Automated Model Quality Rating of Embedded Systems

Jan Scheible[1] and Hartmut Pohlheim[2]

[1] Daimler AG - Group Research and Advanced Engineering
jan.scheible@daimler.com

[2] Model Engineering Solutions GmbH
pohlheim@model-engineers.com

Abstract. As in conventional software development, model-based software development with Matlab Simulink needs a way of safeguarding quality. However, quality assurance of models is becoming increasingly complex as the size and scope of models continue to expand. This paper presents an approach that enables objective and automated rating of model quality with the help of a quality model. The focus of this approach lies on both the Simulink model itself and the entire model-based development process. Metrics are used to determine the existence of desired quality criteria and their degree of fulfillment. Statements regarding the maturity of a model can be made by evaluating the metrics and aggregating the results in a quality model. This approach allows large and complex models to be rated automatically and easily, with much less effort.

1 Introduction

The automotive industry has been moving towards model-based software development in recent years (cf. [KCFG05]). Models now occupy the position of the main artifact, a position which was previously occupied by source code. Source code is generated from the models using a code generator and as such, source code is losing its importance as the artifact to be processed. As a result, model quality directly influences software quality [FHR08].

One tool favored in the model-based software development of embedded systems is e. g. Matlab Simulink with dSPACE TargetLink[3]. The used models consist of blocks (functions), connected by lines (data flows), and are structured hierarchically into subsystems. In contrast to MDA (model-driven architecture) from OMG[4], Simulink does not use UML diagrams, but rather a proprietary block diagram notation. This unites the PIM (platform-independent model) and the PSM (platform-specific model) in one model that contains all information.

Simulink models are growing increasingly large and complex. Large models in the automotive domain can contain up to 15,000 blocks, 700 subsystems and 16 hierarchical levels. This makes quality assurance of models an ever more daunting undertaking. In order to rate model quality without an exorbitant effort, we proposed an automated approach with a quality model for Simulink models in [Sch10] and [SK10].

[3] http://www.dspace.com/de/gmb/home/products/sw/pcgs/targetli.cfm
[4] http://www.omg.org/mda

A quality model can be used to determine the existence of desired quality criteria that characterize a high-quality model. It provides a way of looking at models that permits a statement regarding their quality. Consequently the quality model defines our notion of model quality. If a model fulfills all quality criteria it is of high quality. Our quality model is structured like the quality model of Cavano and McCall in [CM78]. Factors that influence model quality are defined and specific criteria are used to determine whether these factors are fulfilled. To what extent these criteria are fulfilled is measured by means of metrics. A quality model therefore has the structure of a tree, whose leaves represent the metrics. Our quality model currently consists of 6 factors, 17 criteria, and 43 metrics. Thanks to its tree-like structure, a quality model can be extended to include any number of factors, criteria, and metrics.

Quality assurance must be an integral part of the entire development process from start to finish. The goal is to detect errors as early as possible in the development process, as corrections applied at this stage only involve a limited number of development phases [FLS01]. In addition, this forward-looking approach reduces the cost of maintenance and support. This paper discusses *embedding the quality model in the development process* by establishing the relation between the most important artifacts in the model-based development process and the quality model itself. We then proceed to describe *model quality rating* of the analyzed models and *trend analysis*. Finally, we give an overview of our prototypical *implementation and further evaluation* of our approach.

2 Embedding the Quality Model in the Development Process

Previously (see [Sch10] and [SK10]), the focus of the quality model lay on the Simulink models themselves. However, model quality cannot be sufficiently assessed by only looking at the model itself; other artifacts in the development process must also be considered. Even when all modeling guidelines are fulfilled, complexity per subsystem is low, and the model fulfills all other statically verifiable quality criteria, there is still no guarantee that the model possesses the desired functionality. The functionality can only be verified through a manual model review or functional test. This is why other artifacts must be considered in model quality rating.

