the model will be mentioned as IO/Fanout (IOF). The metric is performed on the Simulink mdl-files. In an mdl-file, the content of the model is represented in a structured text-based format. The first size measurement Block Count (BC) uses a function included in Simulink to count the number of blocks in a system. The *sldiagnostics* command can be called from the Matlab console with the parameter *BlockCount*. Through all the layers of the model the existing blocks on each of them are counted and summed up. For the second size measurement, the lines of code of the Simulink mdl-files are counted. This metric is considered relevant as the mdl-file contains all information on the structure of the model.
The second step in the data collection procedure is an interview capturing the expert assessment of the results. The following questions were used during all interviews:

[A] In your opinion, when is a Simulink model maintainable? What makes it maintainable?

[B] How should maintainability be evaluated in practice?

[C] (a) On a scale from 1 to 6, how well suited is metric A for evaluating the maintainability of a model?
  (b) On a scale from 1 to 6, how well suited is metric B for evaluating the maintainability of a model?
  (c) On a scale from 1 to 6, how well suited is metric C for evaluating the maintainability of a model?

[D] (a) From the list of control units, which of them do you know well enough in order to make a statement about its maintainability?
  (b) On a scale from 1 to 6 how maintainable is each of the control units known to you?

[E] (a) Based on the measurement results, order the three approaches according to how close they come to your personal complexity assessment of the models.
  (b) In your opinion, which of the measurement results was created using which approach?

The two open ended questions in the beginning are considered to introduce the topic and elicit major aspects of maintainability important for each of the engineers. During the whole interview, answers for these open ended questions are elicited and noted. After introducing the topic, in 3 the engineers were asked to rate the measurements using three questions in a Likert-format. At this point no measurement result has been shown to the interviewees yet. The scale ranges from very easy to maintain (1) to very difficult to maintain (6). A scale from 1 to 6 was chosen in order to avoid completely neutral answers. The questions 4a and 4b assess the expert opinion on the complexity of the models which are among their responsibility. Question 5a aims for the engineers' opinion on the measurements after showing them the results. The names of the measurements were not visible, so the participants did not know which result belongs to which measurement. The scales for evaluating the metrics before and after showing the results are different. This was intended, to avoid that the interviewees give the same answer as before.

4.3.4 Analysis procedure

The analysis is supposed to show relations between the two data collections – the measurements and the expert opinions. Regarding the measurements as the first step, the control units are analyzed one by one and all three measurements described in Section 4.3.3 are applied to each of them. The results are listed and normalized in order to compare them and make statements about similarities, differences, and correlations.
The interviews are analyzed in two ways. The two open ended questions in the beginning are supposed to deliver qualitative results on complexity and maintainability. They shall be used to validate the quantitative results received by the questions following in the interview. For example, to support the resulting constellation of votes for the measurements with the qualitative data or to oppose them.
In order to gather data from the qualitative results, a procedure described by Seaman [47] is used. The interview notes are coded, meaning that tags are assigned to each of the statements made on complexity and maintainability. The tagged notes are then grouped together in order to make them quantifiable and to locate accumulations of tags. The coded results of the expert opinions on complexity and maintainability shall show if they overlap with the quantitative results received from the subsequent interview questions.
The quantitative data received from the questions 4a and 4b is directly compared with the measured data in order to determine if the measurements correlate with the expert opinion.

4.3.5 Validity procedure

The three measurements have their background either in the literature or are practically used in industry. The background for IOF is explained in Section 4.2. The measurement of LOC is widely used in literature and industry and BC is a measure existing in the Simulink development environment. Although their correlation is not fully agreed (cf. Landman et al. [76]), the McCabe complexity metric was left out in this study as it is meant to have a close relationship to the measure of lines of code (cf. Jay et al. [77]).
In order to receive consistent results among all the interviews, the questions asked were always identical. The semi-structured design of the interviews, including the two open ended questions resulted in discussions that were

different from interview to interview. That was intended to elicit as much qualitative data as possible. In order to assess the qualitative data in a structured way and to retrieve valid results, it was coded and analyzed according to Seaman [47]. This provides an established approach to evaluate qualitative data. The workshops to evaluate collected answers and conclusions drawn from them together with the engineers provided additional validation to the interview results.

4.4 Results

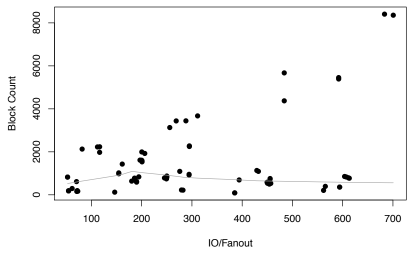
The results are divided in those for comparing the measurements and those received from combining measurements and interviews.

4.4.1 Measurement Results

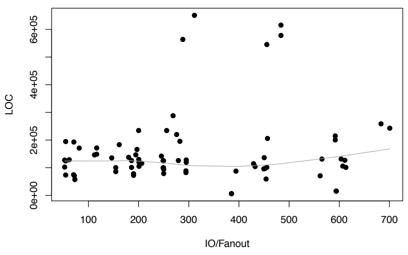
The tables attached as supplementary material in [78] show the results of the measurements in the first four columns. In the columns five to seven, the results were normalized using the formula $Xi = Xi - Xmin/Xmax - Xmin$, in order to restrict the range of the results and to simplify the pairwise comparison for the interviewees. For confidentiality reasons, the names of the control units are anonymized. The table does not represent a complete list of all control units used in the vehicle and at the HILs. Control units for which the interviewees could not provide a statement are not listed.
The three measurements were compared for correlation using the Pearson correlation coefficient. The results for the correlation analysis are shown in Figure 4.1. The captions contain information on the correlation of the measurements. They show that IOF has a weak correlation to both, BC and LOC; BC and LOC correlate moderately. The weak correlation between the size and complexity measurements was expected as the approaches differ greatly. Since no two measurements results show strong correlation, a clear distinction of the experts favorite should be possible when comparing the measurements with the interview data.
In the Figures 4.1b and 4.1c the same outlying models stand out. They contain comparably much functionality and signal routing within few blocks. Hence, those outliers can be traced back to the differences in the model designs and cannot be dropped. They actually might indicate models for further investigation when the company decides to reduce complexity in their models. The outliers in Figure 4.1a indicate two different groups of models as some of the BC values grow with increasing IOF and others do not. Again this behavior can be related to differences in the modeling approaches, as some models contain more functionality and signal routing in only few blocks than other models.

4.4.2 Results from the interview study

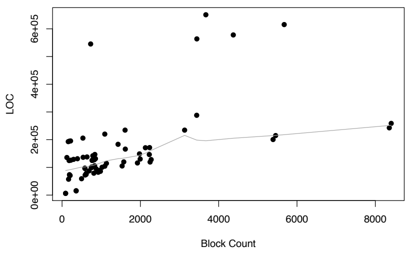
The results for interview question 4b representing a posteriori answers are shown in the last eight columns of the tables supplemented in [78]. Combining interviews and previous results shows how the expert opinion fits the measure-



(a) IOF and BC – correlation: 0.38



(b) IOF and LOC – correlation: 0.14



(c) BC and LOC – correlation: 0.56

Figure 4.1: Scatter plots with regression lines indicating the correlations between the measurements.

ments. The correlations of each expert's results to the measurement results are summed up in Table 4.1.

Table 4.1: The correlation of interview and measurement results using the Pearson coefficient.

Interviewees	number of CUs	IOF	BC	LOC
I1	8	-0.02	0.28	0.00
I2	23	0.40	0.60	0.42
I3	24	0.16	0.73	0.44
I4	63	-0.01	0.03	-0.10
I5	4	-0.98	0.60	-0.39
I6	6	0.98	0.99	0.46
I7	5	-0.94	0.80	0.28
I8	5	-0.94	0.80	0.28

The table also includes the number of control units each interviewee is responsible for. This indicates the weight each of them has. It can be seen, that the correlation between expert opinion and the LOC/BC measurements is higher than the correlation with the IOF metric.
In addition to the correlation observations, three One-Way ANOVA tests were performed post-hoc to investigate the relationships between the experts' opinions and the measurement results. While expecting to observe large effects (effect size = 0.4), a high power ($1 - \beta > 0.8$), and assuming a type-1 error probability of 0.05, the tests resulted in values collected in Table 4.2. In order to achieve this high power, the six possible expert ratings were combined in three groups of two ratings each. The tests found no significant relationship

Table 4.2: The results of the three One-Way ANOVA tests comparing experts' opinions and metrics. The critical f is 3.11.

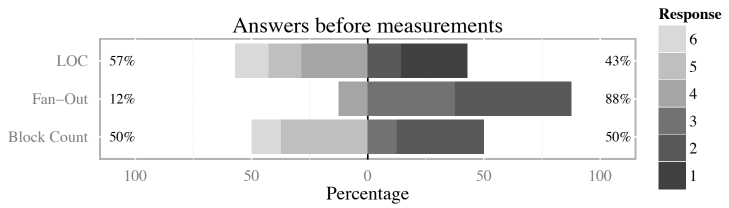
Metric	F-value	p-value
IOF	1.83	0.163
BC	7.40	0.0009
LOC	0.89	0.4136

between any of the metrics and the experts' opinions. The null-hypotheses of equal means were rejected either because of $p > 0.05$ or $F > f$. However, the additionally performed Tukey-test showed that the group of CUs which received the best maintainability ratings by the experts, are also associated with significantly lower BC values. Gathering a larger sample might uncover a stronger relationship between the BC metric and the experts' opinions.

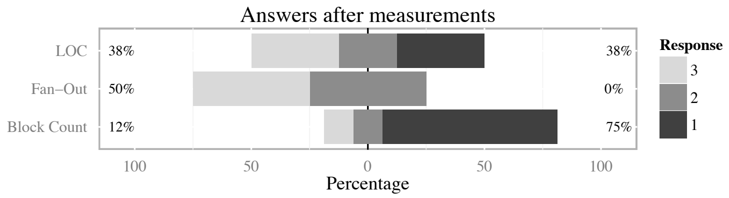
In the interview, question 3 asked for an a priori and question 5a for an a posteriori evaluation of the measurement approaches. The results from both questions are shown in Figure 4.2. The order of metrics favored by the experts changes from the first to the second assessment. The IOF metric is highly accepted, based on the experts' opinion of the underlying approaches, as shown in Figure 4.2a. When assessing the actual measurement outcomes in Figure 4.2b, the metric looses greatly in agreement. While the LOC metric stays average, the Block Count measurement improves in its agreement with the experts opinion in the second assessment and is the one highly accepted according to the experts evaluation of the measurement results. This change in the experts' opinion could be verified applying Wilcoxon Signed Rank test, which detected a significant difference in between the a priori and the a posteriori evaluation for the BC metric.

Finally, the coding of the interview notes of the open ended questions resulted in a list of tags:

- *Documentation* – Issues related to bad or good documentation
- *Communication* – Issues related to inputs, outputs, and signals inside a block and between them
- *Size* – Issues related to the good or bad influence of model size
- *Structure* – Issues related to structural attributes of the models
- *Task* – Issues related to the logic specific to the task a model or single blocks perform
- *Third Party Development* – Issues related to blocks or parts of them, developed by another company or department
- *Tooling* – Issues related to scripts and tools used for development
- *Visualization* – Issues related to good or bad visualization of the models
- *Other* – All issues which did not fit in the previous categories



(a) Rating from 1 to 6 on the measurements given before showing the measurement results



(b) Ranking from 1 to 3 on the measurements given after showing the measurement results

Figure 4.2: Likert-scales showing the interview results for the measurement approach assessment. A higher number stands for a better rating in both evaluations. The scales unveil a change in the expert opinions from favoring the IOF approach before and the BC approach after showing measurement results.

Of this list, the following tags were mentioned most: First, the communication in between blocks, mostly mentioned as strong coupling decreases maintainability. Second, the size of the block. Whereas size was mentioned as having a strong influence as well as having no influence. Third, the structure of the model, mentioning cohesion in blocks as having a positive influence on maintainability. The three tags were mentioned as positive and negative contributions to maintainability and complexity. This shows that the expert opinions gathered from the open ended questions fit their a priori assessment of the measurements shown in 4.2a, which favored the complexity metric IOF. According to the interviewees, structure and IO-Signals should be part of a complexity metrics as well. The tag for model size, representing the LOC and BC measurement, was mentioned in almost all interviews but mostly assigning size a low priority in complexity and maintainability evaluation. The interviewees mention size as a problem that can be overcome with a proper structure. The results received a posteriori draw a different picture, where the LOC and BC measurements came closer to the expert opinion.

4.4.3 Evaluation of validity

The interviews were held by one researcher only. Recording was not possible due to confidentiality reasons. This might have resulted in possible data missed out while writing the protocol. The threat was mitigated by performing on-site workshops validating the answers given by the interviews.
The correlation evaluation and statistical tests for the measurements and the ranking that the experts assigned in interview question 4b bears a threat to validity as only eight interviewees could be taken into account. Additionally,

the number of models the interviewees are responsible for varies greatly. Five of the interviewees are working on less then ten models. This threat was mitigated by analyzing qualitative data from the two open ended questions to validate the results.

4.5 Conclusion and Future Work

This study used three complexity measurements to assess Simulink models of control units. Additionally to comparing the measurements with each other, an interview study showed how applicable and how well accepted the complexity measurements are in the field of HIL integration testing for vehicle simulation. The analysis of the data showed that the three measurements are not strongly correlated. The experts seem to trust size measurements over the complexity one to assess complexity and maintainability. Although, when asking for relevant indicators of complexity and maintainability internal structure and coupling were mentioned most next to the size of a model. Hence, there was a discrepancy in the engineers' opinion what a complexity metric should contain and the measurement they actually preferred.
This study created evidence towards the applicability of complexity measurements in its context and tried to fill the corresponding gap mentioned in Section 4.2. In future studies further validations of the metrics are planned. For example an approach for the validation of metrics from Basili et al. [79]. Additionally, maintainability and other qualities of the available models will be assessed further.

Acknowledgment

We are very grateful to the interview participants for their time and thoughts and to Robert Stoddard for the feedback to our statistical analyses.

Chapter 5

Paper D

Unveiling Anomalies and their Impact on Software Quality in Model-Based Automotive Software Revisions with Software Metrics and Domain Experts

J. Schroeder, C. Berger, M. Staron, T. Herpel, A. Knauss

Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA), Saarbrücken, Germany, July 18-22, 2016.

Abstract

The validation of simulation models (e.g., of electronic control units for vehicles) in industry is becoming increasingly challenging due to their growing complexity. To systematically assess the quality of such models, software metrics seem to be promising. In this paper we explore the use of software metrics and outlier analysis as a means to assess the quality of model-based software. More specifically, we investigate how results from regression analysis applied to measurement data received from size and complexity metrics can be mapped to software quality. Using the moving averages approach, models were fit to data received from over 65,000 software revisions for 71 simulation models that represent different electronic control units of real premium vehicles. Consecutive investigations using studentized deleted residuals and Cook's Distance revealed outliers among the measurements. From these outliers we identified a subset, which provides meaningful information (anomalies) by comparing outlier scores with expert opinions. Eight engineers were interviewed separately for outlier impact on software quality. Findings were validated in consecutive workshops. The results show correlations between outliers and their impact on four of the considered quality characteristics. They also demonstrate the applicability of this approach in industry.

