Model Quality Assessment in Practice: How to Measure and Assess the Quality of Software Models During the Embedded Software Development Process

Ingo Stürmer, Hartmut Pohlheim
Model Engineering Solutions GmbH, Friedrichstr. 55, 10117 Berlin, Germany

Abstract: In this paper we present an approach for assessing the quality of model-based software projects. We demonstrate this approach in embedded software projects in the automotive domain. Our methodology can however be extended to other fields of software development. In general, the quality assessment of software projects is often based on quality criteria such as maintainability, testability, readability of the software or the software model. Our approach expands this basic assessment in that we assess the quality of a project on the basis of the success or degree of realization of quality operations (testing, reviews, etc.) respectively. Furthermore, we do not focus on single artifacts of the development process (e.g. model or code); rather we take all relevant development artifacts into account and use their interrelations to obtain metrics and key figures. We demonstrate our approach using our tool, Model Quality Assessment Center (MQAC), which was expressly developed for this purpose.

1 Introduction

In the automotive domain as well as in the avionics sector, the approach to developing embedded software has changed in recent years. Executable graphical models are now used at all stages of development: from the initial design phase to implementation. Model-based design is now recognized in process standards such as the upcoming ISO 26262 standard for the automotive domain and the DO-178C standard for the avionics sector. Software models are used in different stages of embedded software development. They are used for verifying functional requirements as an executable specification, and also as so-called implementation models used for controller code generation. The models are designed with common graphical modeling languages, such as Simulink and Stateflow from The MathWorks [1] in combination with automatic code generation with TargetLink by dSPACE [2] or the Real-Time Workshop/Embedded Coder by The MathWorks. 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. An automated approach has become necessary in order to assess and rate model quality without an exorbitant effort. 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. The question as to whether the quality of the software model used for code generation is sufficient is important during the whole software development cycle. Also the question as to which metrics and measures need to be applied to assess the quality of the models is of paramount importance.

In this paper we discuss methods and metrics for assessing the quality of software models used in serial production code generation in the automotive domain. We introduce our tool MQAC (Model Quality Assessment Center), which aggregates different metrics and measures from different tools, e.g. from requirements engineering, testing tools, code and model coverage tool, etc. An automatic assessment of the quality metrics aggregated is also given. Finally, we will show how the developer, the quality manager, and the project manager can be supported by the tool with automatic model quality analysis and assessment.

1.1. Model and Code Verification

In model-based development, it is quite common to place the focus of quality assurance on the Simulink models themselves, as well as on the generated code once the code is available. A survey of quality assurance methods in model-based development is given in [4] and will therefore not be discussed here in greater detail.

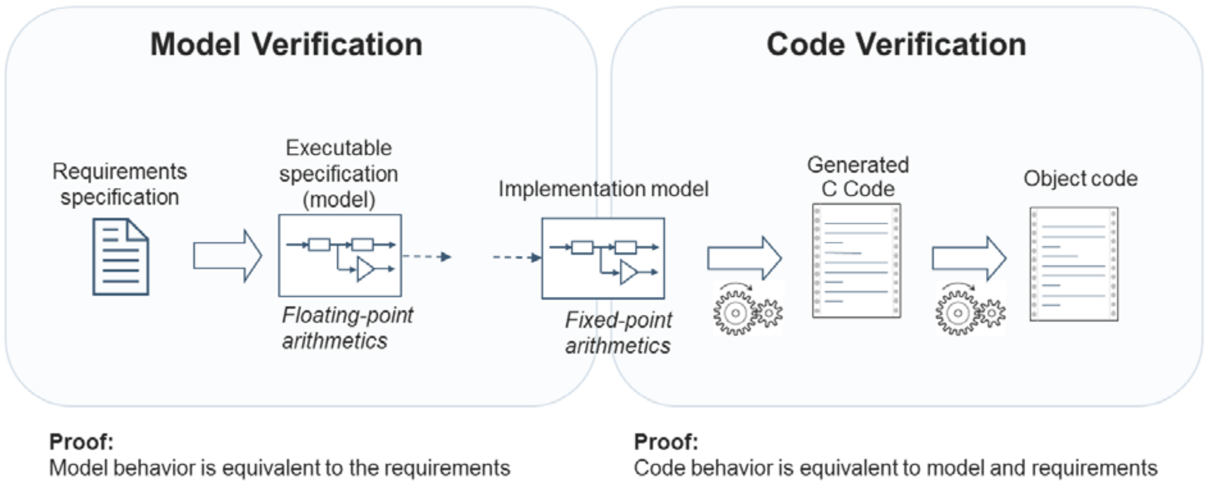

Model Verification
Code Verification
Requirements specification
Executable specification (model)
Floating-point arithmetics
Implementation model
Fixed-point arithmetics
Generated C Code
Object code
Proof:
Model behavior is equivalent to the requirements
Proof:
Code behavior is equivalent to model and requirements