Figure 1 shows a simplified V-model with the most important artifacts from the point of view of model-based development. The rotating arrow in the middle of the process indicates that we are dealing with an iterative procedure.

The artifacts in the process are all in relation to each other. Thus all requirements must be implemented in the model and test specifications must exist for all the requirements. These test specifications must in turn be implemented in the form of test implementations. In initial iterations, only a few requirements are implemented in the model and only a portion of test specifications exist. Towards the end of development, all requirements must have been implemented in the model and all test specifications with their test implementations will have been created. We are then able to make a statement regarding the status of implementation of functionality based on the degree of coverage of individual artifacts in relation to each other.

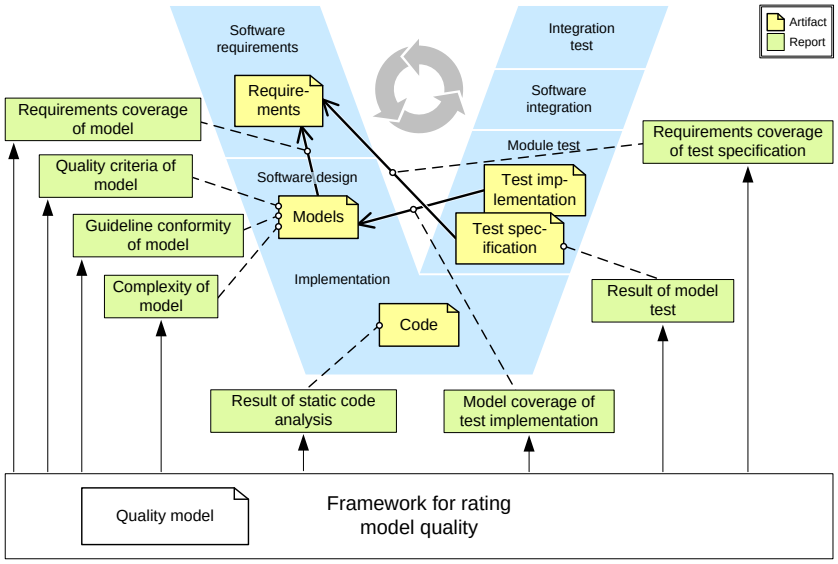

Fig. 1. Development process with artifacts and analysis for quality rating

There are also additional analyses (such as model guideline checking or static code analysis of generated code) that deliver information on the artifacts shown in Figure 1.

The framework for rating model quality from Figure 1 represents the necessary infrastructure for automated quality rating. It is for example responsible for extracting the results of the analyses, which are used as measured values in the metrics.

Factors	Criteria	Metrics
Testability	Test coverage	Requirements coverage of test specification
		Model coverage of test implementation
Comprehensibility	Scale	Complexity of model
Reliability	Runtime errors	Result of static code analysis
Maintainability	Standard conformity	Guideline conformity of model
Correctness	Functionality	Result of model test
		Requirements coverage of model

Table 1. Excerpt from the quality model for considering process artifacts

Table 1 shows the relevant excerpt from the quality model for considering the metrics from Figure 1. The metrics have been classified in the quality model by allocating appropriate criteria. Moreover, the quality model contains all factors, criteria, and metrics from [SK10], which are subsumed in Figure 1 under the report *quality criteria of model*.

3 Model Quality Rating

This section describes the required steps for rating a model. The first step is to identify the measured values of the metrics. This step is not described in detail here as it is specific to each metric. Next, the permissible values for each metric are defined and their measured values are evaluated. In the final step, all evaluations are aggregated upwards

in the quality model tree. We will now discuss in greater detail how to set limits for evaluating measured values and aggregating evaluations.

