Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1563

Automated Continuous Evaluation of AUTOSAR Software Architecture for Complex Powertrain Systems

Hariharan Venkitachalam[1], Kalkin Powale[2], Christian Granrath[3] und Johannes Richenhagen[4]

Abstract: Connectivity of the vehicle to heterogeneous information sources is one of the key factors which lead to complexity of automotive software architecture. The information content from external communication sources modifies the structure, content and the synchronization of control algorithms. In addition to this, time and cost constraints for software development are challenging. With increasing complexity and reduction of development time, ensuring software quality is one of the foremost priorities of vehicle manufacturers. Software architecture plays an important role in ensuring quality by implementing design principles which enhance non-functional quality attributes of the automotive software. The extent to which a software architecture definition fulfills the quality requirements is not verified at early stages of development. As a consequence, design problems are transferred to later stages of development thereby causing rework of software artifacts. The paper focuses on a tool-based evaluation of non-functional quality characteristics using the concept of Continuous Integration for AUTOSAR-based transmission control software. The suggested approach enables early and continuous evaluation of software architecture thereby improving software quality

Keywords: Software Architecture, Software Quality, AUTOSAR, Continuous Integration, Agile Methods, Frontloading

1 Introduction

The complexity of the powertrain software architecture is increasing exponentially [Da13] [VWR16a]. The major reasons for this complexity are emission legislation, efficiency, a large number of variants, connectivity of data to the powertrain and automated driving. A stringent emission legislation has led to the electrification of the powertrain which involves coordination between various powertrain components. The advent of Car-2-Car, Car-2-X, and cloud-based technologies have added more data inputs to the powertrain which help in optimizing fuel consumption and reduce greenhouse gases[Ri15]. The synchronization between powertrain control algorithms and external data sources adds complexity due to timing constraints. With the evolution of autonomous vehicles, a large

[1]RWTH Aachen, Institute for Combustion Engines, Forckenbeckstrasse, 52074 Aachen, venkitachalam@vka.rwth-aachen.de
[2]TU Chemnitz, Department of Computer Science, Strasse der Nationen 62, 09111 Chemnitz, kalkin.p@gmail.com
[3] RWTH Aachen University, Institute for Combustion Engines, Forckenbeckstrasse, 52074 Aachen, granrath@vka.rwth-aachen.de
[4] FEV GmbH, Embedded Software Systems, Neuenhofstrasse 181, 52078 Aachen, richenhagen@fev.com

doi:10.18420/in2017_156

portion of the mission-critical functionality will be realized using software [Mc16].Thus, ensuring a high-quality software is important for the adoption of new technologies in the automotive software domain. The increase in software complexity makes the software difficult to maintain, test and to identify quality defects. Market demand has reduced the time available for the development of automotive control software [Ek15], [KK14]. As a result of this, software quality defects in the vehicle are increasing [SR16]. Given the cost of rework and warranty costs involved in fixing software quality defects, it is important to identify those at the early stages of development [BP88].

1.1 Research Questions

Software architecture is one of the earliest activities in the development lifecycle which involves decomposition of a complex functionality into smaller and manageable parts. The architectural decisions made at the start of the project have a deep impact on structural and dynamic aspects of the software. In the past few years, AUTOSAR has emerged as a standard way to express software architecture, ensure portability of software components and enable software sharing between suppliers and vehicle manufacturers [AU].

The increasing complexity of software architecture has resulted in a lack of consistent overview of software architecture. Reduction of the development time has caused "quick-and-dirty" changes to be included in the software. As a result, quality issues were detected at later stages of development. The automotive software also has to comply with a large number of standards [IS11c], [IS11a]. This leads us to the research question (RQ1)

- RQ1: What are the quality requirements of software architecture in the powertrain domain?

AUTOSAR standardized the interfaces to the basic software and workflow which is supported today by various commercial tools [AU]. However, the impact of the architectural design decisions (for e.g. functional decomposition, interface definition) on software quality of AUTOSAR software components has not been explored in detail. Current evaluation methods for software architecture focus on the manual evaluation of the architecture drafts and design guidelines. Given the high complexity of powertrain software architecture, a manual evaluation of the software quality is difficult. Software architecture is also evolving with the increasing complexity. In the absence of an objective and continuous method to evaluate the software architectural quality, design flaws are passed on the next stages of development. These design flaws lead to rework of software artifacts thereby, increasing development cost. Thus, a continuous evolution of software architecture requires a high-frequency evaluation of software quality attributes. This leads us to the second research question (RQ2)

- RQ2: How to continuously evaluate the quality of AUTOSAR software architecture?