Figure 1: Model and code verification in model-based development

The basic principal behind model verification and code verification in model-based development is illustrated in Fig. 1. In an early development stage, models are created on the basis of the textual requirements specification with the goal of validating the functional behavior of the model, i.e. proving that the execution behavior is equivalent to the specified behavior of the function to be developed. This is characteristic for model-based development, since the function behavior can already be verified on model level before any implemented code is available. Here, constructive as well as analytical quality assurance methods are applied such as (1) use of modeling guidelines; (2) manual review of the requirements specification and the model; (3) simulation and testing of the model. In a later development stage, the 'requirements model' is manually converted into a so-called 'implementation model', i.e. the model is augmented with implementation details for automatic code generation such as (fixed-point) data types, scaling, function distribution, etc. The implementation model is then used for production code generation purposes. The generated C code as well as the object code on the embedded system is safeguarded on different test levels such as unit testing, integration testing, system testing, etc. The quality assurance methods on code level are mainly adopted to ensure that (1) code behavior is equivalent to model behavior; and (2) code behavior is equivalent to implementation behavior, e.g. maximal execution time, as specified in the requirements specification.

1.2. Artifacts of Model-based Software Development

In the process of model and subsequently code development and the safeguarding thereof, different artifacts are developed which are relevant for the quality assessment of the model, the code, and the whole development process. These important artifacts include: (1) textual requirements specifications; (2) the model itself (requirements and implementation model); (3) test specification and test implementation; (4) test reports; (5) review reports for the requirements, the model, and the code; (6) issues, i.e. bug tracking reports; etc. Additionally, different tools generate reports containing metrics and key figures for quality assessment. Examples of these kind of reports include reports on (1) the compliance of modeling guidelines; (2) model coverage and code coverage; (3) test coverage of the requirements; (4) complexity measurement of the model and the code; etc.

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 also be considered in model quality assessment. For the quality assessment of a project, it is common practice to focus on the implementation (i.e. the software) of the function to be developed, with the goal of measuring the success of a project. However, often appropriate methods are lacking for assessing the actual quality of a project, e.g. at a specific time t_1. One reason could be that the implementation is not yet available at that specific point in time. This is particularly true for early development stages. Another reason could be that the degree of function realization is not yet high enough to obtain significant results at time t_1 or t_n.

2 Quality Assessment of Software Projects

No general procedure has thus far been established for the quality assessment of model-based software projects. The factors that influence quality assessment can be delineated by means of the typical questions that arise in the course of quality assessment. These include:

- How many test cases are specified?
- How many of the specified test cases are implemented?
- How many test cases are passed or failed?
- How many requirements are covered by test cases?
- How many requirements are realized within the model?
- How many safety-requirements are realized within the model and tested?
- How many modeling guidelines are passed or failed?
- How many model parts are easy to maintain and not too complex?
- How many model parts are covered by test cases?
- How many issues result from the model review?

Other questions may arise from different perspectives depending on the role of the people involved in the development process:

- **Developers:** What is still open? Where is quality not yet sufficiently given? What has already been accomplished?

- **Integrators:** What is the actual quality of modules to be integrated? What is the quality of the integrated system?

- **Project managers:** How is the planning of tasks and milestones proceeding? I need a compact overview of all project areas (development status, quality assurance status, etc.). How can I get a fast and compact overview of problem areas?

2.1. Basic Principles of Quality Assessment in Software Projects

The basic principles of how the quality assessment of a software project can be carried out are shown in Fig. 2. Artifacts, which are developed during the development process, are categorized by means of specific criteria (model, code, document, etc.) and used as a basis for quality assessment.