3.1 Evaluation of measured values

First the permissible values for each metric are calculated. Limits are defined to check these permissible values. There are three types of limits: interval, maximum, and minimum. In the case of interval limits, a value's compliance with the given minimum and maximum values of the interval is checked. Minimum and maximum limits merely specify a permissible minimum or maximum value. Previously (in [Sch10]), only compliance with limit values was checked, i. e. evaluation for each metric was either *true* or *false*. To enable better differentiation of evaluations, an interval result [0%..100%] is now used instead. A completely fulfilled metric is evaluated with 100%, whereas a completely unfulfilled metric is evaluated with 0%. To allow a soft transition between fulfilled and unfulfilled, a tolerance area is defined at each minimum and maximum. The result in the tolerance areas is currently calculated through linear interpolation, however any number of different interpolation procedures is similarly conceivable.

Two approaches are used for evaluating measured values, i. e. for finding and setting limits. The first approach is based on a *reference model*, which is instantiated on the basis of empirically derived data. The second approach employs *rules*. These rules make statements regarding the desired properties of the models.

Reference model A reference model describes how an average Simulink model should look. By comparing the model under investigation with a reference model, outliers can be detected. An individual reference model is instantiated for each model rating. The starting point for the instantiation of a new reference model is the number of blocks in the model under investigation. This number is used to derive permissible values: for example the number of subsystems, the average degree of subsystem children, or the maximum subsystem depth. These values are then used as limits for the metrics. The permissible values are calculated by assuming a linear relation between the number of blocks in the model under investigation and the average values in a database. In this manner minimum and maximum values are calculated. This calculation effectively scales the created reference model to the size of the model under investigation. In the same way as in the aforementioned examples, other permissible values are derived from the block number. Table 2 shows excerpts from three instantiated reference models for different block numbers.

	200 blocks		1,000 blocks		5,000 blocks	
	Min.	Max.	Min.	Max.	Min.	Max.
#Subsystems	14	46	70	230	348	1,149
#Lines	224	256	1,120	1,280	5,600	6,400
#Crossed lines	-	96	-	480	-	2,400

Table 2. Excerpts from instantiated reference models for different block numbers

The database we currently use for constructing reference models consists of 12 structurally similar models originating from one project. It contains average values and their deviation for measurements of all metrics for each of the twelve models.

The average values and their deviation are currently calculated using corrected sample variance and arithmetic mean. The models have a size of between 1,000 and 10,000 blocks.

Rules In the case of numbers where a useful average value cannot be determined, limits must be set in a different way. This can be done using rules specifying that certain relations between the measured values must be adhered to. So, for example, in a Simulink model there must be either an equal number or more read blocks (data store read) than store blocks (data store memory). If there were to be more store blocks than read blocks, then the value of certain memory blocks would never be read. This relation can be described in the following rule: $\#DataStoreRead >= \#DataStoreMemory$. Rules can also be parametrized. This makes it possible to adapt rules to specific project requirements. For example, by using such parameters, one project can allow an average of one Goto block per subsystem and another project can prohibit Goto blocks entirely.

Visualization Figure 2 shows some of a model's measured metric values and their permissible values displayed in a Kiviat diagram (cf. [Dra96]). The model originates from the database of the reference model. A Kiviat diagram provides a quick overview of multiple measured values as it contains both the values and their interpretation. Each axis represents the measured value of one metric.