1.2 State-of-the-Art

AUTOSAR proposes a layered architecture can be broadly classified into three layers at the highest abstraction level namely - Basic Software (BSW), Real Time Environment (RTE) and Application Layer (APSW) [AU]. The basic software provides a communication framework for the software. The Real Time Environment (RTE) provides communication services to the application software which makes the AUTOSAR software components independent from the mapping to a specific ECU. The application layer (APSW) consists of the control algorithms which are implemented on top of the AUTOSAR framework which is implemented as compositions, components, and runnables. The analysis of the software architecture is focused on the application software.

AUTOSAR focuses on non-functional quality characteristics such as portability of software architecture. However, there are many other quality criteria that need to be fulfilled to ensure software quality (for e.g. testability, maintainability, etc.) that are not directly addressed by AUTOSAR software architecture. The existing validation tools focus on verification of the syntax of the architecture specification. However, they do not address the relationship between the architectural parameters and software quality. This paper focuses on addressing this technological gap which will enable early assessment of AUTOSAR-based application software.

There exist two fundamental approaches to evaluate software architecture namely- scenario-based [Cl09] and metric-based methods. The scenario-based method involves subjective evaluation of software architecture quality based on scenarios which are created by key stakeholders involved in the project with or without a formal architecture specification [Cl09]. The reproducibility of the results from the scenario-based approach is not ensured [VWR16a]. The metric-based approach, on the other hand, needs a semi-formal or formal architecture specification but ensures reproducible results.

2 Methodology

The authors propose a “metric-based continuous evaluation” of powertrain software architecture to perform early, objective and repeated evaluation of software architecture. The fundamental approach is based on the previous works of the author [VWR16a], [Ve17]. This paper deals with the extension of the metric-based evaluation concept for AUTOSAR architectures. In general, the metric-based approach proposed consists of the following steps which are described below

- Definition of the quality model and metrics
- Continuous measurement and evaluation of metrics
- Architecture design improvement using metrics

2.1 Definition of quality model and metrics

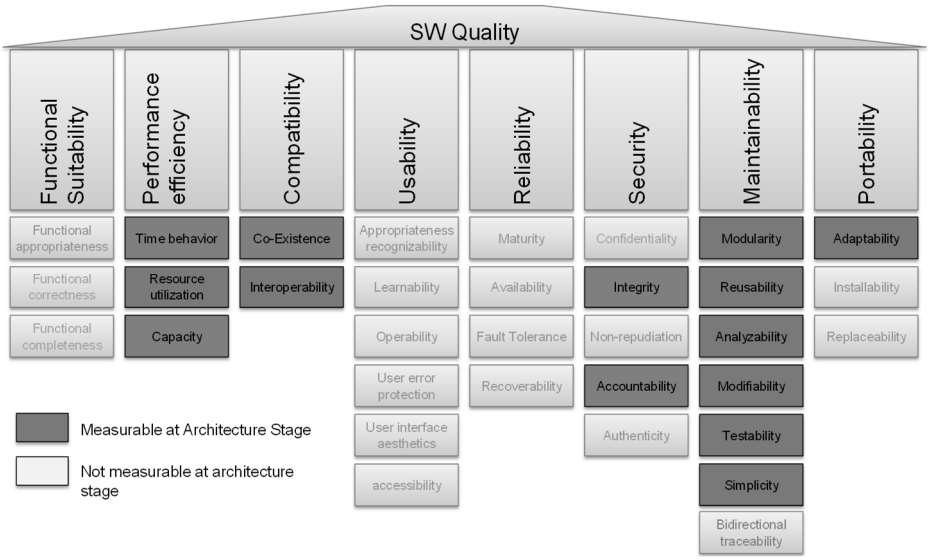

Fig. 1: Quality model for AUTOSAR-based software architectures based on ISO 25010 standard [IS11c]

The first step of architecture evaluation is the definition of the quality model. The key research question that we tackle here in this section is

- RQ1: What are the quality requirements of software architecture in the powertrain domain?

