Towards Metrics for Analyzing System Architectures Modeled with EAST-ADL

Christoph Etzel, Florian Hofhammer and Bernhard Bauer
Institute of Computer Science, University of Augsburg, Germany
{*christoph.etzel, bauer*}*@informatik.uni-augsburg.de, florian.hofhammer@student.uni-augsburg.de*

Keywords: System Architecture Metrics, Model-based Systems Engineering, Automotive Systems Engineering, EAST-ADL.

Abstract: Quality of system and software architectures plays a major role in today's development to reduce the complexity of systems during design and ensure maintainability and extensibility. Metrics can provide system architects with information about the quality of the system architectures and support them to ensure the quality characteristics defined in ISO 25010. In this position paper, we look on selected metrics from code and object-oriented design analysis to use them for the evaluation of system architectures modeled with EAST-ADL. Therefore, the metrics (collections) *Lines of Code*, *Cyclomatic Complexity*, *Chidamber and Kemerer* and *MOOD* are examined for their application on EAST-ADL models.

1 INTRODUCTION

Since the 1970s, software quality became more and more important in the course of the evolution of software development towards an engineering discipline. For this reason, metrics were developed to enable comparability of the software systems quality as key figures. The comparison of different systems as well as of revisions of a single system played are use cases. At the time, quality was mainly defined and measured with regard to the prevailing imperative programming paradigms. The change to object-oriented programming then required fundamental changes and extensions of the previously defined key figures. Since the publication and subsequent distribution of the Unified Modelling Language (UML), extensions emerged with the possibility not only to measure code quality, but also to analyze models and architectures already before the implementation.

Today it is important not only to examine pure software architectures and measure their quality, but also to consider system architectures (more precisely, functional and logical architectures). A prominent example for complex system architectures are modern automobiles, which contain various electronic control units, sensors, actuators and entertainment systems, communicating with each other by an interaction of software and hardware. Among others, the architecture description language EAST-ADL can be used to model such systems.

In the context of the Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL), literature lacks of concrete metrics for measuring the quality of modeled system architectures. In this position paper, we present some potential metrics to narrow this gap. If available, such quality measuring metrics can then be used in review processes like Architecture Tradeoff Analysis Method (ATAM) and support the decision-making. Therefore, approaches of quality analysis from the software development are studied and afterwards possibilities for the adjustment and application of these on EAST-ADL models are pointed out. The aim is to form the basis for empirical evaluations of the quality of system architectures (functional/logical architecture) and thus to lay the foundation for an automated quality analysis of such models. In the context of this position paper, explicitly no extensive evaluations and classifications of applied metrics are made, but possible interpretations for the analysis are pointed out. Furthermore, currently only metrics from software development are considered and none from hardware design.

First, this paper explains the basic structure of the modeling language EAST-ADL. After a clarification of the quality concept in software development, metrics for the analysis of code and object-oriented software designs are presented and transferred to EAST-ADL models where possible. We conclude with a summary and outlook on further steps that can follow this work.

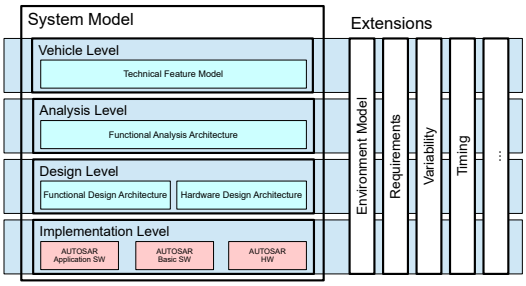

Figure 1: EAST-ADL abstraction levels with the containing models and extensions (based on (Blom et al., 2016)).

2 EAST-ADL

The Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) is developed and administered by the EAST-ADL Association. The current version of the specification is 2.1.12 (EAST-ADL Association, 2013). This architecture description language can be considered an extension of the UML and a specialization of Systems Modelling Language (SysML) (Blom et al., 2016).

EAST-ADL allows to comprehensively represent and model the electronic systems of a vehicle, their features and characteristics, the requirements and the communication from the hardware to the software level (Blom et al., 2016). The system is described in several horizontal levels, which are extended by vertical layers (*Extensions*) to model concerns on different degrees of abstraction.