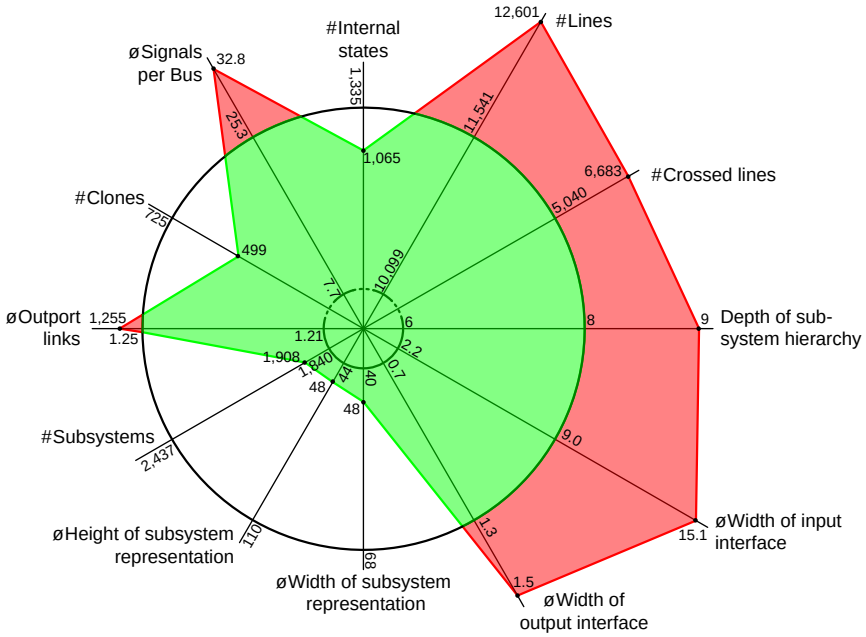

Fig. 2. Measured values and permissible values

The scaling of each axis is chosen in such a way that its minimum permissible value lies on the inner circle and its maximum permissible value is positioned on the outer

circle. A polygon is formed by connecting the measured values. Its red (or dark) areas highlight outliers more clearly.

3.2 Aggregation of evaluations

Section 3.1 described the evaluation of metrics. After evaluation, all leaves in the quality model have an assigned evaluation of between 0% and 100%. The next step is to aggregate the evaluations in the quality model. In other words a procedure is required that describes how the metric evaluations can first be aggregated into criteria and, in the next step, into factors. Aggregated results provide a quick overview of a model's quality.

During aggregation, a criterion with four metrics, for example, three of which are fulfilled with 100% and one with 0%, should not be evaluated as being 75% fulfilled. If one or more metrics are not fulfilled at all or only fulfilled to a minimal extent, this should lead to a devaluation instead. Thus the evaluation of the criterion must be worse than average. This can be achieved by multiplying the arithmetic mean with the evaluation percentages that are larger than a designated threshold value. This results in the following equation for calculating an aggregated evaluation:

$$aggregatedEvaluation = \underbrace{\frac{\sum_i evaluation_i}{\#evaluations}}_{Arithmetic\ mean} \cdot \underbrace{\left(\frac{\#evaluationsOverThreshold}{\#evaluations}\right)^{exponent}}_{Damping\ factor} \tag{1}$$

The damping factor can be further reduced using an exponent, so that evaluations under the chosen threshold value lead to a greatly reduced aggregated evaluation. If all evaluations lie above the threshold value, the aggregated evaluation corresponds with the arithmetic mean. The threshold value can be selected globally or for each aggregation step individually. It can be chosen, for example, in accordance with the ASIL (automotive safety integrity level) defined in ISO 26262. The ASIL determines how many of the specified ISO measures must be fulfilled. If the evaluations that are being aggregated are also weighted, then a weighted arithmetic mean must be used in place of a simple arithmetic mean.