The Goal-Question-Metric approach is followed to define the metrics based on quality criteria [Ba92]. The quality criteria which are relevant for software architecture are described in a quality model which is based on the ISO 25010 and ISO 26262 standards [IS11c], [IS11a], [Ve17]. In addition to this, there are certain organizational goals which need to be considered like reusability of software components [Ri13], [Wi03], [Ha03], modularity [Pa72], [Ri13], [Dr08]. Thus, a quality model is created by considering the standards and the organizational requirements. There exist various models which describe software quality attributes [FHR08], [Wa10].

Only a few quality characteristics can be measured at the early stages of software development. The quality characteristics which are influenced by the software architecture were identified by literature survey [Ri14], [BCK13], [Sc02], [Wa96], [Bo07], [Fr03], [LL13]. The next step is to identify the mapping between the quality criteria and architectural parameters. The extent to which these quality attributes can be measured using the architectural parameters at early stages of development is evaluated. Based on this, we conclude upon the metrics to be used to evaluate software quality based

on the abstraction levels.

2.2 Continuous measurement and evaluation of software architecture

The complexity of powertrain functionality and the reduced time available for the evaluation of architecture has necessitated a tool-based approach to measuring software architectural quality. The sections 2.2 and 2.3 deal with the research question

- RQ2: How to continuously evaluate the quality of AUTOSAR software architecture?

The proposed tool-framework has two major parts which deal with the measurement and evaluation of software architecture. The first step involves extraction of the architectural information from the ARXML file format. The information extracted from the ARXML file is based on the architectural parameters which are required to compute the metrics. The measurement framework uses the extracted architectural information from various powertrain software projects to calculate metrics (defined in in Section 2.1) based on quality criteria.

The evaluation framework uses the measurement information from various projects and calculates thresholds based on statistical data. The outcome of the metric evaluation can vary based on the definition of thresholds. The fulfillment of quality criteria cannot be identified by a single metric. Hence, a combination of various metrics is required to identify the fulfillment of quality criteria. The evaluation framework aggregates and assigns weights for the metrics based on the quality criteria. Depending upon the importance of quality criteria, sufficient weights are assigned to various quality criteria. A detailed explanation of the evaluation framework is covered in the previous works of the author [Ve17]. This paper focuses on extending the measurement framework based on AUTOSAR concepts. The metrics are developed on a MATLAB/Simulink® tool-framework.

As mentioned before, the core focus of this work is to extend the measurement of quality for AUTOSAR-software architectures. Hence, the concept extension impacts only the measurement framework (see Fig. 2). ARXML, which is the standardized file format to describe software architecture, is used as the artifact to extract architectural information. In the previous works of the author, the focus was on structural aspects. However, the ARXML file format provides detailed information regarding timing and memory consumption in the software architecture. Hence these quality characteristics have been evaluated.

2.3 Architecture Design Improvement

The understanding of the relationship between architectural parameters and quality characteristics helps us suggest improvement measures for the software architecture. Tab.

1 shows some of the metrics, their design consequences and improvement measures for a better software architecture design. For each metric, such a design consequence and counter-measures are suggested.

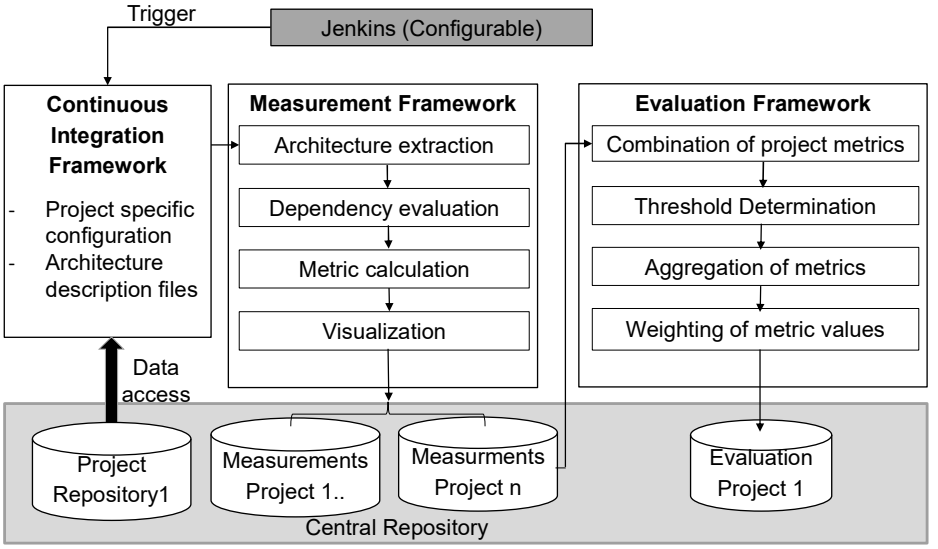

Fig. 2: Tool-framework used for the continuous evaluation of software architecture using metrics