The horizontal structure consists of the *System Model* (see Figure 1) and provides the means to model system architectures. This is supplemented by the *Extensions* with additional modeling possibilities. These are of particular importance with regard to ISO 26262, for example. ISO 26262 describes the functional safety requirements for road vehicles (ISO, 2018). Using the EAST-ADL extensions (e.g. Requirements, Dependability), modeling means can be used to ensure that safety requirements are documented and met by model (Blom et al., 2016). The *Environment Model* describes the external influences on the system. For example, mechanical and hydraulic systems in the vehicle, the environment including road surface, adjacent vehicles and traffic information systems can be included (Blom et al., 2016).

In this paper, just the architectures on *Analysis* and *Design Level* of the system model are further investigated, since these contain functional architectures. The *System Model* itself is divided vertically into four levels, which abstract the system to different degrees. At the *Vehicle Level*, the vehicle is described from a feature perspective. The *Analysis Level* provides the *FunctionalAnalysisArchitecture (FAA)* that defines systems from a functional perspective. The architecture describes "which" functions are required and which data they exchange. Figure 1 shows the *Design Level* further subdivided into the *FunctionalDesignArchitecture (FDA)* and the *HardwareDesignArchitecture (HDA)*. On this level, the FDA models "how" functions are realized from a more technical viewpoint. However, no concrete software elements are modeled. The hardware design architecture, on the other hand, represents the structure of the system in hardware so that different functions from the FDA can then be mapped to the hardware entities of the HDA. Finally, the *Implementation Level* models the actual implementation of the software and hardware. This level is not defined by the EAST-ADL, but by AUTomotive Open System ARchitecture (AUTOSAR). This part is outside the definitions of the EAST-ADL and not analyzed in this paper.

3 THE CONCEPT OF SOFTWARE QUALITY

Ensuring high software quality is an important part of software engineering and plays a major role in reducing the time and cost of developing a software project (Liggesmeyer, 2009). Therefore, more and more attention is placed on this and quality assurance techniques have been developed that deliver comparable results as far as possible, e.g. an absolute numerical value as a result of the application of a quality metric.

When talking about software quality, reference is often made to the ISO 9000 series of standards, which generally provides standards for quality management, and specifically to the ISO/IEC 9126-1:2001 (ISO 9126) standard, as the mention in literature shows: (Liggesmeyer, 2009), (Sneed et al., 2010), (Abran, 2010), (Hoffmann, 2013), (Kan, 2003). ISO 9126 defines criteria for product quality in software development and thus criteria for software quality. It has been replaced by the standard ISO/IEC 25010:2011 (ISO 25010) in 2011 (ISO, 2011).

Since this is a standardized and globally recognized definition of the quality of a software product, the content of this standard is used in the following as criteria for software quality.

Quality Characteristics According to ISO 25010. According to ISO 25010, there are both quality characteristics for software in use and for product quality.

The first grouping refers to the interaction of users with the software and to what degree it meets the specific requirements. The quality characteristics for the software as a product defines quality independently of the user, based on objective requirements (ISO, 2011). This also means that product quality can be determined statically without user interaction. Therefore, only this part of the ISO standard is considered in this paper, since the quality of an architecture is to be determined and not the quality of the final product (usefulness, freedom from health risks ...).

The criteria for product quality can be divided into eight characteristics according to ISO 25010: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability. These are further divided into sub-criteria.

4 METRICS FOR EAST-ADL

In the previous section, ISO 25010 was introduced as a foundation for the definition of software quality. Now it is a matter of measuring this software quality. For this purpose, many metrics have been published in literature, which are supposed to summarize the still abstract concept of quality in numbers. In this section, metrics from code analysis and object-oriented design analysis are presented to calculate key values, which can be used to evaluate architectures according to the quality criteria. The selection of metrics is not intended to be exhaustive.

4.1 Selected Code Metrics

A large number of metrics refer to findings from the source code of a finished software product, which cannot provide direct findings in the case of the analysis of EAST-ADL architecture models. Nevertheless, for the sake of completeness and due to the great importance in literature, two metrics have been selected.

4.1.1 Lines of Code

Lines of Code (LOC) is a software metric for the source code size of a program. A simple connection is established: the higher the number source code line, the more extensive the software project. If the LOC for individual software modules or classes are calculated, conclusions can be drawn with the LOC as a basis, for example with regard to freedom from errors, by evaluating a correlation between LOC and the number of errors. Mapping the metric to quality characteristics of the ISO 25010, high values may negatively influence the functional suitability, reliability and maintainability.

