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

Jan Schroeder, Christian Berger, Miroslaw Staron
University of Gothenburg, Department of Computer Science and Engineering, Gothenburg, Sweden
{jan.schroder; christian.berger; miroslaw.staron}@gu.se

Thomas Herpel
Automotive Safety Technologies GmbH, Ingolstadt, Germany
thomas.herpel@astech-auto.de

Alessia Knauss
Chalmers University of Technology, Department of Computer Science and Engineering, Gothenburg, Sweden
alessia.knauss@chalmers.se

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.

CCS Concepts

•General and reference → Metrics; •Computing methodologies → *Model verification and validation;*

Keywords

Metrics; Outliers; Model-based; Empirical Case Study

ISSTA'16, July 18–20, 2016, Saarbrücken, Germany
ACM. 978-1-4503-4390-9/16/07...$15.00
http://dx.doi.org/10.1145/2931037.2931060

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.

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 [25], 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. [20] and [9]). Additionally, solutions are usually designed for object oriented code and not directly applicable to model-based software.

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:

1. How can existing linear regression / model fitting and respective outlier analysis be applied to revision data from an industrial context?
2. Which anomalies and patterns can be detected applying the above approaches to measurement data in form of software revision time series?
3. Which correlations between outlying values in the measurements and the domain expert opinions can be found?
4. Which combination of measurements and outlier detection produces meaningful observations about software quality?

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.

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.

1.5 Structure of this Study

In this study we followed Runeson and Höst's recommendation on structuring case studies (cf. [24]). Hence, the paper is outlined as follows: In Section 2, we describe the necessary background on all approaches applied in this study and discuss related work. The study design in Section 3 outlines the concept applied to investigate the study's research objectives. Results are presented in Section 4, analyzed in Section 5, and assessed for validity in Section 6. We conclude our study in Section 7 and comment on possible future work.

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.

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. [26]).

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 [21]. Both size measurements are performed offline on the model files.

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. [5] emphasized the difficulty of defining complexity metrics. Fenton & Bieman ([12], pp. 425) refer to Zuse's paper [30], proving that the list of properties for complexity measurements established by Weyuker [29] 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 [6] in 1988. Plaska & Waldén [23] applied the metrics, which were originally intended for software design, to model-based software. In Schroeder et al. [26], 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 model, divided by the number of blocks (n), as shown in Equation 1.

$$SC = \frac{\sum f_i^2}{n} \tag{1}$$

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 2 shows the definition of this metric.

$$DC = \frac{\sum \frac{v_i}{f_i+1}}{n} \tag{2}$$

Both equations conform to their definition according to Card & Agresti [6]. Similar to the size metrics, the complexity metrics can be performed offline by parsing the model files.

2.2 Anomaly Detection

Outlier detection in general is well studied. Surveys by Chandola et al. [7] and specifically for time series data by Gupta et al. [14] collect research and establish taxonomies for outlier detection. For defining anomalies among outliers, we follow Aggarwal [1]. When assessing measurement data for outliers, the observations can be divided into three classes. Figure 1 shows that most of the data indicates normal behavior. Next to the normal data, there is data that behaves

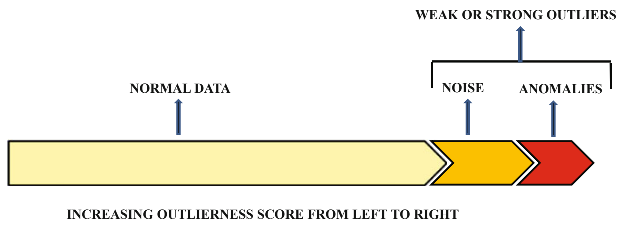

Figure 1: Graphic from [1] showing difference between outliers and anomalies.

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. [16] 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.

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. [16] 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 [4]. 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 2. In the figure, three values are used for the average resulting in a MA(3) model.

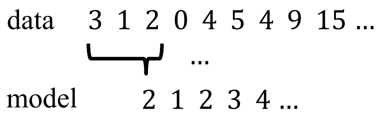

Figure 2: Visualization of a moving average model. The model is created building the average of three previous observations: MA(3).

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. [3]).

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. [3] 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.