3 Validation of architecture metrics

The metric-based architecture evaluation concept was applied to an AUTOSAR-based transmission control unit software. The current realization is validated for the ARXML file generated from Systemdesk® based on the AUTOSAR 4.0 specification. However, the measurement framework can also evaluate the ARXML files which are generated by other tools. Since a single transmission control unit project has been considered for the evaluation of the metric concept, a statistical evaluation is not feasible. Hence, an empirical evaluation of the metric values with software quality criteria is performed by empirical validation with experts. In the case of a larger dataset, the statistical evaluation as shown in Fig. 2 can be used. The empirical validation of the metrics concept suggested by the authors consists of a macro-level validation which evaluates whether the approach tackles the key problems of handling complexity, ensuring software quality and improving development efficiency. At a micro-level, the extent to which the architecture metrics concept deals with these problems in detail is evaluated.

3.1 Handling complexity of software architectures

The tool-based approach was capable of handling a transmission control unit with 62 application software components and more than 300 signals received from the CAN bus. The correctness of the results generated by the tool was manually verified for a set of software components. There is no restriction regarding the complexity of the architecture which can be handled by the tool-based approach.

3.2 Evaluating software quality

The second aspect of validation involves determining the usability of the metrics for determining quality. There exist three approaches to evaluate the usability of the architecture metric. One of the approaches is to correlate the metric with another established metric at later stages of development (model or code) [Ve17]. The second approach involves correlating the metric with defect density [Ku]. The third method involves empirical validation with experts. The third approach involves discussions with developers in the project to identify the conceptual relationship between problems faced in code generation and software integration and the metrics. Unlike other approaches, this is more conceptual in nature and established a causality between AUTOSAR concepts and software quality. Hence, this approach was adopted.

3.3 Reduction of development time and effort

A high degree of automation enables faster feedback loops for software development to respond to shorter development time and enables reduction of costs. The continuous integration framework provides a basis for implementing a technical solution which provides fast feedback loops for the evaluation of software. A traditional scenario-based analysis of software architecture is estimated to involve an effort of 70 man days [Cl09]. The engineering effort estimated for the conception, development, and deployment of architecture metrics is high. However, the applicability of metrics across a software product line or a group of projects reduces the effort over a large number of projects.

Metric	Design consequence	Improvement measures
Number of runnables per software component	Latency time of a component is affected by using a large number of runnables. Depending upon the implementation, there are two categories of runnables namely- basic and extended runnable entities depending upon whether they contain a "wait point" or not. The mapping of extended runnable entities to a single task leads to jitter which affects the time behavior.	The number of runnables per software component must be reduced to reduce the complexity of the software component. A threshold of 3 runnables per component needs to be set for the software architecture to improve understandability of the software component.
Number of inter-runnable variables per software component	Inter-runnable variables are in general protected by critical sections. The latency time increases due to a large number of inter-runnable variables.	The software developers propose a design with less inter-runnable variables for safety-critical applications. This improves the time behavior of the software components.
Fan-in (adapted for AUTOSAR from [HK81])	A large number of fan-in would mean that there are a large number of components coupled to the input side of a software component. This reduces the testability of the software component.	The coupling of components to various software components must be reduced by grouping of functionally or logically cohesive software components.

Tab. 1: Example of design consequences and architecture improvement using metrics

4 Results

The focus of the work is to develop a tool-based evaluation methodology for AUTOSAR software architectures. Fig. 1 shows the quality model for the powertrain software architecture based on the first research question (RQ1). Fig. 2 shows the tool architecture for continuous evaluation of software architecture based on the second research question

The solution was based on two key research questions. The first research question (RQ1) focussed on the development of a quality model for software architecture which outlined

those quality attributes which can be evaluated at early stages of development. The quality goals were derived from various standards like ISO 25010[IS11a] and ISO 26262[IS11b]. Fig. 1 shows the quality goals for software architecture which can be measured at the early stages of development.