With regard to the freedom from errors, particularly small or particularly large modules usually have more errors per LOC. Since the number of external interfaces of a module is usually relatively constant regardless of its size, errors in the interfaces are of greater impact for smaller modules or classes. Large modules become more confusing and difficult for developers to control, which is why more errors creep in. (Kan, 2003)

This metric is supposed to be easy to measure, since you only have to count the lines of the source code files. In fact, this metric is much more complex and difficult to use, because the comparability of the values is usually not given. It depends on many variables like the programming language, programming style, coding guidelines for code formatting, to mention a few (Hoffmann, 2013) (Kan, 2003). In terms of quality, it is very difficult to draw comparable conclusions.

Transfer to EAST-ADL. Although one could measure the LOC of EAXML files, which contain EAST-ADL models in a standardized eXtensible Markup Language (XML) format, this would not provide any significant findings for the quality an architecture. On the one hand, such files generally contain more than the elements of the FAA, FDA and HDA, on the other hand the goal of the analysis of system architectures on the basis of their representative diagrams is missed. Therefore, the application of this metric to EAST-ADL does not provide any valuable insights.

4.1.2 Cyclomatic Complexity (McCabe)

In 1976, McCabe presented the *Cyclomatic Complexity* (McCabe, 1976), which is based on a completely different approach than LOC. The method does not analyze the program code itself, but creates a control flow graph from the code, from whose structure the complexity of the code could be calculated.

The formula for calculating the cyclomatic complexity is a slightly modified formula from graph theory, which describes the maximum number of linear independent cycles in strongly connected graphs (McCabe, 1976) (Liggesmeyer, 2009). The modification refers only to the insertion of an additional edge from the end node to the start node of the control flow graph, otherwise control flow graphs usually cannot be considered as strongly connected. The following terms apply in the further course of this section:

$G =$ control flow chart,

$G_i =$ single independent control flow graph,

$E =$ set of edges,

$N =$ set of nodes,

$n =$ # independent control flow graphs

The cyclomatic complexity of a single strongly connected control flow graph is indicated by the following formula:

$$V(G) = |E| - |N| + 2 \tag{1}$$

This formula results from

$$V(G) = |E| - |N| + 2n \tag{2}$$

with $n = 1$. This generalized form calculates the cyclomatic complexity of several independent control flow graphs (e.g. several methods of a class).

$$V(G) = |E| - |N| + 2n \quad = \sum_{i=1}^{n} V(G_i) \tag{3}$$

Thus, the calculation of the cyclomatic complexity of a network of functions and methods is only as complex as the calculation of the cyclomatic number for the independent graphs of the individual functions and methods.

During the calculation, it becomes clear that the cyclomatic number does not change when sequential instructions are inserted, since in this case the additional edge is compensated by the additional node. The same applies when removing instructions from sequential program parts (McCabe, 1976) (Hoffmann, 2013). Therefore, in principle, no causality can be established between program size and cyclomatic number, even if a certain correlation often occurs (Kan, 2003). A similar correlation can also be observed between the number of errors in the program and the cyclomatic complexity, whereby here too a causality cannot be proven (Abran, 2010) (Kan, 2003).

Nevertheless, McCabe suggests a maximum cyclomatic complexity of 10 (McCabe, 1976). This automatically limits the number of loops and decisions because, unlike sequential statements, they insert more new edges than nodes into a control flow graph. This approach should keep the code readable and thus enable a more efficient and error-free implementation. In addition, this metric simplifies the identification of code sections that have been implemented in an excessively complex way (Kan, 2003). The term “complex” is to be understood as “with many loops and decisions” . Since the cyclomatic complexity specifies the maximum number of test cases for a complete branch coverage, this metric is also suitable as the upper bound of the effort estimate for branch coverage tests (Liggesmeyer, 2009). Thus, the cyclomatic complexity has a direct influence on the ISO 25010 maintainability quality criteria.

Transfer to EAST-ADL. The calculation of the cyclomatic complexity is not directly transferable to EAST-ADL models, because there is no code with control flows in the architectures modeled. Nevertheless, this metric is of interest in the context of EAST-ADL models.