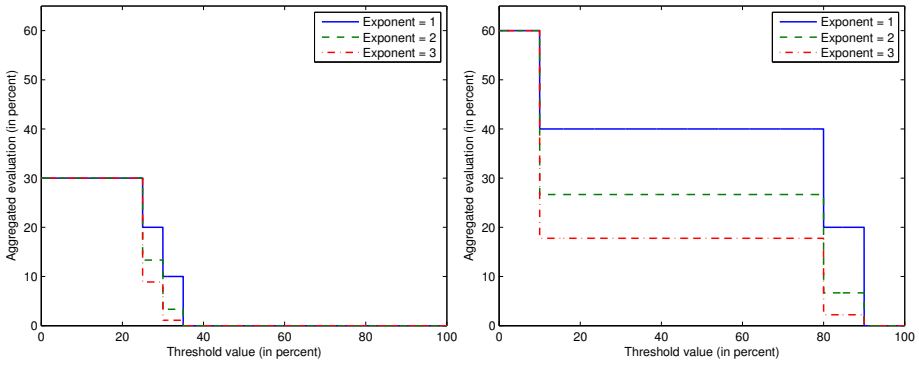

Fig. 3. Aggregation of $e_1 = [25\%, 30\%, 35\%]$ and $e_2 = [10\%, 80\%, 90\%]$

Figure 3 shows two aggregated evaluations, each with three different exponents. The left aggregation uses the evaluations e_1 and the right aggregation uses e_2. The x axis shows the threshold value and the y axis shows the aggregated evaluation. When none of the evaluations lie above the threshold value, we can clearly see that the damping factor becomes 0 and the aggregated evaluation is thus 0.

Aggregation of criteria into factors is carried out in the same way as aggregation of metrics into criteria. If desired, it is also possible to calculate an ultimate overall value for model quality rating, which is the result of a final aggregation of factors.

Visualization Figure 4 shows a complete model quality rating for a model from the database of the reference model. The aggregation is parameterized with a threshold value of 20% and an exponent of 1. A nested pie chart representation of the quality model is used for optimal visualization. The outer ring represents metric evaluations. The next ring in shows the criteria belonging to the metrics (from here on, evaluations are also given as percentages). The criteria are followed by the factors. The center circle of the pie shows the overall evaluation of evaluated factors. The color of each segment codes the evaluation of the respective segment. The evaluation is shown on a linear transition between the colors green (lighter) and red (darker). In this way, for example, completely fulfilled segments are green, partially fulfilled segments are different shades of orange, and non-fulfilled segments are red.

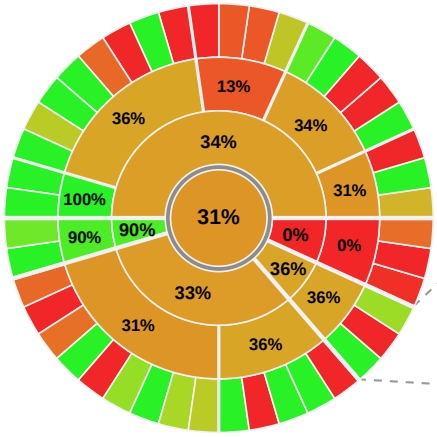

Fig. 4. Complete model quality rating

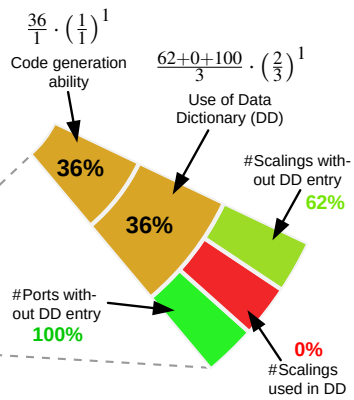

Fig. 5. Rating of a factor

Figure 5 shows an example for rating a factor. The aggregated evaluation of 36% results from the fact that the damping factor from Equation 1 has the value $2/3$, since one of the evaluations does not exceed the specified threshold value of 20%.

4 Retracing Quality Rating over Time

A model quality rating is a snapshot of a model's evaluations at a specific time. Trend analysis is used to retrace the evolution and evaluation of the measured values over

time. Figure 6 and Figure 7 show the trend of two metrics over three iterations of the development process. In Figure 6, the relative trend is shown. If only the metric evaluations from section 3.1 are shown, however, it is impossible to determine to what extent a metric's measured values depart from their permissible values. This is why Figure 7 shows the absolute trend of metrics. This representation clearly shows how the metrics' measured values lie in relation to their limits and tolerance areas. It is thus possible to determine why Metric 2 has the value 0% in the second iteration. Moreover, this representation enables us to determine whether a measured value is moving towards its allowed limits even though it always remains 0% in the relative representation. Trend analysis therefore helps us to understand how quality rating came about over different points in time.