5.1 Introduction

In the automotive industry, software models are used to simulate vehicle functionality of electronic control units (ECUs). In modern cars, ECUs comprise software and hardware for actual vehicle functionality (e.g., engine control, driver assistance functionality, and safety features). Simulations of ECUs are key elements for validation and verification of the real software and hardware during integration testing phases.

5.1.1 Problem Statement

Simulation models are extensive and complex software. At the studied company, their size ranges from 4k to 400k lines of code measured on the model files. They are also highly interconnected. For example, the simulation for the electronic stabilization control (ESC) is strongly connected to engine and braking functionality, distributed over up to nine different simulation models. In their study [20], Schroeder et al. highlighted complexity and issues among creating and using simulation models during integration testing in the same domain as prevailing challenges. This growing complexity and the interconnected nature of the models ask for rigorous assessment strategies in order to control current quality levels and to allocate test resources.

Furthermore, functional and non-functional requirements are clearly specified only for the real ECUs. In contrast thereto, simulation models are typically less extensively specified and focus rather on the expected core functional behavior. Simulation specifications are continuously evolved when new features emerge during integration testing. Reliably validating a model's functional and non-functional attributes in this volatile and growlingly complex environment is a challenge. Validation using software metrics, complementing pure specification-based approaches, is a promising alternative in the domain. However, existing approaches for quality assessment based on software metrics are mostly covering only reliability and maintainability (cf. [80] and [81]). Additionally, solutions are usually designed for object oriented code and not directly applicable to model-based software.

5.1.2 Research Objectives

The goal of this study is to extend current model validation approaches using software metrics. On measurement data received from historic model revisions (software versions based on commits), statistical outlier detection shall reveal anomalous observation. By adding qualitative data received by domain experts, we aim for showing that these calculated outlying observations highlight revisions with high impact on model quality. Additionally, we intend to show the applicability of such an approach in the automotive domain. These objectives are addressed in the following research questions:

[A] How can existing linear regression / model fitting and respective outlier analysis be applied to revision data from an industrial context?

[B] Which anomalies and patterns can be detected applying the above approaches to measurement data in form of software revision time series?

[C] Which correlations between outlying values in the measurements and the domain expert opinions can be found?

[D] Which combination of measurements and outlier detection produces meaningful observations about software quality?

5.1.3 Context

We perform an exploratory case study at the integration testing department at a major German premium car manufacturer, producing around two million vehicles per year. The department manages the development of simulation models and performs integration tests for all interconnected ECUs in their product lines. Currently, the department is responsible for 71 ECU simulation models. Each simulation model covers multiple vehicle variants and is implemented as model based software in Simulink. Simulink is a software tool commonly used in the investigated domain to model physical behavior, process signals, and apply control theory.

We investigate our research questions in this context by applying traditional regression analysis. First, software metrics which are selected based on previous experience in the domain, are applied to the models. Following, regression models are created to fit the measurement data using an autoregressive integrated moving average approach (ARIMA), as we can show its suitability in the context. Thereafter, the well established residual analysis techniques studentized deleted residuals and Cook's distance are used to detect outliers. Finally, we assess the impact of the outliers on software quality through semi-structured interviews with the engineers at the studied company and validate our findings in consecutive workshops.

5.1.4 Contributions

This study demonstrates the applicability of regression based outlier analysis in industry. Additionally, knowledge on outlier data and their correlation to software quality is gathered. We show that correlations between outlier values and four quality criteria can be detected. Furthermore, we provide findings on types and reasons of detected outlying revisions. Altogether, the presented approach complements current validation approaches in the domain as it accounts for information on past, current, and future model quality.

5.1.5 Structure of this Study

In this study we followed Runeson and Höst's recommendation on structuring case studies (cf. [38]). Hence, the paper is outlined as follows: In Section 5.2, we describe the necessary background on all approaches applied in this study and discuss related work. The study design in Section 5.3 outlines the concept applied to investigate the study's research objectives. Results are presented in Section 5.4, analyzed in Section 5.5, and assessed for validity in Section 5.6. We conclude our study in Section 5.7 and comment on possible future work.

5.2 Background and Related Work

The investigations in this study focus on ECU simulation models replacing real ECU behavior. These simulations exhibit varying complexity and usage patterns on hardware-in-the-loop (HIL) test rigs. More functionality is added to these simulation models on request and hence, continuously extending them. Even if a vehicle development project ends, new models usually build upon existing ones and inherit features to a large extent. The Simulink models allow for a graphical representation of physical behavior using blocks and connectors. In layers, blocks can contain further functionality, enabling the creation of interconnected subsystems.

5.2.1 Measurement

For this study, two complexity and two size metrics are used to assess the simulation models. As one objective of this study is the assessment of applicability of the approach in the domain, metrics were chosen that have been proven applicable in the domain before (cf. [23]).

5.2.1.1 Size

We applied two size metrics: Firstly, we count lines of code (LOC) in the model files. The model files are generated by Simulink and store the graphical model in an XML-like format. Every line is evaluated as equally important. The files do not contain empty lines or comments. The second metric is counting blocks contained in the models. It is based on a Simulink internal function called “sldiagnostics”. We are using Matlab and Simulink in version 8.4. This block count metric (BC) counts each block in the model, even those that are inside subsystems, including the lowest layers of the model [28]. Both size measurements are performed offline on the model files.

5.2.1.2 Complexity

By today it is widely understood that no single-valued measurement can satisfy all ideas of software complexity. Briand et al. [82] emphasized the difficulty of defining complexity metrics. Fenton & Bieman ([26], pp. 425) refer to Zuse’s paper [83], proving that the list of properties for complexity measurements established by Weyuker [84] cannot be satisfied by a single-value measure.

In this study we interpret complexity measurements as assessing models in a structural and a local way. We use two measurements first introduced by Card & Agresti [69] in 1988. Plaska & Waldén [70] applied the metrics, which were originally intended for software design, to model-based software. In Schroeder et al. [23], they were used in the automotive domain in an industrial context. The two metrics measure the structural complexity (SC) and the data complexity (DC) of a model.

Structural complexity (SC) aims for assessing the interactions between blocks in a model. Therefore, the fanout value (f) of each block i in a model is measured. Fanout is determined by counting the blocks that are connected to the output of block i. The final metric is the sum of all squared fanouts of a

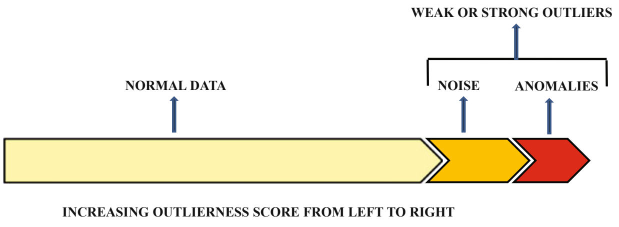


Figure 5.1: Graphic from [87] showing difference between outliers and anomalies.

model, divided by the number of blocks (n), as shown in Equation 5.1.

$$SC = \frac{\sum f_i^2}{n} \tag{5.1}$$

Data complexity (DC) evaluates the workload each block inside a model performs, individually. Additionally to the fanout value (f), it counts input and output variables of a model's blocks. For each block i, the number of inputs and outputs (v) is divided by its fanout value. The sum off all divisions is again divided by the number of blocks in the model (n). Equation 5.2 shows the definition of this metric.

$$DC = \frac{\sum \frac{v_i}{f_i+1}}{n} \tag{5.2}$$

Both equations conform to their definition according to Card & Agresti [69]. Similar to the size metrics, the complexity metrics can be performed offline by parsing the model files.

5.2.2 Anomaly Detection

Outlier detection in general is well studied. Surveys by Chandola et al. [85] and specifically for time series data by Gupta et al. [86] collect research and establish taxonomies for outlier detection. For defining anomalies among outliers, we follow Aggarwal [87]. When assessing measurement data for outliers, the observations can be divided into three classes. Figure 5.1 shows that most of the data indicates normal behavior. Next to the normal data, there is data that behaves differently. These observations are called outliers. In our case, those are measurement values, which are in some way different from the others. Among those outliers is stochastic noise. The noise is naturally present in all measurements and is uninteresting for follow-up investigations. Other observations, which are interesting to the observer are called anomalies; if an observation is interesting or not, does depend on the context. The topic to unveil those interesting outliers among the measurement data in the study's context is addressed in our research questions.

Outlier analysis is well studied and applied in multiple fields. Hartmann et al. [88] investigated already in 1980 the specific field of outliers time series, discussed characteristics of analyzing them, and recommended models to use. In this study, we fit models to the measured observations and perform analysis on resulting residual, to detect outliers in the time series of measurement data.

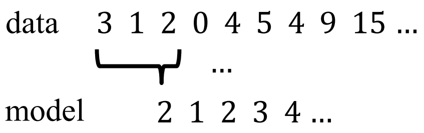


Figure 5.2: Visualization of a moving average model. The model is created building the average of three previous observations: MA(3).

5.2.2.1 Model Fitting

Model fitting and linear regression are common approaches to detect outliers in many kinds of data. For time series data, Hartmann et al. [88] discuss the use of autoregressive integrated moving average (ARIMA) models, which are used in this study as well. Box & Jenkins are known to be the first to apply ARIMA approaches to time series data [89]. Today it is a common approach in general and used in statistics, economics, and electrical engineering, but also areas of computer science, like artificial intelligence.

ARIMA is a combination of three methods; auto regression (AR), moving averages (MA), and integration/differentiation (I). In this study, the MA method is applied to create models fitting the measured time series. A model based on MA is created building the mean value of a set of previous values, as shown in Figure 5.2. In the figure, three values are used for the average resulting in a MA(3) model.

5.2.2.2 Residual Analysis

Once a model is fitted to the data, outliers are detected by analyzing the differences between data and model (the residuals). The two methods used in this study are studentized deleted residuals (SDR) and Cook's distance measure. Both are commonly used methods in practice (cf. [90]).

A regular residual is the difference between an observation in the measured data and a corresponding value predicted by the model. A deleted residual is created by subtracting a predicted value based on an estimate using all observations but the current one. This reduces influence, caused by single data points of the measurements. To create a studentized deleted residual, the standard error of the current residual is subtracted therefrom to create more precise results. Cook's distance measure is an approach combining residuals and leverage values, which can in our case be described as extreme changes in the measurement data.

Both residual calculations can be computed using common statistical packages like R or SPSS. The thresholds, for when one of the residual values should be marked as outlying, are taken from Bowerman et al. [90] and are explained below for replicability reasons.

For SDR, we calculate the t distribution point $t_{0.025}$ with $n-k-2$ degrees of freedom. A residual greater than that value is marked as outlier. The number of observations n in our collected data is always between 100 and 150 and the number of independent variables k is always one. Hence, our calculated minimum value for SDR outliers is between 1.985 and 1.976, depending on n.

The threshold for Cook's distance is calculated similarly. The 50th percentile of a F distribution based on $k+1$ and $n-(k+1)$ degrees of freedom is calculated. A Cook's distance for an observation greater than this value is marked as outlying. Using the same values as above, the threshold for Cook's distance values is always between 0.696 and 0.698.

5.2.3 Similar Studies

Outlier detection is used in fraud and intrusion detection in signal analysis as well as network and system security. In the software engineering domain we found that outliers are used for fault prediction. Alan and Catal [91] detect outliers in results of software metrics applied on different class files. They do not use statistical approaches but their own outlier definition, based on measurement thresholds. Detected outliers are used to improve the fault prediction algorithms. In Hangal and Lam's study, anomaly detection is used for tracking bugs [92]. Their anomaly detection is not based on measurement data but on extracting and refining invariants from running programs. Both studies do not look into the artifacts' development over time or look into other software quality characteristics.

Other studies investigate how well metrics can assess software quality. Many use metrics and apply regression analysis for assessing reliability and predicting faults. Khoshgoftaar and Szabo's study [80] is close to ours as they develop regression models based on complexity measurements. Using neural networks, they predict reliability and faults of software. They are not using outliers for their predictions, though. Herzig et al. [93] analyze software version histories and combine related revisions for a prediction of software defects. They apply software metrics and regression analysis but do use outliers either. Garcia and Shihab [94] use software metrics together with decision trees to detect particularly severe bugs among their data set. They present their prediction model and important factors to determine blocking bugs in software.

Instead of reliability, some studies focus on metrics predicting maintainability. Schroeder et al. [23] assess complexity and size metrics and test them for correlation with stakeholder understandings of maintainability. They showed a preference for size metrics for predicting maintainability in their domain. Based on object oriented metrics, Dagpinar and Jahnke [81] use regression analysis and history data to predict maintainability. Using quantitative maintenance data, they reveal metrics that are able to predict maintainability. Similarly, Gil et al. [95] validate metrics and their assumptions using past software versions. All three studies do not look into outliers among the measurement data.

Software project risk is also investigated and predicted. Choetkiertikul et al. [96] analyze historic data with regression models to detect risks in software projects. Their model detects risk impact and likelihood with certain precision. Pika et al. [97] apply outlier analysis to risk event logs to improve risks indicators for process delays. They do not employ software metrics but improve predictions using statistical outlier detection.

We found that if regression analysis on metrics is used, models are usually based on the measurement data and not on outliers. On the other hand, papers on outlier detection mostly do not assess software quality and sometimes do not apply statistical approaches but own interpretations of outliers. We could not identify studies combining statistical outlier detection based on measurement data for assessing model quality. Additionally, studies mostly assess object-oriented software and hesitate to involve stakeholder opinions.

5.3 Study Design

In order to cope with our research goal in the context of automotive model-based software, the research is performed using an exploratory case study. Thereby, we intend to achieve insights into phenomena observable in the field while avoiding researcher intervention and bias at the same time. Still, we keep the approach as general and clear as possible in order to enable generalizability to similar model-based software as well as replicability in other fields. This is achieved by applying established methods as outlined in Section 5.2, and reporting on all steps of our study. The general approach chosen for the case study is outlined in six steps. Explanations for these steps follow in the remainder of this Section.

[A] We measure all available model revisions, using two software complexity and two size metrics, and get quantitative data in form of four time series;

[B] We employ outlier detection approaches thereto and get four lists of outlying revisions for each simulation model;

[C] We split the measurement results equally in two sets, a test set and a validation set;

[D] Using the test set, we conduct interviews on the impact of the outlying observations and receive qualitative data of impact estimations in form of Likert-scaled stakeholder assessments;

[E] We compare the impact of outliers based on measurements and the evaluation of engineers. Based on the observations, we draw conclusions about the meaningfulness of the detected outliers and their ability to indicate changes in software quality; and

[F] Using the validation set, stakeholder workshops are conducted to validate results.

The purpose of these steps is to collect evidence on the research questions and to address the first study objective of detecting and evaluating anomalous observations with impact on software quality. An analysis of the approach combined with stakeholder discussions from the last step, address the second objective of assessing the applicability in an industrial context. In this study, we follow the ideas from Eisenhardt [98], to derive general knowledge from this case study. Therefore, we analyze the data received from the field before any hypotheses are shaped, according to her model.

In summary, we apply existing approaches from measurement theory and statistics in industry, observe outcomes, and draw conclusions about their meaningfulness and applicability by including qualitative data from domain experts. Finally, we report about derived explanations and experiences to unveil findings on the goal of exploring possible validation approaches for model-based software in the automotive industry.