If one considers the representation of a FAA, FDA or HDA as control flow graphs, the calculation rules could be applied. However, this requires redefinitions, because these models are not control flow graphs but data flow graphs. In the considered EAST-ADL models, connections between entities define mostly directed information flows, which justifies this adaptation. Thus, the entities of the models are not states in the program flow, but states in the data flow.

The problem occurs, however, that entities in EAST-ADL can have several incoming and outgoing edges, since information can enter or being sent at the same time. This could be seen in a control flow graph as an expression of parallelism or concurrency. However, the cyclomatic complexity only considers sequential program flows (de Paoli and Morasca, 1990). In their paper, De Paoli and Morasca propose the use of Petri nets instead of normal control flow graphs and based on this an extension of the cyclomatic complexity to parallel software. The parallelism is modeled by several incoming and outgoing edges in and from transitions.

The necessary intermediate step is a transfer of an EAST-ADL model like the FAA to a Petri net. For example using the functions of the model as places of the Petri net, every place could have exactly one incoming and one outgoing edge of exactly one transition. This transition can then have several incoming or outgoing edges.

A continuation of this consideration as well as its evaluation or other approaches for the transfer of EAST-ADL models to Petri nets, go beyond the scope of this paper. The application of the cyclomatic complexity to EAST-ADL architecture models needs further research.

4.2 Selected Object-oriented Metrics

In addition to the code metrics discussed so far, which mainly refer to code implemented with imperative programming paradigms, there are also many metrics published referring to object-oriented code or object-oriented models.

4.2.1 Chidamber and Kemerer

The metrics by Chidamber and Kemerer (Chidamber and Kemerer, 1994) have often been quoted in literature (Sneed et al., 2010). Most of these metrics can

be applied directly to models such as UML class or sequence diagrams, since they do not require a reference to the code for the calculation.

In this paper, we focus on the metrics *Coupling between Object Classes (CBO)* and *Response for a Class (RFC)*. The other metrics *Weighted Methods per Class (WMC)*, *Lack of Cohesion in Methods (LCOM)*, *Depth of Inheritance Tree (DIT)* and *Number of Children (NOC)* are not explained, since we see no portability to EAST-ADL models and omit them for reasons of space. See the original publication for further information.

Coupling between Object Classes (CBO). The metric CBO is supposed to be a measure for the coupling between classes. In (Chidamber and Kemerer, 1994) it is defined as the number of associations of the considered class to other classes, where in (Sneed et al., 2010) represents another interpretation of the metric using the formula

$$\text{CBO} = 1 - \frac{\text{\# classes with associations}}{\text{\# associations}} \tag{4}$$

In this way, they generalize the measure from one class to an entire system.

Chidamber and Kemerer present three possible interpretations for this metric. First, they write that too high coupling speaks against a modular design and limits the reusability of the class in question due to its many dependencies. Subsequently, they conclude from high coupling to high maintenance effort or poor maintainability, since the high coupling increases the sensitivity for changes in other parts of the software system. Finally, they state that an increased coupling also entails more testing effort. Therefore, the coupling should be kept as low as suitable in order to improve the ISO 25010 characteristics reusability, maintainability and testability.

Response for a Class (RFC). The RFC measure indicates how many methods can be called in response to receiving a message from an object. It is defined by the following formulas:

$$\text{RFC} = |\text{RS}| \tag{5}$$

$$\text{RS} = \{M\} \cup_{\text{all } i} \{R_i\} \tag{6}$$

Where RS is the so-called *Response Set*, which is defined as the methods M of the class to which the called object belongs, and the methods R_i, which can be called by the methods of the class. Thus, this set contains either directly or also transitively all those methods which can be called when receiving a message of the object, no matter whether as direct method call, when creating the object by the constructor or in other ways.

It should be noted, however, that the calculation of the metric is not recursively carried out for reasons of meaningfulness and practicability and therefore only the methods called directly by methods of the class in question are actually included in the set of R_i (Chidamber and Kemerer, 1994).

An alternative definition of the metric is provided by (Sneed et al., 2010):

$$\text{RFC} = 1 - \frac{\text{\# dynamic calls} + 1}{\text{\# potential target methods}} \tag{7}$$

The difference to the original definition by Chidamber and Kemerer is that not the number of potentially callable methods is measured, but a rational measure for the polymorphism is given. In this context, a dynamic call is one that can have several different target methods, for example, if the same method signature occurs with different implementations of the method body in connection with inheritance at several levels of the inheritance hierarchy. For this reason, the definition uses a subset and does not include all methods of a class and messages to it as in the original definition.