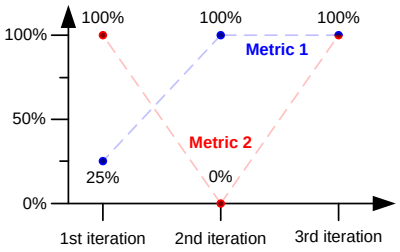

Fig. 6. Relative trend representation

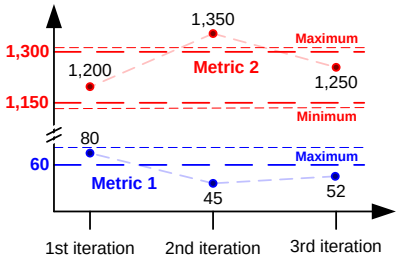

Fig. 7. Absolute trend representation

Furthermore, relative trend analysis is not only restricted to metrics, but can also be applied to every element of the quality model. This makes it possible to get an overview of the progress of particular criteria or factors over time.

5 Implementation and Further Evaluation

The framework for rating model quality (cf. Figure 1) is currently implemented as a Java prototype, however a product called MQA Center (Model Quality Assessment Center) is planned for the future. The prototype is responsible for connecting external tools, executing metrics, and evaluating the quality model, as well as visualizing results. An example of an external tool with a parsed report is the guideline checker MXAM[5]. Alternatively, the external tools themselves can supply metrics that are then collected by the prototype. As soon as all measured values and information are available, model quality rating can be performed and visualized.

The approach presented in this paper will be evaluated with the help of this prototype. The evaluation consists of three main components. The first is the comparison of models with one another. Combining this comparison with the ratings of developers as to the perceived quality of models means we can determine whether the basic direction of the model quality rating is appropriate. Then we can analyze the findings of model reviews. The nature of model review findings is that they very often only detect potential problems (issues and not necessarily errors). So we can test whether models with a lot of findings

[5] http://www.model-examiner.com

have a worse rating. Finally, we will compare the model rating with the appearance of actual bugs in the models. This can be done by getting bug information from our issue tracking. This approach will let us prove the validity of the model quality rating.

6 Related Work

The model quality rating presented in this paper evaluates the measured values of a model in comparison to empirical data and whether a model complies with certain rules. Many other studies have selected a large number of metrics where it is assumed that their measured values permit the desired conclusions. They then investigate whether a correlation exists between the measured values and the actual issues at hand (e. g. [RPRB97], [MPKS00], [BeAM96], and [BBM96]).

The ConQAT framework [DJHWPP02] from the TU Munich primarily focuses on visualizing different quality characteristics of source code. However, it neither offers support for finding permissible values of metrics, nor does it natively support an aggregation comparable to that detailed in Section 3.2. So far only simple aggregations such as e. g. minimum, maximum, average, and median have been integrated into their framework.

Kemmann et al. [KKT10] use customer-specific quality attributes to handle the varying requirements of different domains. This is accomplished by providing the user with a toolkit for creating custom quality indicators, which determine the existence of desired quality attributes. They support multiple modeling languages by using adapters to interface with their generic data flow model. One adapter is, for example, a Matlab Simulink adapter. In their approach the finding of permissible values and aggregation are performed on the level of the quality indicators.

Menkhaus and Andrich [MA05] identify relevant locations for a failure mode effect analysis (FMEA) with the help of metrics. The metrics are applied to Matlab Simulink models. Ultimately, however, only one metric is used as a reference for the subsystems under investigation.

7 Conclusion and Outlook

Our approach not only examines the model itself, but also includes the other artifacts in the development process. A model can thus only receive a good rating when not only all specified statically verifiable quality criteria have been fulfilled, but also all requirements have been referenced and tested. This rating is carried out automatically with the help of the quality model. In a first step the metrics are evaluated, and these evaluations are subsequently aggregated in the quality model. This procedure is not dependent on the size of the model and thereby enables automated model quality rating for large and complex models.