5.3.1 Case and Subject Selection

The first step in the study design regards measurement of the simulation models. We use all 71 Simulink models available at the department of the company described in Section 5.1.3. Each model represents one real ECU in the vehicle. All historic versions of these models covering a period of four years are stored in over 65,000 revisions, from which about 5,000 revisions concern direct changes to the models.

For the interviews, engineers were selected who are able to substantially assess the models. Therefore, the engineers were selected using department managers as proxies to help identifying them. All engineers responsible for software development and testing of one or more of the models were interviewed. The selected engineers hold different responsibilities among the models. An engineer can either be in the role of a developer or a model lead for a given model. A developer is an engineer who has made functional adjustments to the model at least once; a model lead on the other hand has an overview over all activities among the model and decides for strategic development activities. All model leads perform development tasks as well. The responsibilities were extracted using a management overview sheet. The developers were extracted from the actual commit logs where each commit contains a unique identifier for the respective committer.

Four model leads and four model developers were identified. They were selected from eleven developers altogether. Three developers had less than one year of development experience and were excluded. The limit of one year experience was set because it takes time for new engineers to get confident with the models. In addition, recently employed engineers themselves were not comfortable enough to make substantiated statements about the models. It would also have been difficult for them to compare changes, which occurred lately in the model, to changes that happened in the past. The interviewees' modeling experience ranges from two to seven years, with an average of a little less than five years.

5.3.2 Data Collection Procedure

Following our study design, the data collection starts with measuring size and complexity.

5.3.2.1 Measurement

Two size and two complexity measurements as explained in Section 5.2 were performed on all revisions of the 71 simulation models.

Firstly, a script retrieved a revision from the central repository followed by selecting the Simulink model. All four measurements were performed offline directly on the Simulink model files, which contain the model information in a structured format. Figure 5.3 shows the graph for the collected data of all revisions and all measurements for one exemplified simulation model. The figure shows four time series with the revisions numbers on the shared x-axis and measurement results on the four y-axes covering a duration of almost four years.

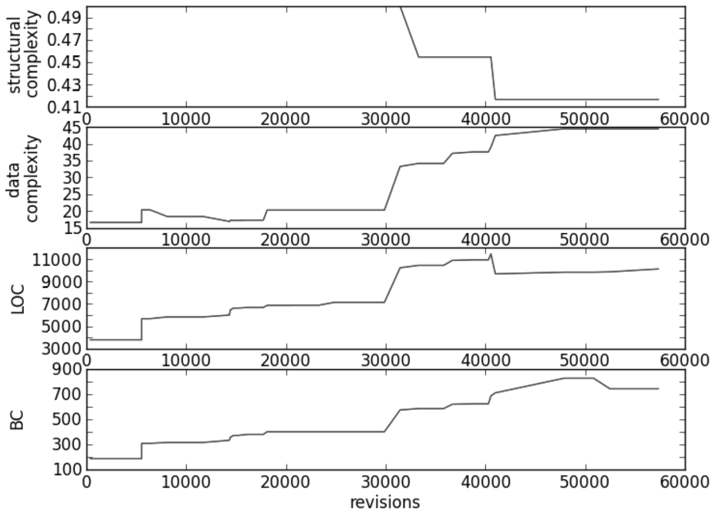


Figure 5.3: Example of measurement result for one anonymized model. It is illustrating a possible data set for the subsequent outlier detection. With revision numbers on the x-axis and measurement values on the y-axes covering a duration of almost four years.

We consider the size metrics that we used as reliable as they base on just counting lines or blocks; the complexity metrics on the other hand require the model to be parsed and interpreted. Thus, these metrics are susceptible to bugs in the models: For example, links in Simulink models without a source or destination block lead to parsing errors. Simple parsing errors could be fixed easily by adjusting in the model, but this is not possible for all revisions in general. Therefore, some measurements resulted in missing values for complexity and lead to a reduced sample size by 12% compared to the size measurements. We do not expect a big influence, because of the availability of the size measurements for respective revisions and the large sample size in general. The result of this step is a list of revisions and four measurements for each revision. Hence, we receive a data set represented by four time series of measurement results as previously seen in Figure 5.3, for all 71 simulation models.

5.3.2.2 Outlier Detection

From the resulting 71 data sets, a test and a validation set is manually created by random. The 35 data sets received as test set are used for all the following analyses and the remaining validation set is used at the very end of our study. During the preliminary analysis, nine simulation models are discovered to be legacy. These models were not updated in the last two years and are removed from the test set before starting outlier detection, as insights gained from outliers in these models might skew the results. Additionally, it is more difficult for engineers to assess models updated more than two years ago. Furthermore, we excluded one model that did not contain any functionality as it is considered as a future extension and thus, not implemented yet. In total, we performed outlier analysis on the test set containing 25 simulation models.

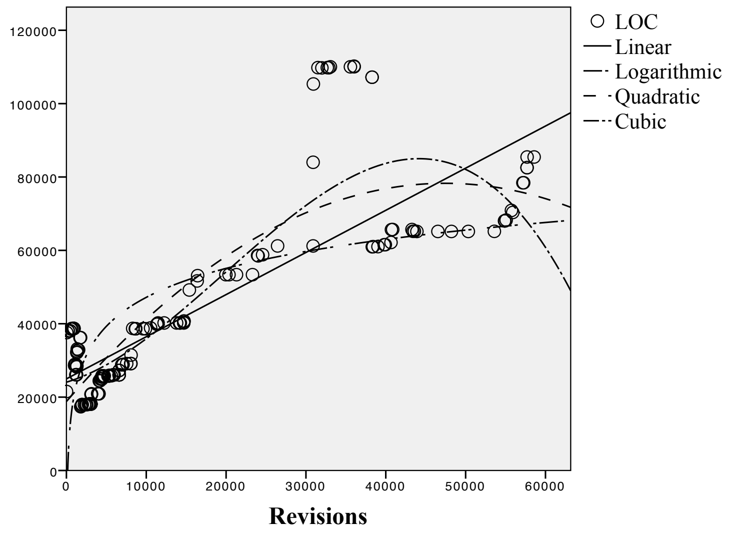


Figure 5.4: Common models fitted to example LOC measurement data.

We also excluded revisions where the actual model files were not changed. In many revisions, tooling, parameters, or similar adjustments where made, which do not affect the model itself. These revisions were excluded from the evaluation, leaving 2,188 revisions for the further analysis.

The outlier detection in this study is conducted in two steps. First, a model is fitted to each time series in the measurement data set. Second, the differences between original data and fitted model are compared. For the first step we began to fit common models, like linear, quadratic, cubic, and logarithmic models. We found that all of them have a common problem fitting our data. For example, if there is a strong increase in lines of code from one revision to the next one, shaping an edge, followed by a series of minor changes, we would be mostly interested in the largely rising edge in the beginning, as there must have happened something influential. Figure 5.4 shows that common models fail in this case. The measured time series represents LOC observations of one exemplified model and is highlighted as scattered circles. When comparing the fitted model and data values, it can be seen that many values following the peak at revision 32000 would count as outliers as well, although the measurement values change only little after the peak.

We decided to use ARIMA models instead and found that a model generated by an average from two prior values MA(2) detects the rising or falling edges of interest in the data robustly. Figure 5.5 shows a MA(2) model fitted to the same data set as used before. Comparing the MA(2) model with the measurement data of all models in the test set results in a list of values representing the differences of the fitted model and data values for our four measurements. Using the same model fitting approach on all simulation models might result in models that do not fit all data perfectly and would in this case lead to detecting more outliers. This reduction of the precision of the outlier detection is acceptable in this study, as the MA(2) model generally adjusted quickly to our data and more outliers for the analysis are not a drawback. We had to limit the MA value to a low number, as we are mainly interested in rising or

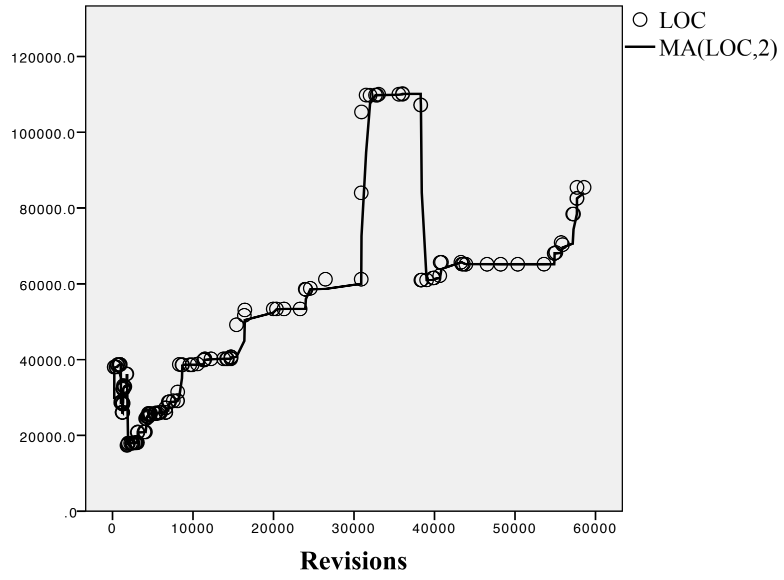


Figure 5.5: MA(2) model fitted to example LOC measurement data

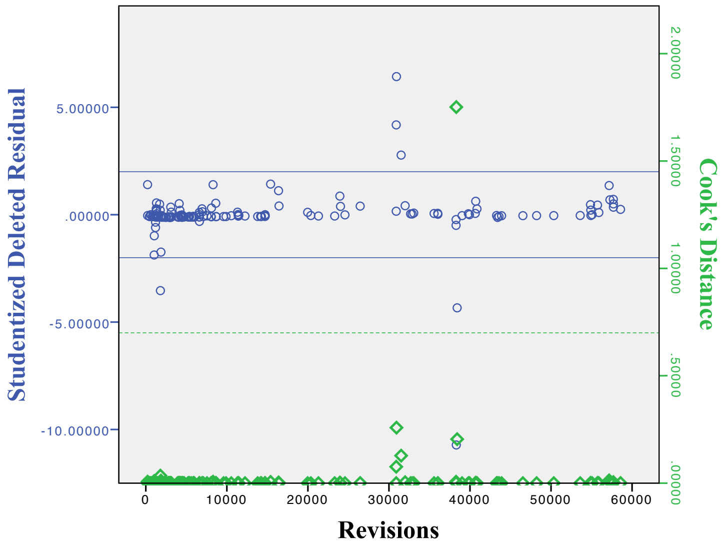


Figure 5.6: SDR and Cook's D for the example model fit in Figure 5.5

falling edges. High MA values would produce more noise among the calculated outliers.

Statistics provide multiple different approaches comparing a fitted model with real data. Typical approaches are for example, Cook's distance value, leverage values, and different kinds of residual values. We decided to use two different approaches, studentized deleted residuals (SDR) and Cook's Distance (Cook's D) as they are both common approaches to analyze residuals. Figure 5.6 shows residuals as they are received during the analysis for the MA(2) model fit to the example data shown in Figure 5.5. Blue circles represent SDR and green squares Cook's D. Outlier thresholds are calculated as described in Section 5.2.2.2 and visualized as blue lines for SDR and a green dashed line for Cook's D. It can be seen, that Cook's D is more rigorous than SDR and results in less outliers. When using SDR, more outliers are produced, which in turn result in more noise. For example, SDR produces high outlier values, even for small changes if there are only few changes within the model. As, our first intention was to use Cook's D only, we also included SDR to extend the set of outliers to be discussed with the engineers in our qualitative analysis.

Outliers received from Cook's D were directly usable. For results received from SDR, absolute values had to be calculated, as negative outliers result in negative values. This would lead to wrong correlations, as outlier impact was evaluated only positively by the engineers. Eventually, these two approaches result in two severity assessments of the outlying revisions in each of the four measurements. Thus, we could determine for every revision how outlying it is in respect to each of the four measurements. Hence, it is possible to rank the revision by severity on a interval scale. As there are SDR values and Cook's D values for each of the four measurements, we get eight lists of calculated outliers. This statistical approach is intended as a tool to provide ranked outlying observations automatically, based on the raw data.

5.3.2.3 Interviews

The ranked list of outliers provided quantitative data. In the next step we complemented it with qualitative data by conducting expert interviews. In interviews, engineers were asked to assess the impact of each revision on six software quality categories. The categories were taken from the ISO/IEC standard 9126 [16]. The interviews were conducted in a semi-structured manner, following the guidelines from Shull et al. [46]. Semi-structured interviews were chosen to not limit the stakeholders in starting discussions or providing additional insights. The interviews were performed individually by one researcher on-site and took 60 to 90 minutes. To avoid bias through communication between the engineers, interviews on the same models are conducted consecutively. To assess each revision, the engineers were allowed to use all information they required, including commit logs, personal notes, and source files. Hand-written notes were taken as audio recordings were not allowed on the company's premises. The questions used for all engineers were identical. For each outlying revision, we asked:

[A] What happened?

[B] What was the reason for the changes in that revision?

[C] On a scale from 1 to 5 (1 - no impact, 2 - low impact, 3 - average impact, 4 - higher impact, 5 - strong impact), how severe do you estimate the impact the revision had on:

   (a) Functionality
   (b) Reliability
   (c) Usability
   (d) Efficiency

(e) Maintainability

(f) Portability

[D] Are there revisions related to this one; in this or other simulation models?

[E] Did we miss a revision, which you think had a strong influence?

Asking for the impact on the software quality in that way forces estimations relative to the respective simulation model. It is not possible to compare impact between two models. This is not a restriction as the outlier measurements are calculated relative to each respective model as well.

We asked each engineer for which models they felt comfortable to make an assessment of impact. It became obvious in the interviews that investigating revisions from more than two years ago involved mainly guessing. Thus, revisions older than two year were not discussed if the engineer did not feel comfortably enough with providing reliable information.

For each model, we aimed for at least two engineers to be interviewed. In about 50% of the cases it turned out being not possible as, for example, developers have left the department.

The answers for the first two questions were grouped into similar keywords in order to make them comparable. After this grouping, the answers from the first three questions on each revision were directly comparable to the quantitative results received from the measurements. Thus, for each detected outlier we have a quantitative impact from the measurements and a qualitative assessment of description, reason, and impact on software quality.

Additionally, the interviews are used to determine the performance of the outlier detection. We calculated precision and recall based on the interview results. An outlier in a measurement is not valid, if one or more engineers evaluated them with none or only marginal impact in all quality categories. Additionally, recall is calculated using the fifth question, which asks for missed revisions that might also have a strong influence as well.

5.3.3 Analysis Procedure

We receive eight lists of calculated outliers from the measurements and six Likert-based quality impact assessments and descriptive information (description and reason) from the interviews. Both, measurement data and interview data is relative to each model. Table 5.1 visualizes how the collected results are compared. Each outlying revision has eight calculated outlier values (R_{loc} to C_{sc}) and at least six impacts on quality (I_F to I_P) assessed by one ore more engineers. Additionally, there are descriptions and reasons for each outlier. Comparing the measured outliers, which are assessed to be on an interval scale with ordinal scale, Likert-based impact assessments requires non-parametric correlation analysis; therefore, we calculated the Spearman correlation coefficient. All calculations were conducted in SPSS and R.