The idea of this RFC definition is that an increased polymorphism increases the complexity of the call, since it is less predictable which actual implementation of the method will be called. A similar relationship is established with the original definition. Chidamber and Kemerer describe that an increased value for RFC requires a better code understanding and makes testing and debugging more difficult (Chidamber and Kemerer, 1994). They therefore conclude that increasing the RFC value also increases the complexity of the class. Therefore, one can conclude that the value of this metric should be kept low. However, it does not make sense to keep the number of methods unnecessarily low, but an appropriate mean should be sought.

Transfer to EAST-ADL. As initial stated, some of measures can be excluded from an application to the FAA, FDA and HDA. This is because the considered parts of the EAST-ADL system models have no correspondences for methods and inheritance. Thus, the metrics WMC and LCOM are ruled out due to a lack of methods, since they are based only on instance variables, complexity or number of the methods. The same applies to the metrics DIT and NOC, which require an inheritance hierarchy for the calculation.

The metric CBO can be transferred to EAST-ADL models. All entities of the considered models are used as correspondences of classes. The connections

between the input and output interfaces of the entities are regarded here as connections. Since the measure of coupling aims to measure the information flow between individual entities or objects, this metric is meaningful on the levels of FAA and FDA, since the connections model the data flow. If, on the other hand, the HDA is considered, the meaningfulness of this metric can decrease, because in addition to data flow paths (bus systems), electrical power supply lines can also be modeled. From a functional data flow perspective, connections representing power lines should not have an influence on the coupling and therefore be excluded.

The RFC metric requires closer consideration, since it depends on knowledge of the methods of a class to calculate this metric. According to Chidamber and Kemerer, this metric is generally a measure for the potential communication between classes (Chidamber and Kemerer, 1994). Based on the idea of communication potential, a definition for the calculation of this metric can be derived for EAST-ADL models. The *RS* is not understood as a set of potentially callable methods inside and outside the class, but is redefined as the set of potentially usable message channels for sending messages as a reaction to incoming information. This corresponds to the idea that entities in the FAA, FDA and HDA react to something. For example, in the case of sensors in the HDA model, this can be environmental influences detected by the sensors. Or in the case of an AnalysisFunction of the FAA model, this can be data received through incoming connections. As a response, any entity with outgoing connections can then output processed data. Depending on the number of outgoing connections, the potential for outgoing data flows is different, so this modified metric provides an indication of this potential. In this definition, however, the number of outgoing messages per connection is irrelevant. This is equivalent to the original definition of Chidamber and Kemerer, where only the set of potentially callable methods plays a role and not the number of method calls.

Analogous to the original metrics of Chidamber and Kemerer it is advisable to keep the values of the CBO and RFC low. This makes individual functions or components more easily interchangeable, which makes the system easier to adapt to special deployment scenarios during development or, in the case of defective hardware components during run-time, easier to maintain.

4.2.2 MOOD

The set of *Metrics for Object-Oriented Design (MOOD)* contains metrics to determine the quality of an object-oriented software design. It was published in 1994 with a number of eight metrics (Abreu and Carapuça, 1994).

The eight metrics have in common that their values are all in $[0,1] \subset \mathbb{R}$ and therefore can be given as degrees from 0% to 100%. 0% stands for total absence and 100% for total presence of a property measured by the corresponding metric. Furthermore, the metrics do not refer to a single class (as the Chidamber and Kemerer metrics), but to the whole system. Since we did not transfer all metrics for the use with EAST-ADL models, we ommit the description of the *Method Hiding Factor (MHF)*, *Attribute Hiding Factor (AHF)*, *Method Inheritance Factor (MIF)*, *Attribute Inheritance Factor (AIF)*, *Polymorphism Factor (PF)* and *Reuse Factor (RF)*.

Coupling Factor (COF) and Clustering Factor (CLF). The *Coupling Factor (COF)* indicates how strong the coupling of the classes of the system is. The formula for this metric is

$$\text{COF} = \frac{\text{\# all refs from classes to other classes}}{n^2 - n} \quad (8)$$

where n is the number of classes in the system (Abreu and Carapuça, 1994). The term references must be understood here very comprehensively: according to the authors, for example, method calls can be summarized here just like global or local variables with reference to another class (Abreu and Carapuça, 1994).