The next step will be the evaluation described in Section 5. Furthermore, we need to check whether it is sufficient to assume linear relations in the reference model. We also have to evaluate alternatives for using the number of blocks as a scaling factor for the instantiation of the reference model. One promising idea is to take model complexity as a basis for calculating permissible block numbers. Complexity could, for example, be calculated using the Halstead volume [SPR10]. We also need to investigate to what

extent models must be of the same type, i. e. whether only models from the same project can be compared using this approach.

Moreover, the quality model must be further refined. On one hand more Simulink-specific metrics must be integrated. These would take the semantics of Simulink models into consideration and thus enable more detailed statements. On the other hand, we must validate which metrics have the highest relevance for model quality evaluation. Ultimately the goal is to know which relevant metrics must be combined to obtain a meaningful and compact quality model.

References

[BBM96] Victor R. Basili, Lionel C. Briand, and Walclio L. Melo. *A Validation of Object-Oriented Design Metrics as Quality Indicators.* IEEE Trans. Softw. Eng., 22(10):751761, 1996.

[BeAM96] F. Brito e. Abreu and W. Melo. *Evaluating the Impact of Object-Oriented Design on Software Quality.* In METRICS 96: Proceedings of the 3rd International Symposium on Software Metrics, page 90, 1996. IEEE Computer Society.

[CM78] J. P. Cavano and J. A. McCall. *A framework for the measurement of software quality.* In Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, pages 133139, 1978.

[DJHWPP02] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. Mas y Parareda, and M. Pizka. *Tool Support for Continuous Quality Control.* IEEE Software. Vol. 25, Nr. 5, IEEE Computer Society, September 2008.

[Dra96] T. Drake. *Measuring Software Quality: A Case Study.* Computer, IEEE Computer Society Press, 29(11):7887, 1996.

[FHR08] F. Fieber, M. Huhn, and B. Rumpe. *Modellqualität als Indikator für Softwarequalität: eine Taxonomie.* Informatik-Spektrum, 31(5):408424, October 2008.

[FLS01] K. Frühauf, J. Ludewig, and H. Sandmayr. *Software-Projektmanagement und Qualitätssicherung,* Chapter Software-Nutzen und -Kosten. Teubner Verlag, 2001.

[KCFG05] T. Klein, M. Conrad, I. Fey, and M. Grochtmann. *Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler.* Informatik - Forschung und Entwicklung, 20(1-2):310, 2005.

[KKT10] S. Kemmann, T. Kuhn, and M. Trapp. *Extensible and Automated Model-Evaluations with INProVE.* Workshop on System Analysis and Modeling, 2010.

[MA05] G. Menkhaus and B. Andrich. *Metric Suite for Directing the Failure Mode Analysis of Embedded Software Systems.* In Proc. of ICEIS, 2005.

[MPKS00] S. Muthanna, K. Ponnambalam, K. Kontogiannis, and B. Stacey. *A Maintainability Model for Industrial Software Systems Using Design Level Metrics.* In Proc. of the 7. WCRE, page 248, Washington, DC, USA, 2000. IEEE Computer Society.

[RPRB97] S. Rivard, G. Poirier, L. Raymond, and F. Bergeron. *Development of a measure to assess the quality of user-developed applications.* SIGMIS Database, 1997.

[Sch10] J. Scheible. *Ein Framework zur automatisierten Ermittlung der Modellqualität bei eingebetteten Systemen.* In Proc. of the Dagstuhl-Workshop: Model-Based Development of Embedded Systems, Schloss Dagstuhl, Germany, 2010.

[SK10] J. Scheible and I. Kreuz. *Ein Qualitätsmodell zur automatisierten Ermittlung der Modellqualität bei eingebetteten Systemen.* In Proc. of the 8. GI Workshop Automotive Software Engineering, Leipzig, Germany, 2010.

[SPR10] I. Stürmer, H. Pohlheim, and T Rogier. *Berechnung und Visualisierung der Modellkomplexität bei der modellbasierten Entwicklung sicherheits-relevanter Software.* In Automotive Safety & Security 2010, pages 6982. Shaker Verlag, 2010.