The next key research question (RQ2) focussed on the evaluation of the software architecture continuously. The quality metrics defined were programmed into the continuous integration environment (as shown in Fig. 2). These metrics were evaluated for a transmission control unit software. The metrics are based on the abstraction level of the elements in the software architecture. Fig. 3 shows some of the metric results at the abstraction level of the AUTOSAR software components. Tab. 2 shows some of the metrics at the system level including various electronic control units in the architecture based on the ARXML file. These results at various abstraction levels are used to identify improvement measures for the software architecture similar to Tab. 1.

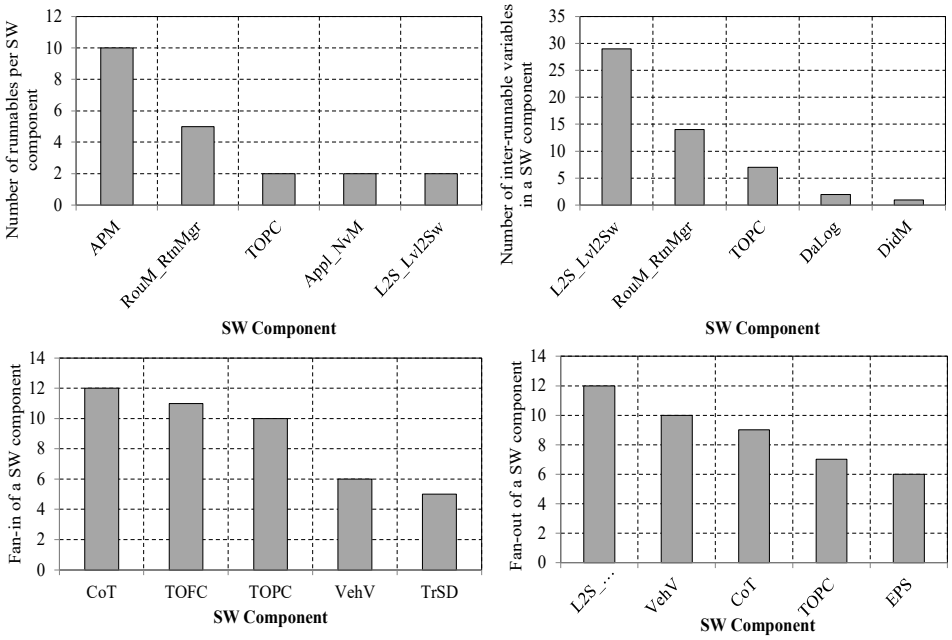

Fig. 3: Metric results for an AUTOSAR-based transmission control unit

Description of the system level metric	Measured values
Total number of COM signals	324
Number of OS tasks	67
Number of Application SW components	62

Number of service SW components	8
Number of complex device drivers	3

Tab. 2: Metrics at the system level derived from the ARXML file with the measured values.

5 Conclusion and Future Outlook

Metric-based analysis of software architecture tackles some of the key problems faced related to quality and development time. Although AUTOSAR standardizes the interfaces to the basic software, aspects of testability, maintainability and their relationship with architectural parameters were not delved into. This approach fundamentally focuses on quantifying the impact of architectural decisions on software quality. The tool-based implementation provides a framework to handle the architectural complexity of the powertrain and provides feedback regarding the quality attributes. The continuous integration framework provides the possibility to evaluate software architecture after modifications thereby reducing the cost of rework. Hence, it supports the agile development of software functionalities for the vehicle without compromising on software architectural principles.

Definition of threshold values for a large number of AUTOSAR-metrics is still an open topic. The evaluation framework proposed by the authors in their previous work is based on statistical data from multiple projects. Since the availability of statistical data is limited, the approach is to define empirical thresholds for various metrics based on consultation with experts. These empirical thresholds will form the basis for tracking the fulfillment of design guidelines for future AUTOSAR projects.

References

[BCK13] Bass, L.; Clements, P.; Kazman, R.: Software architecture in practice. Addison-Wesley, Upper Saddle River, NJ, 2013.

[Bo07] Bosch, J.: Design and use of software architectures. Adopting and evolving a product line approach. Addison-Wesley, Harlow [u.a.], 2007.

[BP88] Boehm, B. W.; Papaccio, P. N.: Understanding and controlling software costs. In IEEE Transactions on Software Engineering, 1988, 14; pp. 1462–1477.

[Cl09] Clements, P.: Evaluating Software Architectures. Methods and Case Studies. Addison-Wesley, 2009.