As already stated for the metric CBO in Section 4.2.1, the coupling should be kept as low as suitable. Abreu and Carapuça also confirm this objective by writing that classes should communicate with each other as little as possible and if it is not avoidable, exchange as little data as possible. In addition, a high coupling would increase the complexity and would reduce the encapsulation, reusability, comprehensibility and maintainability of the code. This would either directly or indirectly counteract the goals of the ISO 25010 quality model (see Section 3).

The *Clustering Factor (CLF)* metric is similar, because it is a measure for the coupling in a system, albeit on a different level. A cluster of classes is to be understood as a coherent subgraph of a graph that represents the system, with the classes representing the nodes and any associations, references, or inheritances representing the edges between the nodes. This graph can consist of disjunctive and unconnected subgraphs, each such subgraph being a cluster. Thus, the CLF is defined by

$$\text{CLF} = \frac{\text{\# clusters}}{\text{\# classes in the system}} \quad (9)$$

It can be concluded from the definition that a higher CLF is desirable, since smaller clusters can be an indication of good function and data encapsulation in the individual clusters and better reusability due to less dependence on other classes.

Both COF and CLF are difficult to calculate from a component model or class diagram alone, since they are not only based on associations and inheritances, but also explicitly on references such as method calls in the code. One could limit the metrics to the calculation by means of information readable from class diagrams, but this falsifies the expressiveness of the metrics and should therefore not be understood as a fully-fledged alternative, but at most be used as a supplement to other, more expressive measures.

Transfer to EAST-ADL. The lack of correspondences for methods or inheritance in EAST-ADL often prevents the direct transfer of the MOOD metrics. Attributes can be provided to entities in the EAST-ADL architecture models, e.g. for electrical components of the HDA, but there exists no equivalent for the visibility of attributes. However, in cases where attributes are used to calculate metrics, the visibility or inheritance of the attributes plays a role. For these reasons the metrics MHF, AHF, MIF, AIF and PF are excluded for further consideration. In addition, the number of library classes is used for the calculation of RF. Although there are packages defined in EAST-ADL, they are intended to structure the entire EAST-ADL model in a human understandable way and should therefore not be understood as packaging in the sense of software libraries.

In contrast, the CLF can be transferred without problems. As in the original definition, the disjoint, coherent subgraphs are understood as clusters. Classes correspond in this interpretation to the elements of the considered architecture model. Thus, the value of this metric can be calculated as the quotient of the number of clusters and the number of elements in the model. As in the version intended for software architectures, it can also be concluded here, that a larger value is desirable for the CLF in order to keep the dependency on other elements.

Finally, the coupling is measured by the COF, which is originally defined by the references between classes and the number of classes. The entities contained in an EAST-ADL model can be used as classes again. With the references, however, a new definition of this term is necessary. According to the original definition, references can be both associations between elements and method calls on other objects (Abreu and Carapuça, 1994). Since the aspect of methods is missing in the architecture models of the EAST-ADL it cannot be considered and is omitted for the calculation. Analogous to the CBO metric (see 4.2.1) , the power lines on the HDA should be excluded from the set of associations.

5 RELATED WORK

There are many publications regarding quality metrics for software and system engineering. An overview of metrics can be found for example in (Mohagheghi and Dehlen, 2009).

MATLAB Simulink models are commonly used in the automotive domain to solve control system engineering tasks. EAST-ADL provides the feature to link such models to have a behavioral description of functions via the behavior extension. Scheible (Scheible, 2012) presents an approach for automatic quality rating of MATLAB Simulink models. The metrics in this approach are created for Simulink models and cannot simply be applied to the FAA, FDA or HDA of EAST-ADL models.

SysML is closely related to EAST-ADL. Friedenthal et al. describe in their book “A Practical Guide to SysML” (Friedenthal et al., 2014), that model-based metrics are a good tool to assess on the quality of a design. They do not define concrete metrics, but give hints, which parameters of a SysML model are of interest for different kinds of metrics.