With both data sets being comparable, we are able to perform multiple analyses on:

[A] correlation between raw outlier data and impacts;

[B] correlation between combinations of outlier data and impact;

Table 5.1: Illustration of how results were compared. For each revision in each model, measurement outliers (R for residuals, C for Cook's D) are juxtaposed with the impacts (i) on software quality categories from ISO 9126 [16], descriptions, and reasons, received by the engineers (Eng.).

		Measurements (interval scales)								Interviews (ordinal/nominal scales)								
Model	Rev.	R_{loc}	C_{loc}	R_{bc}	C_{bc}	R_{dc}	C_{dc}	R_{sc}	C_{sc}	Eng.	I_F	I_R	I_E	I_U	I_M	I_P	Desc.	Reas.
	1	r	c	r	c	r	c	r	c	E1	i	i	i	i	i	i	d	r
										E2	i	i	i	i	i	i	d	r
M1	2	r	c	r	c	r	c	r	c	E1	i	i	i	i	i	i	d	r
										E2	i	i	i	i	i	i	d	r
	3	r	c	r	c	r	c	r	c	E1	i	i	i	i	i	i	d	r
										E2	i	i	i	i	i	i	d	r
	4	r	c	r	c	r	c	r	c	E3	i	i	i	i	i	i	d	r
M2	5	r	c	r	c	r	c	r	c	E3	i	i	i	i	i	i	d	r
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

[C] correlations between principal components among the outlier data and impact;

[D] dependencies between outliers and descriptions and reasons for the revision provided by the engineers; and

[E] similarities between results received from size and complexity measurements.

Consecutive workshops were conducted with the four lead engineers. The intention of the workshops was to confirm and validate the unveiled evidence and evaluate the methods. This step shall ensure that feedback captured from stakeholders remained consistent and that conclusions were drawn correctly. To achieve that, results and theories received from the analysis are discussed. Furthermore, results received from measurements and interviews are combined in a prediction model, which is applied to the validation set. Thus, the most likely outliers in the validation set and their most likely impact are presented to the engineers for confirmation.

5.3.4 Validity Procedure

This study is susceptible to bias as we intended to not restrict the study to one single focus and instead accept the possibility of multiple outcomes. Hence, it is important to follow rigorous approaches in all steps and to employ a strict validity procedure. A key aspect for the study's validity is using triangulation, achieved by:

- Multiple measurements and outlier detection methods;
- Combination of stakeholder feedback and measurement values;
- Combination of interviews and workshops; and
- Multiple engineers per model and revision, if possible.

Two researchers were working on-site to avoid researcher bias and to ensure validity by continuous discussions on intermediate results. Two researchers worked remotely and preserved an outside view on the methodology and the conclusions drawn from the results. By reporting results back to the industrial partners in the workshops, bias was reduced and findings could be verified.

5.4 Results

Based on the four measurements, we found 221 outliers in 2,188 revisions from 25 simulation models. That means, on average, we analyzed 88 revisions per simulation model, with the minimum at 23 and the maximum at 338. Among them, on average, we found 9 outliers per model; at least 2 and not more than 30 altogether. For the 221 outlying revisions, the engineers could substantially evaluate 139 thereof as early revisions were excluded as pointed out in Section 5.3.2.2. On these 139 data sets, correlation observations were performed. Table 5.2 shows correlation results using Spearman's correlation coefficient. For

Table 5.2: Spearman's ρ correlations between calculated outliers and software quality characteristics. (SC – structural complexity, DC – data complexity, LOC – lines of code, BC – block count, COOK – Cook's D, SDR – studentized deleted residuals)

		Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
SC_COOK	Corr. Coef.	0.037	0.208*	0.063	0.112	0.124	0.087
	N	118	118	118	118	118	118
DC_COOK	Corr. Coef.	0.105	0.058	0.172	0.158	0.176	0.199*
	N	120	120	120	120	120	120
LOC_COOK	Corr. Coef.	0.180*	0.005	0.270**	0.230**	0.210*	0.147
	N	138	137	138	138	138	138
BC_COOK	Corr. Coef.	0.341**	0.093	0.353**	0.203*	0.266**	0.104
	N	138	137	138	138	138	138
SC_SDR	Corr. Coef.	0.092	0.182*	0.082	0.206*	0.158	0.135
	N	118	118	118	118	118	118
DC_SDR	Corr. Coef.	0.215*	0.044	0.191*	0.315**	0.234*	0.220*
	N	120	120	120	120	120	120
LOC_SDR	Corr. Coef.	0.310**	0.009	0.261**	0.392**	0.242**	0.140
	N	138	137	138	138	138	138
BC_SDR	Corr. Coef.	0.454**	0.136	0.389**	0.392**	0.334**	0.152
	N	138	137	138	138	138	138

Table 5.3: Component matrix showing PCA results. The abbreviations are explained in Table 5.2.

	Component	
	1	2
LOC_COOK	0.872	-0.166
DC_COOK	0.852	0.038
LOC_SDR	0.845	-0.151
BC_SDR	0.806	-0.286
DC_SDR	0.781	0.164
BC_COOK	0.728	-0.348
SC_COOK	0.249	0.785
SC_SDR	0.542	0.701

Table 5.4: Descriptions and Reasons for anomalies.

Description	%	Reason	%
Logic/Functional	50.8%	Requirement	83.1%
Architectural	15.3%	Maintenance	5.9%
Interfaces	11.9%	Bug Fixes	1.7%
Combinations	13.5%	Combinations	5.9%
No Description	8.5%	No Reason	3.4%

visibility reasons, the p-values are not displayed but significant values at 0.01 level are marked with two asterisks and values at 0.05 level with one.

A factor analysis revealed that two principal components extracted from the eight outlier calculations explain 73% of the variance among them. The analysis in Table 5.3 shows that the first component explains outliers from both size and data complexity measurements. The second component explains outliers received from the measurement of structural complexity. Finally, we evaluated the dependencies between outliers and descriptions and reasons. Results are summarized in Table 5.4. This evaluation provides insights on what types of anomalies our approach is able to unveil. The descriptions on outlying revisions collected from the engineers were summarized to “changes in the model logic”, “changes in the model architecture”, “changes in model interfaces”, and combinations thereof. Architecture changes refer to changing the model structure, for example splitting up a block in two. Interface changes relate to input and output signals, like adding signals to a bus. The majority of changes were modifications to the model logic, with 50.8% of the descriptions. Architecture and interface changes were mentioned 15.3% and 11.9% of the time, respectively. No description was given for 8.5% of the revisions. Combinations of descriptions cover the remaining 13.5%.

Reasons for abnormal changes in the software as provided by the engineers were “requirement”, “maintenance”, “bug fixing”, as well as combinations thereof. With 83.1%, requirements were the most common reason for a change. Maintenance was mentioned in 5.9% of the cases. Bug fixing was a reason for change only in 1.7% of the cases. It occurred more often in combination with requirements (3.4%). No reason was given in 3.4% of the cases and the remaining 2.5% were combinations thereof.

5.5 Analysis of Results

In order to answer the research questions, we analyzed the received results. Research questions 2 and 3, on which kind of outliers are detected and how they correlate with domain expert assessments can directly be answered by the results

First, to address research question 2, we analyzed the dependencies between impact, reasons, and descriptions. The results show that anomalies from size and complexity measurements reveal mostly changes in the model logic based on requirements. 50.8% of the anomalies could be related to logic adjustments. 83.1% of the anomalies are due to requirements. Hence, the presented approach is able to detect abnormal events based on requirements, which mostly result in functional changes in the simulation models. Activities like architectural changes and interface adjustments cause only few anomalies in our study. Respectively, maintenance and bug-fixing activities are not related to the abnormal events detected. Multinomial logistic regressions were performed, to reveal dependencies between the different anomalies, their reasons, and descriptions. No significant dependencies could be determined. That means, that no single outlier value has the ability to predict what kind of change has occurred or the reason of it.

Second, addressing research question 3, the analysis of the correlations in Table 5.2 showed:

- We did not encounter unexpected behavior among the correlations, as there are no negative correlations between measured outliers and the engineers impact assessment.
- The strongest correlation was observed between BC SDR and functionality. Also Cook's D for BC shows weak correlations, supporting the finding that outliers from the block count measurement overlap with the engineers feedback on functional changes. Also LOC SDR shows weak correlation. Both observations were expected, as size measurements are often used to assess software functionality.
- The engineers feedback on efficiency matches with three outlier values, DC, LOC, and BC. This is the only quality characteristic, showing correlations with data complexity outliers.
- The only outlier values that correlate with maintainability are the SDR values for the BC metric.
- For usability both BC outlier values correlate.
- SC outliers do not correlate and thus, it might not be a good metric to measure software quality.
- None of the outliers showed the ability to detect reliability or portability. SC has a weak tendency to explain reliability, however.

- None of the correlations is strong.

All mentioned correlations are significant with a p-value below 0.01. Based on the correlation results, we can relate anomalies found with our approach to different quality attributes of the simulation models. We can therefor detect noticeable changes in functionality, usability, efficiency, and maintainability as they occur. Furthermore, it is possible to investigate past events with high impact on those attributes.

The principal component analysis results in Table 5.3 show that two of eight components explain most of the variance. This behavior was expected, as size and complexity measurements have been found to correlate before (cf. [77]). The first component comprises both size and the data complexity measurements. It correlates with the same qualities as the single measurements it is created from, which suggests combining of those measurements into one component. The second component comprises the SC outliers and correlates with none of the qualities but there is an indication for correlation with reliability, again. That means that outliers from the structural complexity metric differ from the rest. Additional analysis could reveal it as indicator for reliability.

To address research question 1 and 4, on the applicability of the approach and meaningfulness of the results, we investigated the capabilities of the outlier detection approach. In general, if the approach is once determined, most of the steps can be automated. For example, the data extraction from version control, the measurement, the outlier detection and the correlation analysis were fully or partly automated. Hence, the effort required to perform repeated assessments of the models is moderate. By utilizing the interview data, as described in Section 5.3.2.3, we could calculate precision and recall to investigate the quality of the outlier detection. Twelve of the 139 revisions were evaluated with no impact and 18 more with only minor impact. That means, based on the interviews, the precision of our outlier detection is between 91.36%. and 78.42%. During the interviews, seven revisions having a high impact were mentioned to be missed by our approach. Considering 109 true positives from 139 revisions, we achieve a recall of 93.97 percent, based on engineer opinions.

Additionally, to show the applicability of the approach we interpreted the results of the consecutive stakeholder workshops. The weak correlations and the anomaly detection approach was discussed. We concluded together with the stakeholders that in industry more supervised approaches could lead to better results. The scientific thinking of collecting all possible revisions on all possible models yields observations on general model behavior in the domain. According to the engineers, the following two steps might improve results for more specific contexts:

- Considering only models, which are constantly maintained and extended, as less frequently used models skew the prediction capabilities of the results.
- Consulting only lead engineers with the evaluation of detected outliers. Less experienced stakeholders might misinterpret impact among the complex model environment.

These insights indicate, that data received in this industry domain would benefit from a supervised approach, but results are then limited to certain domain contexts.

Summarized, the data analysis showed that our approach can detect abnormal events in software measurement results. Applying it to software size and complexity metrics in industry unveils abnormal strong increasing or decreasing changes, mostly functional in nature and based on requirements. Those events were found to have a high impact on different software quality attributes. This enables early detection and past analysis of events with strong impact on functionality, maintainability, efficiency, and usability.

5.6 Threats to Validity

We split threats to validity according to categories from Feldt & Magazinius [43]. Regarding **external validity and generalizability**, the results and analysis are limited to model-based software in the automotive domain. Results are extracted from this domain and are therefore only transferable to a similar one. Nevertheless, as the measurements and statistical approaches are common and used in other domains as well, we see no reason why similar results should not be received from Simulink models in other domains. The sample of eight engineers limits the generalizability as well. A larger sample would have increased generalizability even in the same domain. Still, the selected engineers are strongly familiar with the models and experienced in the field. Their assessments is expected to represent experienced software developers in the domain.

Concerning **construct and internal validity**, there are also threats to be considered. The interviews have a high impact on the outcomes. At the same time, engineers' recall is not perfect and they might have biased opinions on revisions, their impact, and reasons therefor. Additionally, not all engineers were still available at the studied company. We mitigate the threat of human bias with triangulation, by complementing the interviews with consecutive workshops and asking as many engineers as possible for the same analyzed revision. Still, the bias cannot be eliminated completely, as the same lead engineers were participating in the workshops.

There is also the threat that engineers evaluate differently among each other or perceive quality differently. This threat is mitigated by clear and consistent explanation of scales and quality. Thanks to the discussions throughout the interviews, additional misunderstandings could be resolved. Strictly structured interviews would not have allowed for that; on the other hand, discussions in the interviews might have biased interviewees. While avoiding biasing comments in the interviews, we accepted this to happen as discussions ensure that the topic was understood and revealed insights not covered by the questions otherwise.

As mentioned before, the complexity metrics resulted in missing values for some revisions, as bugs prevented proper parsing and measurement. If larger changes happened in these missing revisions, we cannot be sure if outliers measured afterwards happened during or after the missing values. This reduces the reliability of the two complexity measurements.

Creating correlation coefficients bases on different engineers feedback can lead to skewness in the results and affect the study's **conclusion validity**. The ratings on the Likert scale and the meaning of the software quality categories have to be clear to them. As mentioned before, to mitigate this threat a detailed introduction about software quality and also about the scale was provided in the beginning of the interview. Misunderstandings throughout the interviews were clarified. The limited amount of stakeholders interviewed could skew the received findings and the conclusions drawn therefrom and therefor affect the conclusions as well. More studies in the similar domains are necessary to mitigate this threat.

5.7 Conclusions and Future Work

The main goal of this study was to improve model validation and to investigate model quality by applying statistical outlier detection methods to model revisions quantified by software metrics and evaluate them using expert knowledge.

Our results show that unveiling outliers using quantitative instruments is in general leading to reliable discoveries. Considering stakeholder assessments, we found meaningful outliers with a high precision and related them to the software quality characteristics used in this study. Additionally, we can differentiate well between important and unimportant revisions. That in turn supports model validation by assessing and predicting current, past, and future model quality. We further found which activities and reasons cause respective outlying revisions. Together with the previous findings and the fact that most parts of the approach can be automated, we conclude that the general approach is very well applicable in the studied domain.

The study provides further opportunities for future work:

- The possibility to adjusting outlier thresholds together with already collected impact evaluations enables the potential for optimizing thresholds to fit the data better.
- Further metrics and adjusted models could reveal further details on measurable qualities. For example, efficiency metrics could additionally be integrated with the existing approach.
- Analyzing only specific time frames instead of the whole data set might help reducing noise created by time frames not interesting in specific contexts. For example, disregarding the beginning of a project or specific maintenance phases might result in different observations.

Chapter 6

Paper E

Prediction of Software Model Growth in Practice

J. Schroeder, C. Berger, A. Knauss, H. Preenja, M. Ali, M. Staron, T. Herpel

In submission to the 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, May, 20-28, 2017.