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 [2] 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[15]. 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 [20] 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. [17] 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 [28] 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. [26] 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 [9] 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. [13] 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. [8] 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. [22] 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.

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 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.

1. We measure all available model revisions, using two software complexity and two size metrics, and get quantitative data in form of four time series;
2. We employ outlier detection approaches thereto and get four lists of outlying revisions for each simulation model;
3. We split the measurement results equally in two sets, a test set and a validation set;
4. 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;
5. 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
6. 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 [10], 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.

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 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.

3.2 Data Collection Procedure

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

3.2.1 Measurement

Two size and two complexity measurements as explained in Section 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 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.

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 3, for all 71 simulation models.

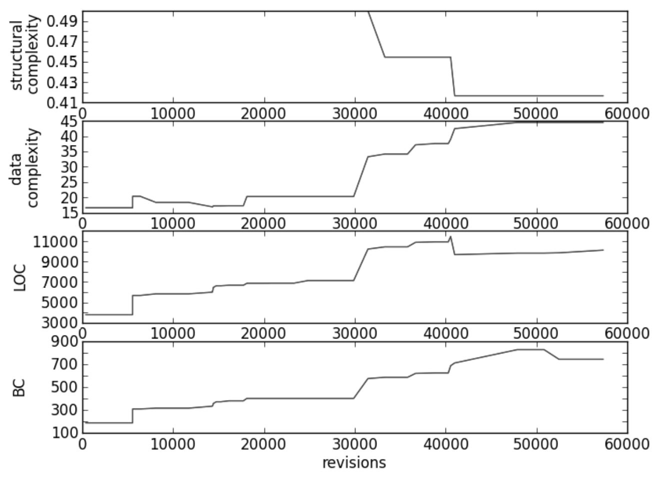

Figure 3: Example of measurement result for one anonymized model. It is illustrating a possible data set for the subsequent outlier detection. With revision numbers on the x-axis and measurement values on the y-axes covering a duration of almost four years.

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.

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 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

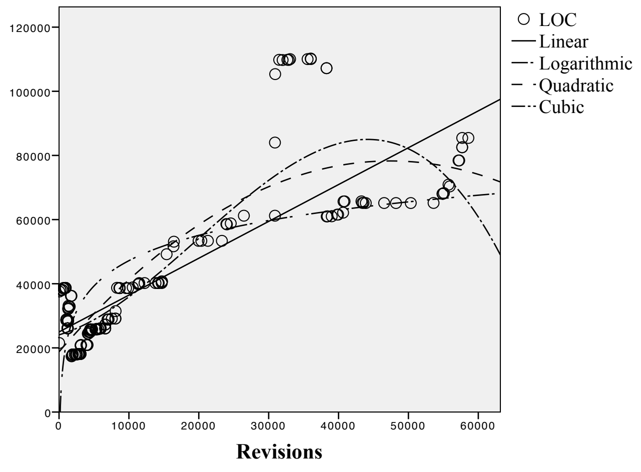

Figure 4: Common models fitted to example LOC measurement data.

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 shows a MA(2) model fitted to the same data set as used before. Comparing the MA(2) model with

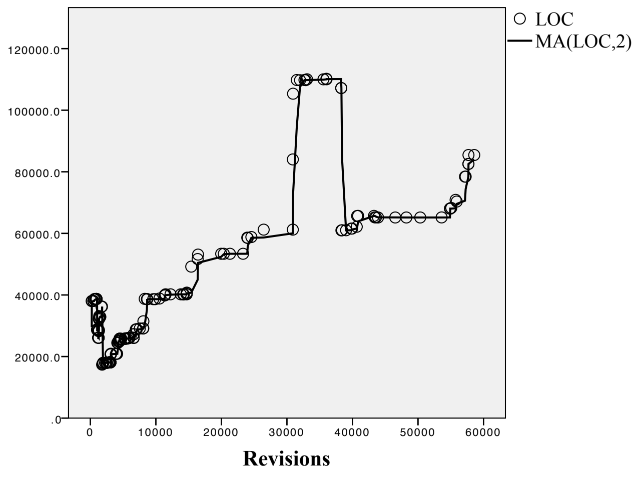

Figure 5: MA(2) model fitted to example LOC measurement data

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 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 6 shows residuals as they are received during the analysis for the MA(2) model fit to the example data shown in Figure 5. Blue circles represent SDR