Architecture evaluations represent a comprehensive approach to the evaluation and analysis of specific or general quality criteria. Examples are Architecture-Level Modifiability Analysis (ALMA) (Bengtsson et al., 2004), Scenario-based Architecture Analysis Method (SAAM) (Kazman et al., 1994) and Architecture Trade-off Analysis Method (ATAM) (Kazman et al., 1998). These evaluations go beyond a pure metric assessment. However, metrics can play a supporting role in the various evaluation steps of the processes.

6 CONCLUSIONS AND FUTURE WORK

In this position paper, we presented metrics to support system designer in development process of EAST-ADL system architectures. Therefore, we gave a short introduction of the quality characteristics for system and software development defined by ISO 25010. In order to be able to conclude on the quality of EAST-ADL architectures we examined selected metrics from code analysis and object-oriented design

and proposed the application of these on EAST-ADL architecture models where possible.

The results of this paper do not provide a classification of which values can be regarded as "good" or "bad" for the metrics. For this purpose, a detailed evaluation of the metrics on models of already known quality is necessary. Then, using these metrics as key values to support other analysis like the partition analysis of EAST-ADL models (Etzel and Bauer, 2019) would be an application.

Section 4 exploits only a very small selection of available metrics for the quality determination of software, system and hardware design. Many more measures can be examined for their transferability to EAST-ADL models (see Section 5). Transferring the presented metrics to the more widespread SysML was not considered in this paper but may be of particular interest, since EAST-ADL is a specialization of this modeling language. As mentioned, a further analysis of the cyclomatic complexity in the context of EAST-ADL models may be valuable (see Section 4.1.2).

ACKNOWLEDGEMENTS

This work was partially funded within the project ARAMiS II by the German Federal Ministry for Education and Research with the funding ID 01IS16025. The responsibility for the content remains with the authors.

REFERENCES

Abran, A. (2010). *Software Metrics and Software Metrology*. John Wiley & Sons, Inc., Hoboken, New Jersey.

Abreu, F. B. and Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In *Proc. Int'l Conf. Software Quality (QSIC)*.

Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H. (2004). Architecture-level modifiability analysis (alma). *Journal of Systems and Software*, 69(1):129 – 147.

Blom, H., Chen, D., Kaijser, H., Lönn, H., Papadopoulos, Y., Reiser, M., Kolagari, R., and Tucci-Piergiovanni, S. (2016). EAST-ADL: An architecture description language for automotive software-intensive systems in the light of recent use and research. *International Journal of System Dynamics Applications*, 5:1–20.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20:476–493.

de Paoli, F. and Morasca, S. (1990). Extending software complexity metrics to concurrent programs. In *Proceedings., Fourteenth Annual International Computer Software and Applications Conference*, pages 414–419.

EAST-ADL Association (2013). EAST-ADL Domain Model Specification. Version V2.1.12.

Etzel, C. and Bauer, B. (2019). Extending EAST-ADL for modeling and analysis of partitions on functional architectures. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, pages 169–178. INSTICC, SciTePress.

Friedenthal, S., Moore, A., and Steiner, R. (2014). *A Practical Guide to SysML, Third Edition: The Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

Hoffmann, D. W. (2013). *Software-Qualität*. Springer Vieweg, Springer-Verlag Berlin Heidelberg, 2 edition.

ISO (2011). ISO/IEC 25010:2011. Standard, International Organization for Standardization.

ISO (2018). ISO 26262-1:2018. Standard, International Organization for Standardization.

Kan, S. H. (2003). *Metrics and models in software quality engineering*. Pearson Education, Inc., 2 edition.

Kazman, R., Bass, L., Webb, M., and Abowd, G. (1994). Saam: A method for analyzing the properties of software architectures. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 81–90, Los Alamitos, CA, USA. IEEE Computer Society Press.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J. (1998). The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference on*, pages 68–78.

Liggesmeyer, P. (2009). *Software-Qualität – Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag Heidelberg, 2 edition.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Mohagheghi, P. and Dehlen, V. (2009). Existing model metrics and relations to model quality. In *Proceedings of the Seventh ICSE Conference on Software Quality*, WOSQ'09, pages 39–45, Washington, DC, USA. IEEE Computer Society.

Scheible, J. (2012). Automated quality rating on the example of matlab simulink models in the automotive domain. Dissertation, Universität Tübingen.

Sneed, H. M., Seidl, R., and Baumgartner, M. (2010). *Software in Zahlen - Die Vermessung von Applikationen*. Carl Hanser Verlag München.