Abstract

The size of a software artifact naturally has an influence on software quality and its development process. For example, an unforeseen demand for refactoring of an artifact that exceeds a certain size threshold can significantly slow down a project in practice. Predicting the time when the size grows to the level where maintenance is needed can avoid unexpected efforts and help to highlight problematic artifacts earlier.

The amount of prediction approaches in literature is vast. However, it is unclear how well they fit with prerequisites and demands common in practice. We investigate this problem by performing an industrial case study in the automotive industry. In a first step, we elicit requirements towards predictions in practice using a survey and stakeholder workshops. We then measure software size of 48 software artifacts throughout four years of revision history resulting in 4,668 data points. In the last step, this data is used to assess the applicability of state-of-the-art prediction approaches found in literature including methods based on machine learning by systematically analyzing how well they fulfill the requirements elicited in step one.

This quantitative and qualitative empirical analysis revealed strengths and weaknesses of the different approaches in practice. We found, that the approaches provide significantly different results.

6.1 Introduction

Software development in the automotive industry is facing steadily growing size and complexity among its artifacts. For example at Volvo Cars in Sweden, the amount of software in cars has increased exponentially in the last twenty years: In 2006, vehicles contained 10.9MB of code, in 2011 around 117.5MB, and in 2014 already 917MB [99]. Other car companies have seen similar developments and according to Wyman [100], the entire sector is facing an increase in technological complexity. Concerning the amount of software and tests being introduced with autonomous vehicles, this upwards trend is not going to slow down any time soon.

In practice, software exceeding size limits set by hardware or software requirements causes long loading, build, and deployment times. Hence, while software grows in size and complexity, situations arise where refactoring becomes necessary. When these refactorings occur unexpectedly and become inevitable, they can slow down or block whole development cycles causing delay and result in increased development costs. Reliable predictions of software status and quality can prevent such issues by foreseeing problematic projects or artifacts to improve release planning, for example. Extensive research on predictions in other fields like software fault prediction shows the need for knowledge about the future development of software artifacts. For example, Catal and Diri [101] report on the increasing trend in the amount of prediction papers in this field.

Predicting software size can be done by assessing software growth information collected from past software development. By measuring size of an artifact over multiple past software revisions, time series data can be created, which shows the artifacts' quality development throughout the whole software life cycle. For predicting time series data, many approaches exist and already in 1970, Box and Jenkins [89] presented approaches to analyze and predict time series that are used by stock market or meteorology time series analysis, for example. More recently, also machine learning approaches found their way to prediction of time series data [102]. They have been found to outperform classical approaches regarding prediction accuracy in some application domains [103]. However, it is still unclear how well such approaches perform in a real world, industrial setting where additional factors affecting their practicability need to be considered like time needed for training predictors or their maintainability. Empirical evidence is needed to show which approaches are applicable to data actually gathered from realistic scenarios in practice.

6.1.1 Research Goal and Research Questions

In this study, we investigate the applicability of prediction approaches in practice. We perform an industrial case study on model-based software in the automotive domain. The following research questions contribute to the goal for our study.

RQ1 To what extent are prediction approaches applicable in practice?

RQ1.1 Which expectations do practitioners have on prediction of software size in practice?

RQ1.2 What are most important prediction approaches for time series data that are mentioned in the literature?

RQ1.3 How do different prediction approaches perform regarding the criteria elicited in RQ1.1?

6.1.2 Contributions

This paper contributes by comparing the performance of multiple prediction approaches in an industrial context in the domain of automotive software engineering. Our comparison includes traditional statistical approaches next to modern machine learning approaches to provide empirical evidence about their respective performance. We also extract lessons learned on strengths and weaknesses of the used approaches to provide practitioners with evidence on which approach is suitable in similar contexts. We found that the results received from applying the approaches in practice vary. We highlight strengths and weaknesses among the approaches and give guidance on which ones to use in practice.

6.1.3 Paper Structure

The rest of the paper is structured as follows: We introduce background knowledge for the domain where we conducted our study in Section 6.2; additionally, we cover related work and show similarities and differences to our findings. In Section 6.3, we outline our systematic approach to address the research questions. Following the methodology, we present our results, analyze, and discuss them in Section 6.4. Section 6.5 concludes the research and presents future work.

6.2 Background and Related Work

6.2.1 Measurements

The increasing size and complexity of software can cause severe problems especially in systems that are limited by the underlying hardware, for example, in embedded systems. We assess these factors in this study by measuring lines of code (LOC) and block count (BC) as our previous study showed that they are related to maintainability issues and software complexity [23]. Furthermore, they turned out to outperform cohesion and coupling measurements in their ability to assess model complexity in practice. Both measurements can be considered as static code analysis as they do not require the models to run to provide results. Hence, they work even if syntax errors exist in the model. This kept data preprocessing to a minimum and avoided unnecessary transformations or interpolation of missing values, which could skew the data unintentionally.

To calculate the LOC metric in this study, the .mdl files of Simulink models are assessed by counting each line in the XML-like representation of the respective model. They do not contain comments or similar non-code related entities. The BC metric is counting blocks using the Matlab/Simulink environment. The built-in function *sldiagnostics* is used that counts all blocks

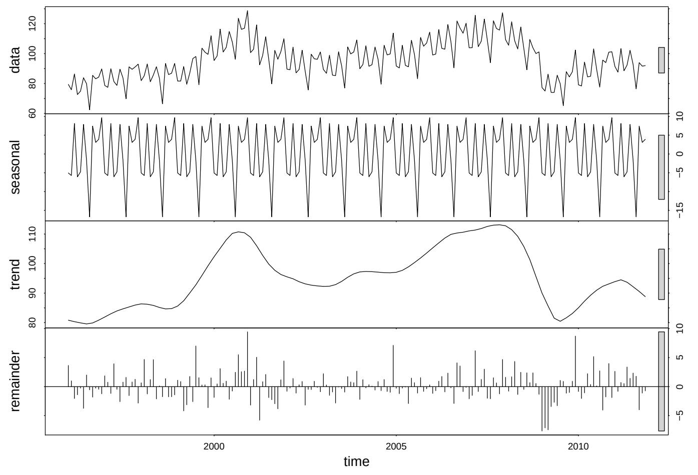


Figure 6.1: Example for a time series decomposition done in R using the command *stl* on example data showing manufacturing numbers per month.

contained in a model. The function considers all blocks in the model, even masked blocks on the lowest layers.

6.2.2 Time Series Data and Analysis

All approaches presented in this paper are applied to time series data. Time series are sequences of observations collected over time, usually in equidistant time intervals. Time series can have different properties such as trend, seasonality, and random behavior. In Figure 6.1, time series properties are shown using example data. In this example data, trend, monthly seasonality, and remaining random aspects can be clearly extracted. In measurement data received from our case not all aspects are as easily visible as in the example data. Tools like autocorrelation functions (e.g., *acf* and *pacf* in the statistics environment R) can reveal seasonality and trends can be made visible using exponential smoothing (*ses* in R) for example.

6.2.3 Prediction and Evaluation

Prediction can be categorized into classification and regression. In classification, the goal is to assign and learn classes to a set of input values and predict these classes for each new input value. Regression on the other hand aims for learning the values of some input data and predicting a new value for new or unknown input data. An example for classification is predicting nominal categories like $(requirement, feature, bug)$ for a set of natural language-based issue tracker data. A set of issues would be learned by the algorithm and a new issue would be predicted to be in one of the three categories. Predicting regressions usually regards an interval or ratio scale like the development of sales over time. Learning sales data of the past enables a prediction of the sales in the future. In this study, we aim for predicting future values of a time series based on previous values of the same time series.

Respectively, the approaches for evaluating the performance of predictions differ in classification and regression problems. Measures like F1-measure, Precision, Recall, etc. work well in classification. Contrary to classification problems, the performance of regressions problems is assessed how close they approximate a real value. Hence, the data is usually split in one learning and one test set. Predictions are then made based on the learning set and compared with the test set. An error measurement is applied to evaluate accuracy of the predictions.

6.2.4 Context

This case study is conducted at a testing department of a German premium automobile manufacturer. The company produces approximately two million vehicles per year. In the automotive domain multiple software projects are combined to create the software of a vehicle. Resulting artifacts from these projects are usually electronic control units (ECUs) to be installed into a vehicle. Simulations of ECUs are used during testing to replace unfinished real ECUs. This enables the emulation of a real car environment for ECUs under test. The simulations usually run on several real-time computers to meet the computational requirements.

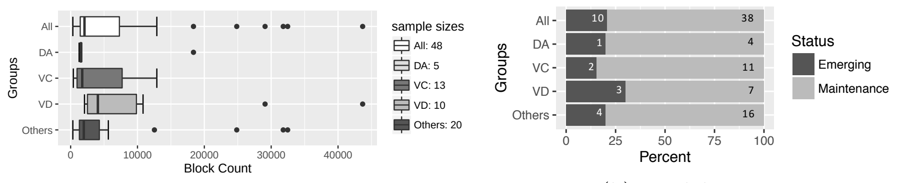
At the department where this study is conducted all simulation models needed to simulate a complete vehicle were made accessible to the researchers for analysis. The models are realized with Matlab/Simulink and multiple development teams are responsible for separate groups of ECUs with similar functionality. Detailed information about the simulation models can not be disclosed, but in order to understand the distribution of size and attributes among them, an overview is provided in Figure 6.2. The figure shows the different groups of simulation models, namely driver assistance (DA), vehicle control (VC), vehicle dynamics (VD), and others. They are communicating using five different bus systems. Figure 6.2a lists the current model sizes within the groups and the whole data set, calculated with the block count metric (BC).

The sizes of the majority of the models range between 331 and 18,376 blocks, with seven positive outlying models. The strongest outlier with 107,857 blocks within the others-group is not shown in the figure due to visibility reasons. The status in Figure 6.2b visualizes how often the models are extended. A model is evaluated as being in software maintenance if the amount of work on this models is decreasing over its lifetime. Currently, these models experience only little evolution, mostly restricted to adding or removing signals from a vehicle bus to meet updated specifications. Emerging models experience many updates in recent time and are in the focus of the developers at the moment.

6.2.5 Similar Studies

In this study, we combine research from three established fields. The field of data mining and prediction, the field of mining software repositories, and the field of measurements.

Many existing studies assess different data mining approaches. Malhotra [103] studied 64 publications regarding application of machine learning



(a) Model size

(b) Model status

Figure 6.2: Overview over amount, size (BC), and status of the models used in this study. The groups are driver assistance models (DA), vehicle control models (VC), vehicle dynamics models (VD), and other models

techniques for software fault prediction. Malhotra found that 19 out of 64 studies involved a comparison element between statistical and machine learning methods. The results demonstrate that the machine learning approaches mostly outperform statistical linear approaches. The author identified three frequently used machine learning approaches for software fault prediction: 1) decision trees, 2) neural networks, and 3) support vector machines. Malhotra's results are primarily limited to software fault prediction. Fu [102] performed a literature review on prediction approaches as well. The author provides a comprehensive overview of existing techniques and classifies them according to their application. We use the two literature reviews as a main source of information to answer RQ1.2. Nevertheless, they are lacking concrete, in-depth evaluations of the applicability of the approaches in practice.

Lastly, Martínez-Álvarez et al. [104] also review recent work on time series prediction. They split results in linear statistical and non-linear machine learning approaches. We follow their notation in this study. They list multiple prediction approaches and error measurements currently used in literature. Their study however is specifically designed for electricity-related time series.

Our study is related to the field of mining software repositories. Hence, the work published within the MSR community is relevant. Software repositories are used to extract different kinds of information. Robillard et al. provide an overview of distinct recommendation systems using software repository data [105]. Many studies focus on predicting defects, like in Zimmermann [106]. Other studies have used software repositories to investigate refactoring practices (cf. [107]). However, we know of no other study combining the aspects of software repository mining and measurements with prediction approaches, in practice.

Size measurement is well studied. Studies investigating software size have shown that size can be used to assess productivity [108] and defects [109] in practice. Schröder et al. [23] have previously shown how valuable simple metrics like lines of code can be, to assess software complexity and maintainability in practice.

6.3 Methodology

We perform a case study, in which we observe and investigate a specific case in a real world context [34] while avoiding interventions of researchers with the case [36]. We follow Runeson and Höst's guidelines on designing,

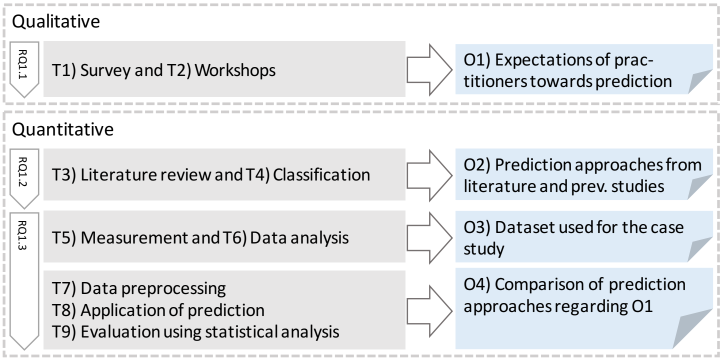

Qualitative
RQ1.1
T1) Survey and T2) Workshops
O1) Expectations of practitioners towards prediction
Quantitative
RQ1.2
T3) Literature review and T4) Classification
O2) Prediction approaches from literature and prev. studies
RQ1.3
T5) Measurement and T6) Data analysis
O3) Dataset used for the case study
T7) Data preprocessing
T8) Application of prediction
T9) Evaluation using statistical analysis
O4) Comparison of prediction approaches regarding O1

Figure 6.3: The study design consisting of a qualitative and a quantitative part. Tasks are presented in gray squared boxes, while the outcomes are depicted in blue boxes with bent corners. They are attributed to our research questions.

planning, conducting, and reporting the case study [38]. Our study comprises a quantitative and a qualitative part, which both consist of multiple tasks outlined in Figure 6.3. First, the case and the context are investigated qualitatively. By conducting a survey and regular industry workshops, the expectations of practitioners towards prediction of software size are elicited. They are the basis for the quantitative applicability investigations afterwards. Results of this step answer RQ1.1. The second part investigates quantitative aspects. We identify prediction approaches, apply them to measurement data, and evaluate their applicability in our context. This step results in a classified set of prediction approaches and their evaluation regarding applicability in practice, which answers RQ1.2 and RQ1.3. The steps are outlined in greater detail in the following sections.

6.3.1 Case and Subject Selection

The artifacts being assessed in this study are Simulink models simulating electronic control unit (ECU) functionality. All 70 models available at the department are considered for the data collection. Most models are still frequently updated with functionality or quality improvements; other models are only maintained on their interfaces to keep them usable in combination with the rest; a third group are legacy models which are not used anymore, and a last group is formed by newly created models with little past data. Legacy models could skew the results as they might not represent the current development practices. Models that are too new do not provide sufficient data for our analysis. For this study, only the first two groups were considered resulting in 48 models that provide a representative view on the development conducted for integration testing at the company. For the 48 models 4,668 revisions were assessed in total. This includes only revisions where one of the models' code was changed.

Six engineers participated in the industry workshops and the survey. The engineers have 3.8 years experience on average (1-7 years) and contribute as developers, testers, or team leaders. Hence, all existing roles present at the department are considered for the survey. All engineers are working on the shared set of models while having different development focus including driver assistance, vehicle dynamics, and general vehicle control. The subjects were picked by convenience from the group of engineers responsible for the models. All lead developers for the previously mentioned function groups were interviewed, as well as the respective team leader.