[Da13] Davey, C.: Automotive Software Systems Complexity. Challenges and Opportunities, 2013.

[Dr08] Dressler, J.M.: A Walk Through EMS2010 Modular Software Development, 2008.

[Ek15] Eklund, U.: The future of Automotive Software Engineering, 2015.

[FHR08] Fieber, F.; Huhn, M.; Rumpe, B.: Modellqualität als Indikator für Softwarequalität. Eine Taxonomie. In Informatik-Spektrum, 2008, 31; pp. 408–424.

[Fr03] Frankel, D.: Model driven architecture. Applying MDA to enterprise computing. Wiley, New York, 2003.

[Ha03] Hammel, C. et.al.: A Common Software Architecture for Diesel and Gasoline Engine Control Systems of the New Generation EDC/ME(D)17. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2003.

[HK81] Henry, S.; Kafura, D.: Software Structure Metrics Based on Information Flow. In IEEE Transactions on Software Engineering, 1981, SE-7; pp. 510–518.

[IS11a] ISO: ISO/IEC 25010:2011 Systems and software engineering -Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, 2011.

[IS11b] ISO: ISO/DIS 26262-Road vehicles – Functional safety- Part 6: Product development at the software level. International Standardization Organization, 2011.

[IS11c] ISO: ISO/DIS 26262-Road vehicles - Functional safety- Part 6: Product development at the software level. International Standardization Organization, 2011.

[KK14] Katumba, B.; Knauss, E.: Agile Development in Automotive Software Development: Challenges and Opportunities. In (Jedlitschka, A. Ed.): Product-focused software process improvement. 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014 proceedings. Springer, Cham, 2014; pp. 33–47.

[LL13] Ludewig, J.; Lichter, H.: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken. dpunkt-Verl., Heidelberg, 2013.

[Mc16] McKinsey: Numetrics R&D Analytics. Introduction.

[Pa72] Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Modules, 1972.

[Ri13] Richenhagen, J.: Development of modular powertrain controls with continuous integration, 2013.

[Ri14] Richenhagen, J.: Entwicklung von Steuerungs-Software für den automobilen Antriebsstrang mit agilen Methoden. Dissertation, Aachen, 2014.

[Ri15] Richenhagen, J. et al.: PERSIST – A scalable software architecture for the control of diverse automotive hybrid topologies. In (Bargende, M.; Reuss, H.-C.; Wiedemann, J. Eds.): 15. Internationales Stuttgarter Symposium. Automobil- und Motorentechnik. Springer Vieweg, Wiesbaden, 2015; pp. 37–56.

[Sc02] Schmidt, D. C.: Pattern-orientierte Software-Architektur. Muster für nebenläufige und vernetzte Objekte. dpunkt-Verl., Heidelberg, 2002.

[SR16] SRR Global Financial Advisory Services: Industry Insights for the road ahead. Automotive Warranty and Recall Report 2016.

[Ve17] Venkitachalam, H. et al.: Metric-based Evaluation of Powertrain Software Architecture. In SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 2017, 10.

[VWR16a] Venkitachalam, H.; Wissel, D. von; Richenhagen, J.: Metric-based Evaluation of Software Architecture for an Engine Management System. In SAE International Journal of Engines, 2016, 9.

[Wa10] Wagner, S. et al.: Softwarequalitätsmodelle – Praxisempfehlungen und Forschungsagenda. In Informatik-Spektrum, 2010, 33; pp. 37–44.

[Wa96] Wasserman, A. I.: Toward a discipline of software engineering. In IEEE Software, 1996, 13; pp. 23–31.

[Wi03] Williams, C.: Algorithms, Algorithm Modeling, Software, & Software Architecture. http://www.eecs.umich.edu/courses/eecs486/win03/notes/GMVisit.pdf.

[Ku] Kugler, C. et al.: Metrics-based strategies for quality assurance of automotive embedded software. In (Bargende, M.; Reuss, H.-C.; Wiedemann, J. Eds.): 17. Internationales Stuttgarter Symposium. Automobil- und Motorentechnik. Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2017; pp. 711–730.

[AU] AUTOSAR: Layered Software Architecture. Version 4.2.2. www.autosar.org, accessed 2017.

[Ba92] Basili, V.: Software Modelling and Measurement. The Goal/Question/Metric Paradigm, report CS-TR-2956, Department of Computer Science, University of Maryland, College Park, 1992