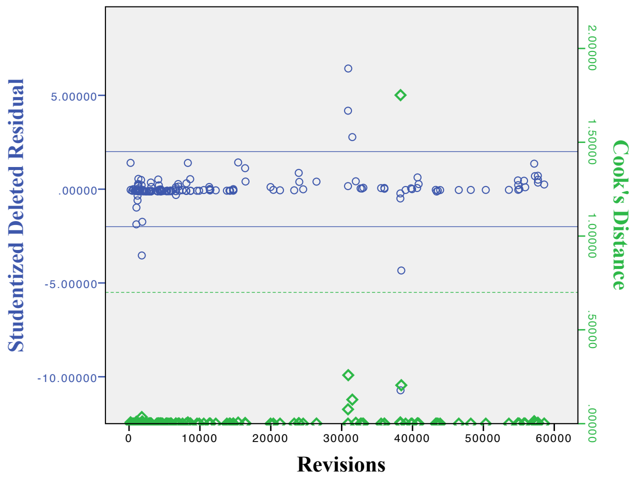

Figure 6: SDR and Cook's D for the example model fit in Figure 5

and green squares Cook's D. Outlier thresholds are calculated as described in Section 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.

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 [18]. The interviews were conducted in a semi-structured manner, following the guidelines from Shull et al. [27]. 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:

1. What happened?
2. What was the reason for the changes in that revision?
3. 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
4. Are there revisions related to this one; in this or other simulation models?
5. 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.

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 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:

1. correlation between raw outlier data and impacts;
2. correlation between combinations of outlier data and impact;
3. correlations between principal components among the outlier data and impact;
4. dependencies between outliers and descriptions and reasons for the revision provided by the engineers; and
5. 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.

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.

Table 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 [18], 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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

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.

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 3.2.2. On these 139 data sets, correlation observations were performed. Table 2 shows correlation results using Spearman's correlation coefficient. For 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 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 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. 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 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

Table 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 3: Component matrix showing PCA results. The abbreviations are explained in Table 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 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%

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 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. [19]). 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 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.

6. THREATS TO VALIDITY

We split threats to validity according to categories from Feldt & Magazinius [11]. 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.

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.

8. REFERENCES

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

[2] O. Alan and C. Catal. Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. *Expert Systems w Applications*, 38(4):3440–3445, 2011.

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

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

[5] L. C. Briand, S. Morasca, and V. R. Basili. Property-based software engineering measurement. *IEEE Trans. Softw. Eng.*, 22(1):68–86, 1996.

[6] D. Card and W. Agresti. Measuring software design complexity. *Journal of Systems and Software*, 8(3):185 – 197, 1988.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[8] 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*, MSR, pages 280–291, 2015.

[9] 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*, WCRE, pages 155–164, 2003.

[10] K. M. Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, 1989.

[11] 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*, pages 374–379, 2010.

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

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

[14] M. Gupta, J. Gao, C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Trans. on Knowledge and Data Engineering*, 26(9):2250–2267, Sept 2014.

[15] S. Hangal and M. S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE, pages 291–301, 2002.

[16] 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*, 13(4):543–559, 1980.

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

[18] ISO/IEC. Information technology - Software product quality - Part 1: Quality model, ISO/IEC FDIS 9126-1:2000(E). 2000.

[19] 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*, 2(3):137–143, 2009.

[20] 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.

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

[22] 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, Valencia, Spain, June 17-21, 2013. Proceedings*, chapter Profiling Event Logs to Configure Risk Indicators for Process Delays, pages 465–481. Springer Berlin Heidelberg, 2013.

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

[24] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, Dec. 2008.

[25] 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*, WASA, pages 39–42, 2015.

[26] 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)*, pages 35–40, 2015.

[27] F. Shull, J. Singer, and D. I. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer-Verlag New York, Secaucus, NJ, USA, 2007.

[28] 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*, MSR 2014, pages 72–81, 2014.

[29] E. Weyuker. Evaluating software complexity measures. *IEEE Trans. Softw. Eng.*, 14(9):1357–1365, Sep 1988.

[30] H. Zuse. Properties of software measures. *Software Quality Journal*, 1(4):225–260, 1992.