6.3.2 Data Collection and Analysis Procedure

Our study is comprised of several tasks (T) resulting in outputs (O), which are depicted in Figure 6.3 and outlined in detail below.

T1) Survey and T2) Workshops

The goal of the survey and workshops in this study is to complement the quantitative analysis with practical information from practitioners and to provide a qualitative view on software size prediction. The survey in particular has the purpose to collect the practitioners' expectations on predictions. After introducing the topic and emphasizing the focus towards predictions of software size, the following questions were asked.

[A] How long should a prediction take?

[B] How accurate should a prediction be?

[C] How many models should be predicted accurately?

[D] How far ahead should the prediction be accurate?

[E] How much maintenance effort is acceptable?

[F] What are additional important properties of a prediction?

[G] Rank the prediction properties by their importance: *maintenance, run time, accuracy, distance of predictions, additional properties*

The questions aim to assess the importance of prediction properties, but also on possible quantification of these properties by asking for thresholds. Quantification is achieved by assigning values to the ranks assigned in the last question, ranging from one to the number of properties discovered. Those values are counted and compared to determine the importance of the properties.

The industry workshops serve the two purposes of discussion and verification of the gathered results. Findings extracted from the anonymous survey were discussed and verified. The workshops were conducted without a prescribed structure to allow for discussions about the intermediate results and to find additional input like prediction properties of interest. This provides input for RQ1.1 as well as it validates the applicability of prediction approaches in practice.

T3) Literature Review and T4) Classification

We conducted a literature review to investigate the most important prediction approaches. We examined existing literature that focuses on algorithms and approaches used in the context of predicting time series data, as well as their implementation in software development. Hence, the search string was constructed by combining synonyms of the three keywords *prediction*, *time series*, and *data mining*. We focused on research applying prediction approaches. Optimization or in-depth performance analysis of existing approaches was out of scope of the review. The literature studies of Fu [102], Malhotra [103], and Martínez-Álvarez et al. [104] were used as main source for approaches, as they provide existing investigations and comparison.

In various domains both, linear and non-linear approaches are implemented for predicting. In the review, we identified the methods (linear or non-linear) used for predicting. For each of the identified methods, we investigated which approaches exactly were used in the selected articles. Based on this information, we identified the most common approaches that are implemented in the context of time series prediction. The results of the literature review are published in Preenja and Ali [110]

T5) Measurements and T6) Data Analysis

After the literature review the data set for the predictions under investigation has to be created. The two size metrics described in Section 6.2.1 are applied to all 4,668 revisions of the 48 software models. The collection of the measurement data was automated. A script went through each revision of a model and fetched the main model file to be measured from the model repository. This resulted in two measurement values, one for LOC and one for BC, for each revision and therefore two lists for each model, depicting the development of the metrics over time. On the resulting data set manual data interpretation and time series analysis are performed to understand how the data behaves in order to fit appropriate prediction approaches. This includes analysis of trend, seasonality, and outlying/random data. Trends in the data are estimated by plotting linear or polynomial regression lines. Seasonal dependencies are visualized using the auto-covariance an auto-correlation functions ACF and ACF in the statistics environment R. The analysis of the time series showed that they are mostly determined by their trend. All models grow in size. Seasonality in the growth of software size could not be detected coherently for the assessed series.

T7) Data Preprocessing

In order to evaluate the prediction accuracy using ground truth data, the data set consisting of measured models throughout revision history had to be split up. The received measurement data was divided into three sets: a training set, a validation set, and a test set. The training set was determined by our previous study [111], where we collected model data until the end of 2015. As this study continues the previous work, we use the data from the previous study as training set and started a second data collection in 2016 to gather the test and validation sets. The remaining data from 2016 was split equally in a test and a validation set. The resulting data set used in this case study is summarized in the Table 6.1.

Altogether, we collected data from 4,668 revisions in the times between 2013 and mid 2016. In this set, we found at least seven revisions and at most 440 per model. Additionally, Figure 6.4 shows the distribution of all revisions throughout the groups of functionality.

Table 6.1: Summary of the collected revision data.

	No. of revisions	Min rev.	Max rev.	Avg.	Period
All Revisions	4,668	7	440	88	1/2013-6/2016
Learning set	3,882	6	351	73	1/2013-12/2015
Validation set	376	1	44	7	1/2016-3/2016
Test set	410	0	44	7	4/2016-6/2016

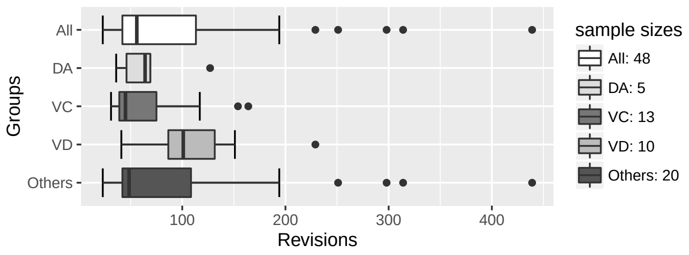


Figure 6.4: Amounts of model revisions available in this case study, grouped by functionality.

To perform a correct time series analysis according to [89] on this data set, time intervals between data points have to be equidistant. However, in the context of revisions and commits, this is not the case as commits, and consequently measurements are conducted whenever a developer decides to make changes to the project. There are multiple ways to address this problem. Eckner [112] presents an approach directly applicable to non-equidistant data and Rehfeld et al. present a resampling approach [113]. Both approaches are not widely applied yet. Analyzing time series without constant intervals, also called unevenly spaced time series or irregularly sampled time series, still requires further research.

A more intuitive and straight-forward approach in the case of measurement samples is data interpolation: a fixed time interval is chosen, for example daily intervals, and missing values are estimated. Data interpolation is applied widely but is not without critique as it can skew the data set as it cannot be ensured that in between two measured data points a significant change of values has occurred. This value will inevitably be missed by interpolation, as it is an estimate of previous or following values. In our case, it is save to assume that data values actually do not change in between time samples, as we measure every time a change to the models was made. The software does not change in between commits and any missing measurement value can be interpolated from the last data point. Hence, we can take the smallest time interval between measurements and interpolate the rest of the measurements. This has another advantage as we can provide some insights into the discussion if re-sampling affects the correctness of the results as we can statistically test if the results of re-sampled data are similar to the original data and thus, we can determine if the extra effort of re-sampling is necessary in practice. We interpolated to daily intervals by using latest measurement at a specific day. This interval might introduce bias as it could hide interesting observations during a day but it provides a more realistic data set for practical observations.

Additional steps like differentiation and normalizing of the time series might be necessary. Differentiation removes the trend of a time series and enables separate investigations of trend and seasonality. Normalization of the data might be necessary, particularly for the use with neural networks as they are often adjusted to work with inputs ranging from 0 to 1.

T8) Application of the Predictions

The selection of prediction approaches is based on the outcome of the literature review. The criteria for the selection is how often an approach is mentioned in literature. The approaches have to be mentioned handling similarly structured data sets as in our case. For our study, both, statistical and machine learning approaches are considered. In our earlier study, the autoregressive integrated moving average (ARIMA) approach was applied [111]. As ARIMA is an established and proven approach to predict time series, we consider it as an appropriate approach for benchmarking purposes.

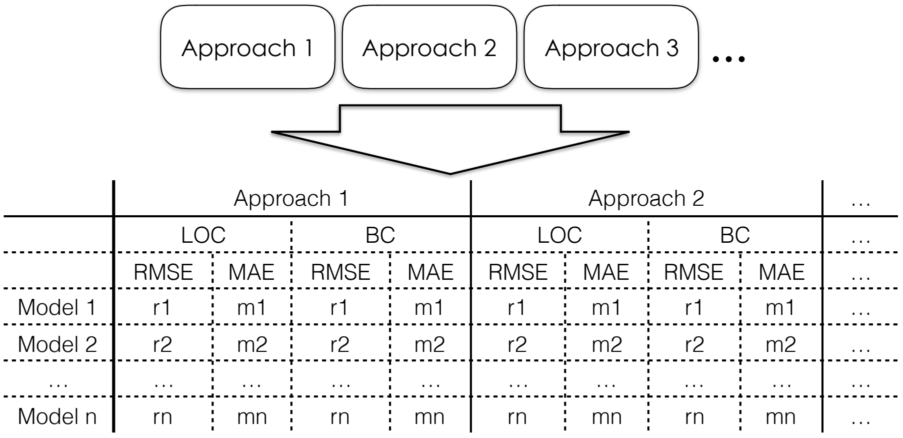
The application of the prediction approaches also has to consider their respective parameterization. Much time and effort is spent on optimizing parameters of predicting approaches to the data at hand. Research is conducted on how to fit prediction approaches best to a specific data set. Furthermore, for the majority of the approaches there are parameter estimation algorithms. As it is not the aim of this study to investigate perfect parameterizations but instead to provide a practical overview over existing approaches and their applicability in practice, we decided to use existing parameter estimation algorithms. The libraries providing the approaches usually also provide optimization algorithms for parameter estimation. The selected and configured approaches are then applied to the two lists containing the measurement data for the LOC and BC metrics created for all models and revisions in task T5. Therefore, we created the predictions for each metric and each model, respectively.

6.3.3 Analysis Procedure

In this section, we describe how the data collected in the survey and workshops (T1,T2) and the data from literature (T3,T4) was evaluated to answer RQ1.1 and RQ1.2. Furthermore, the analysis to determine the performance of the prediction approaches in step T9 answering RQ1.3 is outlined.

T1),T2) Analysis of Survey/Workshop Results and T3),T4) Classification of Literature

For the assessment of survey results we are not using statistical tests due to the limited sample size. We combine the answers in a table and visualize results to make informed decisions on their implications. The approaches found in the literature review are compared and selected by how often they are mentioned to be used on data with a similar structure.



	Approach 1				Approach 2				…
	LOC		BC		LOC		BC		…
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	…
Model 1	r1	m1	r1	m1	r1	m1	r1	m1	…
Model 2	r2	m2	r2	m2	r2	m2	r2	m2	…
…	…	…	…	…	…	…	…	…	…
Model n	rn	mn	rn	mn	rn	mn	rn	mn	…

Figure 6.5: Evaluation of the calculated error measurements. For each approach, the measurement results for all models are compared regarding their prediction error.

T9) Evaluation of Prediction Results

To compare the prediction results regarding their accuracy, the prediction error has to be defined. For predicting regression-based, we determine how close predictions are to the ground truth data. There are multiple possible prediction error measurements mentioned in literature having different strengths, weaknesses, and biases. Additionally, many approaches are very closely related. Based on Adhikari and Agrawal [114] and Hyndman and Athanasopoulos [115], we selected the root mean squared error (RMSE), and the mean absolute error (MAE); RMSE is calculated by $\sqrt{\sum(\frac{(Prediction-GroundTruth)^2}{n})}$ and MAE is calculated by $(n * \sum|Prediction - GroundTruth|)^{-1}$. While the RMSE has advantages of making errors comparable across models, the MAE provides more conclusive information about the error for each model respectively. Hence, the RMSE will be used for comparing predictions, while the MAE is used to fulfill engineers' expectations of accuracy.

In order to systematically compare the results, we followed guidelines from Basili et al. [41] and Wohlin et al. [42]. We created a hypothesis, determined independent variables, and ran statistical tests to find statistical significant observations. The comparison of the prediction results is visualized in Figure 6.5. Using the data presented in this form, the analysis starts with determining the distribution of the data to decide if parametric or non-parametric tests should be used. The next step is comparing the prediction approaches by their prediction errors. We want to find out if there is a statistically significant difference among the error measurements grouped by the different approaches. Hence, the following hypothesis is created:

H0 The samples of error measurements for the different approaches originate from identical populations.

Ha The samples come from different populations.

Parametric or non-parametric statistical tests are used to reveal significant differences within the results. Using tests, we can find out if there is one approach performing significantly better or worse than the others regarding prediction accuracy on short and long prediction ranges and regarding the used metric. Additionally, we also investigate maintenance effort and run time that the practitioners were asked about in the workshops.

6.3.4 Validity Procedure

To ensure validity while conducting the study, we focus on employing multiple combinations of approaches and methods in order to avoid bias. We use five different prediction approaches with different underlying algorithms, two different size metrics, and 48 independent models with and without data interpolation. We are performing rigorous statistical analysis using established statistical tests.

6.4 Results

In the results, we consecutively answer the research questions by presenting the outcomes of the associated tasks. Practitioners expectations are followed by important prediction approaches from literature. Lastly, we present and analyze the prediction results.

6.4.1 Expectations of Practitioners towards Predictions

Findings from the survey are summarized in Table 6.2. The table shows that predicting over long intervals is most important to the stakeholders. The respective answer received 22 out of 24 points. The importance of the other three answers is close. Short time accuracy received 14, maintenance time 13, and run time 11 points. The answers for how much time a prediction may take are varying. Two practitioners expect it to take less than 1 hour, two find 24 hours and more acceptable. The other two are in between these values. The answers for accuracy are more clear: Engineers accept a high error rate of 10% but expect it to hold for the majority of the models, with 75%. Both, the results for the error rates and for how far to predict confirm the results of the importance rating. High error rates are acceptable as long as predictions stay accurate even for long distances. The acceptable maintenance times mentioned by the participants highlight automation. Engineers expect the predictions to be run in an automated way and accept associated initial effort. Engineers expect less than three hours of maintenance work on the predictions per week. We could not identify a study investigating practitioners expectations in a similar context to compare the received data to.

6.4.2 Predictions approaches Identified in Literature and Previous Studies

The results of the conducted literature review reveal two statements regarding the comparison between linear and non-linear approaches and the comparison within machine learning approaches. Firstly, linear approaches are usually outperformed by non-linear ones, especially in cases when the data exhibits lots of random noise (cf. [103], [116]). Secondly, a majority of identified studies

Table 6.2: The answers collected from the survey.

	Importance Rating				max time	acceptable	predictions	how far to	acceptable maintenance
	accuracy long	accuracy short	mainte-nance	run-time	to predict	error	correct for % models	predict	
P1	●●●●	●●	●●●	●	>24h	10%	60%	14 days	“automation, big initial effort accept.”
P2	●●	●●●	●	●●●●	<1h	10%	70%	21 days	“less than a day”
P3	●●●●	●●●	●●	●	<12h	10%	80%	7 days	“3h per week”
P4	●●●●	●●●	●	●●	>24h	10%	80%	30 days	“automation, mod. init. effort accept.”
P5	●●●●	●	●●●	●●	<1h	5%	80%	90 days	“1h per week”
P6	●●●●	●●	●●●	●	≤24h	15%	80%	28 days	“2h per week”

Table 6.3: The approaches used in this study and their specifications.

Category	Approach	Tool	Library	Parameter estimation	Parameters
Statistical, Linear	HOLT	R	forecast	none	none
Statistical, Linear	ARIMA	R	forecast	built-in optimization	AR, I, MA
Machine Learning, Non-Linear	SVR	Python	scikit-learn	grid search	kernel, C, gamma
Machine Learning, Non-Linear	AVNNET	R	caret	caret	lag, hidden
Machine Learning, Non-Linear	LSTM	Python	Keras	manual and built-in grid search	lag, # epochs, hidden neurons neurons, optimizer

highlight the implementation of support vector machines (SVM) and artificial neural networks (ANN) implemented in a variety of domains.

Notably, both linear and non-linear approaches have their strengths and weaknesses. The linear approaches exhibit good prediction performance when time series are stationary, non-trending data (cf. [117]). This is because linear approaches predict values based on the previous data in the time series. To circumvent this weakness, approaches exist to make input data for linear approaches stationary. Non-linear approaches such as ANN or SVM have their strengths in the robustness of the prediction if the data is limited, or from a short-term period. However, their weakness is a lengthy training process as mentioned by Sapankevych and Sankar [118], Vanajakshi and Rilett [119], and Meyfroidt et al. [120].

From literature, we extracted the approaches as listed in Table 6.3 that are briefly outlined in the following. With this list of approaches and their descriptions, we answer RQ1.2 on the most important prediction approaches currently used in literature.

Holt's linear trend method (HOLT)

This approach is serving as reference for comparison. As shown in Section 6.3.2 in task T6, the time series are characterized by their upwards trend. Smoothing approaches like Holt's are designed for forecasting trends (cf. [115]) as it provides a more accurate prediction than a simple linear regression. As it is provided with R using the "holt" function contained in the forecast package and does not require configuration, it was chosen as the prediction benchmark approach to which the more complex approaches are compared. The only input required is the time series itself.

Autoregressive integrated moving average (ARIMA) model

ARIMA is an advanced regression approach and commonly used with time series data. The approach combines an autoregressive function (AR), differentiation (I), and a moving averages function (MA). These functions build the three main configuration parameters for model creation with ARIMA. ARIMA was used in a previous study with similar data to create models for outlier detection (cf. [111]). The R package "forecast" provides an "auto-arima" function, which compares multiple ARIMA configurations and selects the configuration according to the model quality. The only input required is the time series itself.

Feed-Forward Artificial Neural Network (ANN)

Feed-forward artificial neural networks are widely used for time series analysis already. As neural networks are based on learning from past data, it has to be determined which input the network should be provided and how. To receive comparable results, we taught the ANN with the same data as all other approaches: one time series of measurement data to create a prediction for the same data. For training and prediction, we provided the network with a set of past data points, called lagged data points, instead of feeding one data point at a time. Hence, the network can learn to predict a point in the future from multiple past revisions. We used the "AVNNET" implementation provided by the “caret” package in R as it provides automatic parameter estimation functions. The only configuration parameter was the size of the input.As ANN base on so-called activation functions, typically ranging from 0 to 1, we adjusted our data to this range by normalization.

Long Short Term Memory (LSTM)

Long short term memories are recurrent neural networks, which have drawn attention in the field of forecasting time series in the recent years due to their performance (cf. [121], [122], and [123]). An LSTM enhances a plain feed-forward neural network by a memory layer to store information from a learning step and reuse it to influence a current learning step. The “Keras” package provides LSTM algorithms for Python. It also provides a grid search algorithm for parameter estimation. The grid search tries all provided combinations of configuration parameters and supplies the network with the least error in a given test data. As for the ANN, LSTM is provided with lagged, normalized input. The following parameters are tried during the grid search: Number of hidden neurons, length of lagged input, number of epochs to train, and the optimizer to be used.

Both neural networks use random weights of the neurons in the beginning of learning. Hence, the results created are not deterministic every time the networks are trained. To address the non-determinism and receive similar results for each teaching of the networks, we averaged the results of multiple runs as an established means in the field (cf. [124]).

Support Vector Regression (SVR)

Support Vector Regression is based on the Support Vector Machine. We implemented it using the “scikit-learn” package in Python. The configuration parameters are the error metric to be used for predictions, the kernel, the penalty parameter C, and the coefficient for the kernel $gamma$. The scikit-learn package provides a grid search algorithm to find the best set of configuration parameters.

6.4.3 Prediction Results

In this section we present results from applying the previous approaches to data received in industry. We analyze the results to conclude about the applicability of the approaches regarding the priorities elicited in the survey and workshops.

6.4.3.1 Prediction Accuracy

The results of the accuracy investigations are summarized in Figure 6.6. The figure shows how the errors computed by RMSE for all five approaches are distributed among the data. In general, most errors are low while some outlying spikes are visible. If there are spikes, the approaches mostly increase altogether like for model 5 and 28. Furthermore, sometimes only the machine learning approaches are outlying, like in model 16 and 21. Additionally, as the models are ordered by size in each diagram, it can be seen that the prediction error increases with size in each function group.

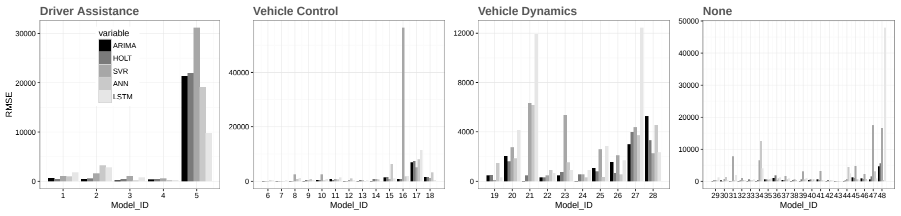


Figure 6.6: Comparison of the approaches' prediction errors (RMSE), arranged by four groups of functionality using the LOC metric.

Table 6.4: Results of the accuracy investigations considering practitioners' requirements of 10% using all 48 models.

	HOLT	ARIMA	SVR	ANN	LSTM
# of models above 10%	5	5	18	11	13
% of models above 10%	10.4	10.4	37.5	22.9	27.1

We investigated the prediction accuracy using the engineers expectations. According to Table 6.2 in Section 6.4.1, practitioners found a deviation of 10% from the ground truth acceptable on average, if it is achieved for 60% to 80% of the models. To calculate the percentage, we used the mean absolute error (MAE), as it represents the absolute error for each and should in this case not be bigger than 10% of the average ground truth data. Table 6.4 shows the results of this analysis.

This data shows the robustness of the approaches by counting how many models did not achieve required accuracy. The table shows that the approaches are generally able to fulfill the accuracy requirements. All approaches achieved the required accuracy for more than 60% of the models. The two statistical approaches achieve it for almost 90% of the models.

As these results are specific to the studied case, we address the hypothesis on a difference within the approaches by performing statistical tests on the error data as previously shown in Figure 6.5 in Section 6.3.2 to evaluate a generalization. The resulting 48 values of RMSE for all approaches are not normally distributed. Hence, we use the KruskalWallis analysis of variance for comparison and we test for short and long term prediction accuracy. Whereas for long term the whole set is used, we use four revisions into the future for the short term, which is the smallest amount of prediction length achieved by all approaches. Additionally, LOC and BC metrics are compared for validity reasons in both cases. Hence, we have four cases to compare the five approaches. Table 6.5 shows the results of the tests. The test compares group medians and from the table we conclude that not all group medians are equal. Regarding the short term predictions, we can reject the null hypothesis of equal populations. We have evidence that at least one approach is significantly different. For long term predictions we can reject the null hypothesis in case of the BC metric. Both considering a significance level α of 0.05. The ranks provided by the test give an idea on the effect size. ARIMA has the lowest medians within RMSE of all approaches, while SVR has the highest. Additionally, the results suggest

Table 6.5: Accuracy comparison using the Kruskal-Wallis test.

		short term prediction		long term prediction	
		LOC	BC	LOC	BC
	HOLT	110.25	112.23	114.02	113.27
Kruskal-	ARIMA	87.67	100.60	98.00	105.67
Wallis	SVR	144.96	151.35	137.54	146.17
Ranks	ANN	126.38	110.54	125.02	113.75
	LSTM	133.25	127.77	127.92	123.65
p-values		0.001	0.004	0.059	.043

that the approaches converge towards long term predictions. As long term predictions were by far the most important property to the practitioners, this is a major finding.

6.4.3.2 Prediction Maintenance Effort

The implementations of the approaches in the different languages using different libraries strongly depend on expertise with the respective language. Maintenance effort based on code size or complexity cannot be used as it is hard to compare the metrics among different languages. Nevertheless, as we used parameter estimation algorithms which automatically estimate the optimal set of parameters, the maintenance effort for future predictions is small for all approaches. As data changes the algorithms will find matching sets of parameters. In this regard, we evaluate all approaches as performing similarly well.

6.4.3.3 Prediction Run Time

Similarly to the maintenance effort, run time depends on the implementation and the underlying computer system used. Run time differences can still be compared, as machine learning approaches require a learning period while statistical approaches do not require this step. It depends on the accuracy required how long learning periods have to be and how often they are performed. If run time is an important aspect, statistical approaches are preferable. As in practice predictions could run during night time, run time is not an issue in the context of predicting time series of measurement data.

6.4.4 Threats to Validity

As this case study focuses on one specific case in industry, we assess generalizability to other domains as the biggest threat to the validity of the results. We mitigated this threat by designing the study in a way to cover multiple different models, metrics, data formats, and error measures, to reduce the chance that results are just received by chance in the context.

Using machine learning and particularly deep learning approaches, there is always the possibility of further optimization. While the results obtained from these approaches can be further optimized, we mitigated this threat to construct validity by using automatic parameter estimators to ensure a fair treatment of all approaches.

Furthermore, the approaches might perform differently using other programming languages or libraries. Even though the approaches should be implemented as specified by formulas in publications, it could happen that the same approach is implemented differently in different libraries/tools. Hence, there is a threat that results can differ when replicating the research with different libraries or tools. We mitigated that threat by implementing our approaches in widely used tools using common libraries.

Due to the small sample size in the survey, it does not represent all practitioners in industry. We tried to mitigate this threat by considering practitioners in all different existing roles at the department. Additionally, the lead developers of all function groups were surveyed. Still, in other domains the expectations towards similar predictions might differ.

6.5 Conclusion and Future Work

All investigated approaches are applicable to perform time series predictions in the studied context. They all fulfill the basic requirements requested by practitioners in our qualitative study. Particularly, long term predictions were important to them; where the approaches showed less deviation among their results. Nevertheless, we found evidence that there are actual differences in accuracy of the approaches for three of the four statistical tests performed. In the context of our case, ARIMA showed best accuracy results and short run times; Support Vector Regression on the other hand requires longer run time due to training and achieved worse accuracy results. When considering only machine learning approaches simple feed-forward provide the highest accuracy in the given context.

As machine learning approaches are designed to take multiple parallel inputs, an future investigation of additional inputs might result in improved prediction accuracy of those approaches. Generating a big data set from all models to enable prediction for a specific model based on learning data from all other models might be a first step to increase the amount of inputs.

Acknowledgment

Thanks to Hans Salomonsson and the BigData group at Chalmers University of Technology for the support and supervision during the application of the machine learning approaches.

Bibliography

[1] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 33–42.

[2] "Codebases – millions of lines of code," http://www.informationisbeautiful.net/visualizations/million-lines-of-code/, accessed: 2016-11-18.

[3] R. N. Charette, "This car runs on code," http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code, 2009, accessed: 2016-11-18.

[4] V. Kaznov, "ARCHER – FUSE Final Seminar," Presentations at the FUSE Final Seminar, 2016 (accessed October 12, 2016). [Online]. Available: http://www.fuse-project.se/Homepage/Download-File/f/884201/h/510e23331012b7ac1206ba994387b951/2016%2B09%2B23+ARCHER+%2B+FUSE+seminar+%2B+Scania+%2B+Viktor+Kaznov

[5] "2008 volkswagen passat 81747 recalls," http://www.volkswagenextendedwarranty.com/recalls/volkswagen/2008/passat/nhtsa-81747.html/, 2008, accessed: 2016-11-18.

[6] P. Koopman, "A case study of toyota unintended acceleration and software safety," October 2014, [Accessed 21 December 2014]. [Online]. Available: http://betterembsw.blogspot.de/2014/09/a-case-study-of-toyota-unintended.html

[7] J. Mössinger, "Software in automotive systems," *IEEE Software*, vol. 27, no. 2, pp. 92–94, March 2010.

[8] C. Brückner and P. Weitl, "Modulare simulationsmodelle für x-in-the-loop-prüfstände," in *Proceedings of 3. deutschsprachige NAFEMS Regionalkonferenz*, 2016.

[9] IEEE, "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, Dec 1990.

[10] B. W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Software*, vol. 1, no. 1, pp. 75–88, Jan 1984.

[11] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals.* Santa Barbara: Rocky Nook, 2011.

[12] M. Glinz, "On non-functional requirements," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, Oct 2007, pp. 21–26.

[13] J. Eckhardt, A. Vogelsang, and D. Méndez Fernández, "On the distinction of functional and quality requirements in practice," in *Proceedings of the 17th International Conference on Product-Focused Software Process Improvement*, Nov 2016.

[14] F. D. Giraldo, S. Espana, and O. Pastor, "Analysing the concept of quality in model-driven engineering literature: A systematic review," in *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, May 2014, pp. 1–12.

[15] "Mda - the architecture of choice for a changing world," www.omg.org/mda/, accessed: 2016-11-18.

[16] ISO/IEC, "Information technology - software product quality - part 1: Quality model, iso/iec fdis 9126-1:2000(e)," Geneva, 2000.

[17] ——, "Systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models, iso/iec 25010:2011(en)," 2011.

[18] H. Al-Kilidar, K. Cox, and B. Kitchenham, "The use and usefulness of the iso/iec 9126 quality standard," in *2005 International Symposium on Empirical Software Engineering.*, 2005, pp. 7–pp.

[19] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development a review of literature," *Information and Software Technology*, vol. 51, no. 12, pp. 1646 – 1669, 2009.

[20] J. Schroeder, C. Berger, and T. Herpel, "Challenges from integration testing using interconnected hardware-in-the-loop test rigs at an automotive oem: An industrial experience report," in *Proceedings of the First International Workshop on Automotive Software Architecture*, ser. WASA, 2015, pp. 39–42.

[21] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, no. 0, pp. 193 – 220, 2015.

[22] W. Cunningham, "The wycash portfolio management system," in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30.

[23] J. Schroeder, C. Berger, T. Herpel, and M. Staron, "Comparing the applicability of complexity measurements for simulink models during integration testing – an industrial case study," in *Proceedings of the Second International Workshop on Software Architecture and Metrics (SAM)*, 2015, pp. 35–40.

[24] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498 – 1516, 2013.

[25] N. S. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, “Identification and management of technical debt: A systematic mapping study,” *Information and Software Technology*, vol. 70, pp. 100 – 121, 2016.

[26] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2014.

[27] A. Abran, *Software metrics and software metrology.* John Wiley & Sons, 2010.

[28] MathWorks Inc., *Sldiagnostics – Display diagnostic information about Simulink system*, 2016 (accessed January 25, 2016). [Online]. Available: http://www.mathworks.com/help/simulink/slref/sldiagnostics.html

[29] A. Jantsch, “Chapter one - introduction,” in *Modeling Embedded Systems and SoC’s*, ser. Systems on Silicon, A. Jantsch, Ed. San Francisco: Morgan Kaufmann, 2003, pp. 1 – 46.

[30] B. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd ed. Academic Press, 2000.

[31] M. Staron, *Adopting Model Driven Software Development in Industry – A Case Study at Two Companies.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 57–72.

[32] R. Malhotra, *Empirical Research in Software Engineering: Concepts, Analysis, and Applications.* Boca Raton: Taylor & Francis, 2016.

[33] K.-J. Stol and B. Fitzgerald, “A holistic overview of software engineering research strategies,” in *Proceedings of the Third International Workshop on Conducting Empirical Studies in Industry*, ser. CESI ’15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 47–54.

[34] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples.* John Wiley & Sons, 2012.

[35] J. McDonough and S. McDonough, *Research Methods for English Language Teachers*, ser. Hodder Arnold Publication. Arnold, 1997.

[36] M. V. Zelkowitz and D. R. Wallace, “Experimental models for validating technology,” *Computer*, vol. 31, no. 5, pp. 23–31, May 1998. [Online]. Available: http://dx.doi.org/10.1109/2.675630

[37] Z. Zainal, “Case study as a research method,” *Jurnal Kemanusiaan*, no. 9, pp. 1–6, 2007.

[38] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[39] C. B. Seaman, *Qualitative Methods.* London: Springer London, 2008, pp. 35–62.

[40] J. Singer, S. E. Sim, and T. C. Lethbridge, *Software Engineering Data Collection for Field Studies.* London: Springer London, 2008, pp. 9–34.

[41] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Trans. Softw. Eng.*, vol. 12, no. 7, pp. 733–743, Jul. 1986. [Online]. Available: http://dl.acm.org/citation.cfm?id=9775.9777

[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[43] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research - An Initial Survey," in *22nd International Conference on Software Engineering and Knowledge Engineering*, 2010, pp. 374–379.

[44] K. Grimm, "Software technology in an automotive company - major challenges," in *Proceedings of the 25th International Conference on Software Engineering*, 2003, pp. 498–503.

[45] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2014, pp. 85–92.

[46] F. Shull, J. Singer, and D. I. K. Sjøberg, Eds., *Guide to Advanced Empirical Software Engineering.* London: Springer London, 2008. [Online]. Available: http://www.springerlink.com/index/10.1007/978-1-84800-044-5

[47] C. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 557–572, July 1999.

[48] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. ACM, 2010, pp. 47–52.

[49] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola, "Towards an ontology of terms on technical debt," in *Proceedings of the 6th IEEE International Workshop on Managing Technical Debt.* Los Alamitos, CA, USA: IEEE, 2014, pp. 1–7.

[50] N. A. Ernst, "On the role of requirements in understanding and managing technical debt," in *Proceedings of the Third International Workshop on Managing Technical Debt*, ser. MTD '12. Piscataway, NJ, USA: IEEE, 2012, pp. 61–64.

[51] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, pp. 18–21, 2012.

[52] K. Lee, Y. Ki, J. Cheon, G. Hwang, and H. Ahn, "Approach to functional safety-compliant ecu design for electro-mechanical brake systems," *International Journal of Automotive Technology*, vol. 15, no. 2, pp. 325–332, 2014.

[53] S. Köhl and D. Jegminat, "How to do hardware-in-the-loop simulation right," SAE International, Tech. Rep. SAE Technical Paper 2005-01-1657, 2005.

[54] J. Schroeder, D. Holzner, C. Berger, C.-J. Hoel, L. Laine, and A. Magnusson, "Design and evaluation of a customizable multi-domain reference architecture on top of product lines of self-driving heavy vehicles an industrial case study," in *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, May 2015.

[55] IEEE, "IEEE standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, Dec 1990.

[56] A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and P. Traverso, "Model checking safety critical software with spin: an application to a railway interlocking system," in *Computer Safety, Reliability and Security*, ser. Lecture Notes in Computer Science, W. Ehrenberger, Ed. Springer Berlin Heidelberg, 1998, vol. 1516, pp. 284–293.

[57] J. Bowen and M. Hinchey, "The use of industrial-strength formal methods," in *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International*, Aug 1997, pp. 332–337.

[58] C. Snook and R. Harrison, "Practitioners' views on the use of formal methods: an industrial survey by structured interview," *Information and Software Technology*, vol. 43, no. 4, pp. 275 – 283, 2001.

[59] G. Broadfoot and P. Broadfoot, "Academia and industry meet: some experiences of formal methods in practice," in *Software Engineering Conference, 2003. Tenth Asia-Pacific*, Dec 2003, pp. 49–58.

[60] C. Heitmeyer, "On the need for practical formal methods," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ser. Lecture Notes in Computer Science, A. Ravn and H. Rischel, Eds. Springer Berlin Heidelberg, 1998, vol. 1486, pp. 18–26. [Online]. Available: http://dx.doi.org/10.1007/BFb0055332

[61] M. Whalen, D. Cofer, S. Miller, B. Krogh, and W. Storm, "Integration of formal analysis into a model-based software development process," in *Formal Methods for Industrial Critical Systems*, ser. Lecture Notes in Computer Science, S. Leue and P. Merino, Eds. Springer Berlin Heidelberg, 2008, vol. 4916, pp. 68–84.

[62] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Trner, "Early verification and validation according to iso 26262 by combining fault injection and mutation testing," in *Software Technologies*, ser. Communications in Computer and Information Science, J. Cordeiro and M. van Sinderen, Eds. Springer Berlin Heidelberg, 2014, vol. 457, pp. 164–179.

[63] R. Wieringa and M. Daneva, "Six strategies for generalizing software engineering theories," *Science of Computer Programming*, vol. 101, no. 0, pp. 136 – 152, 2015.

[64] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decision Support Systems*, vol. 13, no. 34, pp. 241 – 262, 1995.

[65] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, June 1994.

[66] M. Plaska, "Simulink-specific design quality metrics," Turku Centre for Computer Science, Tech. Rep. TUCS Tech. Rep. 1002, February 2011.

[67] C. F. Kemerer, "Software complexity and software maintenance: A survey of empirical research," *Annals of Software Engineering*, vol. 1, no. 1, pp. 1–22, 1995.

[68] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig, "Software errors and software maintenance management," *Inf. Technol. and Management*, vol. 3, no. 1-2, pp. 25–41, January 2002.

[69] D. Card and W. Agresti, "Measuring software design complexity," *Journal of Systems and Software*, vol. 8, no. 3, pp. 185 – 197, 1988.

[70] M. Plaska and M. Waldén, "Quality comparison and evaluation of digital hydraulic control systems," Turku Centre for Computer Science, Tech. Rep. TUCS Technical Report 857, 2007.

[71] P. Boström, R. Grönblom, T. Huotari, and J. Wiik, "An approach to contract-based verification of simulink models," Turku Centre for Computer Science, Tech. Rep. TUCS Technical Report 985, 2010.

[72] Y. Dajsuren, M. G. van den Brand, A. Serebrenik, and S. Roubtsov, "Simulink models are also software: Modularity assessment," in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, QoSA '13*. New York, NY, USA: ACM, 2013, pp. 99–106.

[73] R. Feldmann, "Complexity- and performance analysis of different controller implementations on a soft plc," in *Demos/Posters/StudentResearch@MoDELS'13*, 2013, pp. 118–123.

[74] E. A. Antonio, F. Ferrari, G. A. d. P. Caurin, and S. C. P. F. Fabbri, "A set of metrics for characterizing simulink model comprehension," *Journal of Computer Science & Technology*, vol. 14, no. 2, pp. 88–94, 2014.

[75] B. Anda, D. Sjoberg, and A. Mockus, "Variability and reproducibility in software engineering: A study of four companies that developed the same system," *IEEE Trans. Softw. Eng.*, vol. 35, no. 3, pp. 407–429, May 2009.

[76] D. Landman, A. Serebrenik, and J. Vinju, "Empirical analysis of the relationship between cc and sloc in a large corpus of java methods," in *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014*, 2014, pp. 221–231.

[77] G. Jay, J. Hale, R. Smith, D. Hale, N. Kraft, and C. Ward, "Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship," *Journal of Software Engineering and Applications*, vol. 2, no. 3, pp. 137–143, 2009.

[78] J. Schroeder, C. Berger, T. Herpel, and M. Staron, "Comparing the applicability of complexity measurements for simulink models during integration testing an industrial case study supplementary material," [Accessed 26 February 2015]. [Online]. Available: http://goo.gl/pKChy0

[79] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, October 1996.

[80] T. Khoshgoftaar and R. Szabo, "Improving code churn predictions during the system test and maintenance phases," in *Software Maintenance, 1994. Proceedings., International Conference on*, Sep 1994.

[81] M. Dagpinar and J. H. Jahnke, "Predicting maintainability with object-oriented metrics - an empirical comparison," in *Proceedings of the 10th Working Conference on Reverse Engineering*, ser. WCRE '03, 2003, pp. 155–164.

[82] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, pp. 68–86, 1996.

[83] H. Zuse, "Properties of software measures," *Software Quality Journal*, vol. 1, no. 4, pp. 225–260, 1992.

[84] E. Weyuker, "Evaluating software complexity measures," *IEEE Trans. Softw. Eng.*, vol. 14, no. 9, pp. 1357–1365, Sep 1988.

[85] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[86] M. Gupta, J. Gao, C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Trans. on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, Sept 2014.

[87] C. C. Aggarwal, *Outlier Analysis*, 1st ed. Springer-Verlag New York, 2013.

[88] D. P. Hartmann, J. M. Gottman, R. R. Jones, W. Gardner, A. E. Kazdin, and R. S. Vaught, "Interrupted time-series analysis and its application to behavioral data," *Journal of Applied Behavior Analysis*, vol. 13, no. 4, pp. 543–559, 1980.

[89] G. E. B. Box and G. M. Jenkins, *Time Series Analysis, Forecasting and Control.* San Francisco: Holden Day, 1970.

[90] B. L. Bowerman, R. T. O'Connell, and E. S. Murphree, *Business Statistics in Practice*, 5th ed. McGraw Hill Higher Education, 2008.

[91] O. Alan and C. Catal, "Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3440 – 3445, 2011.

[92] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02, 2002, pp. 291–301.

[93] K. Herzig, S. Just, A. Rau, and A. Zeller, "Predicting defects using change genealogies," in *Proceedings of the 2013 IEEE 24nd International Symposium on Software Reliability Engineering.* IEEE, November 2013.

[94] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, 2014, pp. 72–81.

[95] J. Gil, M. Goldstein, and D. Moshkovich, "An empirical investigation of changes in some software properties over time," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, June 2012, pp. 227–236.

[96] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose, "Characterization and prediction of issue-related risks in software projects," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, 2015, pp. 280–291.

[97] A. Pika, W. M. P. Aalst, C. J. Fidge, A. H. M. Hofstede, and M. T. Wynn, *Advanced Information Systems Engineering: 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings.* Springer Berlin Heidelberg, 2013, ch. Profiling Event Logs to Configure Risk Indicators for Process Delays, pp. 465–481.

[98] K. M. Eisenhardt, "Building theories from case study research," *The Academy of Management Review*, vol. 14, no. 4, pp. 532–550, 1989.

[99] M. Hiller, "Thoughts on the Future of the Automotive Electronic Architecture," Presentations at the FUSE Final Seminar, 2016 (accessed October 12, 2016). [Online]. Available: http://www.fuse-project.se/Homepage/Download-File/f/874242/h/5cfffa89cdaf3a5a46a600d3420b920b/Martin+Hiller+Thoughts+on+

the+Future+of+the+Automotive+Electronic+Architecture+%2B+V1.1

[100] O. Wyman, "A comprehensive study on innovation in the automotive industry," Oliver Wyman Group, Tech. Rep., 2015. [Online]. Available: http://www.oliverwyman.com/content/dam/oliver-wyman/global/en/2014/dec/CarInnovation2015_eng_final.pdf

[101] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, May 2009. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2008.10.027

[102] T. chung Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164 – 181, 2011.

[103] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504 – 518, 2015.

[104] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. C. Riquelme, "A survey on data mining techniques applied to electricity-related time series forecasting," *Energies*, vol. 8, no. 11, pp. 13 162–13 193, 2015.

[105] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, July 2010.

[106] T. Zimmermann, "Changes and bugs – mining and predicting development activities," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, Sept 2009, pp. 443–446.

[107] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, Jan 2012.

[108] M. Arnold and P. Pedross, "Software size measurement and productivity rating in a large-scale software development department," in *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, Apr 1998, pp. 490–493.

[109] A. G. Koru and H. Liu, "Building effective defect-prediction models in practice," *IEEE Software*, vol. 22, no. 6, pp. 23–29, Nov 2005.

[110] H. Preenja and M. Ali, "Predicting time series data collected from software measurements with machine learning approaches," Master's thesis, University of Gothenburg, Gothenburg, Sweden, 6 2016.

[111] J. Schroeder, C. Berger, M. Staron, T. Herpel, and A. Knauss, "Unveiling anomalies and their impact on software quality in model-based automotive software revisions with software metrics and domain experts," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: ACM, 2016, pp. 154–164. [Online]. Available: http://doi.acm.org/10.1145/2931037.2931060

[112] A. Eckner, "A framework for the analysis of unevenly spaced time series data," *Preprint. Available at: http://www.eckner.com/papers/unevenly_spaced_time_series_analysis*, 2012.

[113] K. Rehfeld, N. Marwan, J. Heitzig, and J. Kurths, "Comparison of correlation analysis techniques for irregularly sampled time series," *Nonlinear Processes in Geophysics*, vol. 18, no. 3, pp. 389–404, 2011.

[114] R. Adhikari and R. K. Agrawal, "An introductory study on time series modeling and forecasting," *CoRR*, vol. abs/1302.6613, 2013.

[115] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2013.

[116] S. F. Crone, S. Lessmann, and S. Pietsch, "Forecasting with computational intelligence - an evaluation of support vector regression and artificial neural networks for time series prediction," in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 2006, pp. 3159–3166.

[117] M. P. Garcia and D. S. Kirschen, "Forecasting system imbalance volumes in competitive electricity markets," in *Power Systems Conference and Exposition, 2004. IEEE PES*, Oct 2004, pp. 1805–1812 vol.3.

[118] N. I. Sapankevych and R. Sankar, "Time series prediction using support vector machines: A survey," *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, May 2009.

[119] L. Vanajakshi and L. R. Rilett, "Support vector machine technique for the short term prediction of travel time," in *2007 IEEE Intelligent Vehicles Symposium*, June 2007, pp. 600–605.

[120] G. Meyfroidt, F. Giza, J. Ramon, and M. Bruynooghe, "Machine learning techniques to examine large patient databases," *Best Practice & Research Clinical Anaesthesiology*, vol. 23, no. 1, pp. 127 – 143, 2009.

[121] M. Assaad, R. Bon, and H. Cardot, "A new boosting algorithm for improved time-series forecasting with recurrent neural networks," *Information Fusion*, vol. 9, no. 1, pp. 41 – 55, 2008, special Issue on Applications of Ensemble Methods.

[122] S. Ho, M. Xie, and T. Goh, "A comparative study of neural network and box-jenkins ARIMA modeling in time series prediction," *Computers & Industrial Engineering*, vol. 42, no. 24, pp. 371 – 375, 2002.

[123] H. Cheng, P.-N. Tan, J. Gao, and J. Scripps, *Multistep-Ahead Time Series Prediction.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 765–774.

[124] B. D. Ripley, *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.