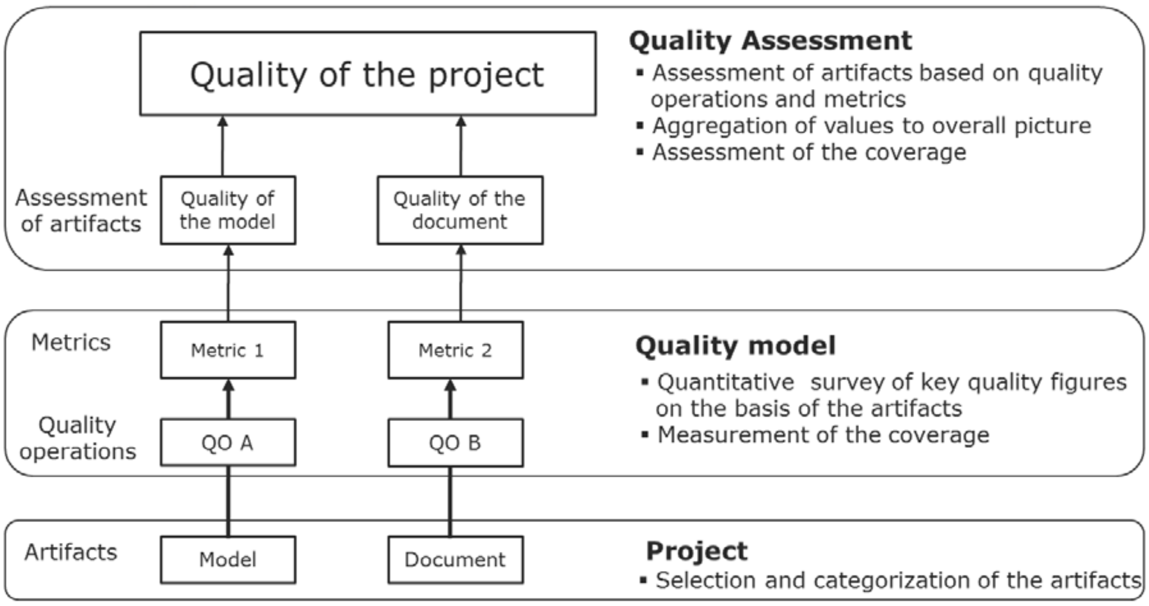

Quality of the project
Quality Assessment
▪ Assessment of artifacts based on quality operations and metrics
▪ Aggregation of values to overall picture
▪ Assessment of the coverage
Assessment of artifacts
Quality of the model
Quality of the document
Metrics
Metric 1
Metric 2
Quality model
▪ Quantitative survey of key quality figures on the basis of the artifacts
▪ Measurement of the coverage
Quality operations
QO A
QO B
Artifacts
Model
Document
Project
▪ Selection and categorization of the artifacts

Figure 2: Basic principles of quality assessment in software projects

The next step is to define the *quality model* of the project. This quality model is used as a reference on the basis of which the artifacts are analyzed and assessed. One approach for how to develop and adopt a quality model for Simulink models is described in [6]. In general, quality models are often based on quality criteria such as maintainability, testability, readability of the software or the software model. Our approach expands this basic assessment in that we assess the quality of a project on the basis of the success or degree of realization of quality operations (testing, reviews, etc.) respectively.

Definition: *Quality operations* are any means which are adopted in order to obtain metrics and (quality) key figures such as reviews of the requirements specification or test specification, execution of a model review, execution of the model test, etc.

Furthermore, we do not focus on single artifacts of the development process (e.g. model or code); rather we take all relevant development artifacts into account and use their interrelations to obtain metrics and key figures. Success criteria are needed to calculate the required metrics for any of the quality operations. In the simplest case, success criteria could be criteria such as pass/fail results. Quality operations could also calculate the traceability of safety requirements to the software on unit level. A respective metric could then provide key figures on how many requirements are realized and covered within the model. Quality operations such as complexity measurement are finally used to calculate metrics and key figures, which provide statements about modeling depth, model structure, library concept, etc.

2.2. Quality Assessment of Model-based Software Projects

Many different factors influence the quality of model-based software projects. We will illustrate the most important of these below (cf. Fig. 3).

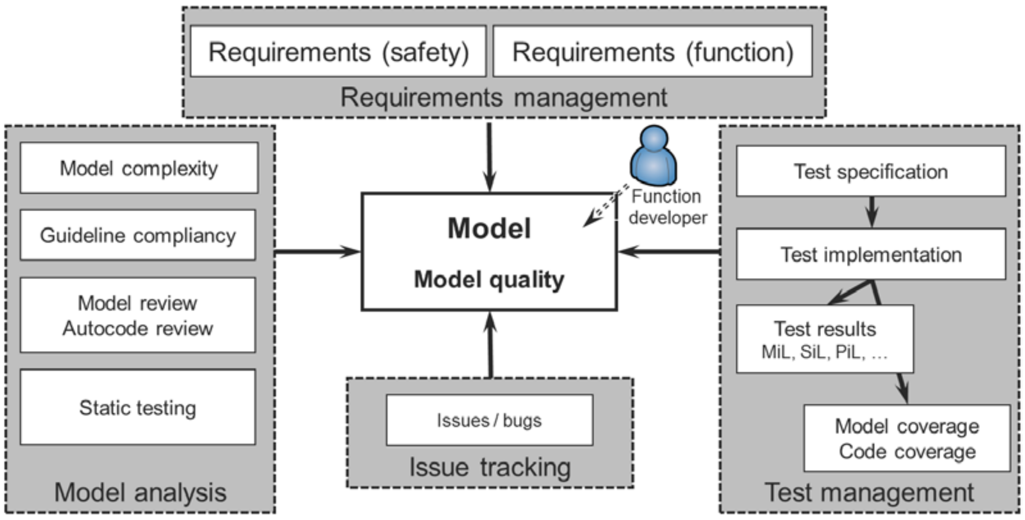


Figure 3: Key factors influencing model quality

The quality of a model used in software design and code generation is determined by four main factors:

(1) The quality of the requirements specification

(2) The scope and intensity of the model analysis, which includes complexity measurement, guideline checking, reviews, and further static analysis methods

(3) The scope and intensity (i.e. depth) of testing

(4) The tracking and traceability of issues, which might also include software errors (bugs)

These four factors cannot be assessed individually, because their interrelation also has to be an integral part of the quality assessment. For example, it is important to know the degree to which the requirements are realized within the model with the goal of realizing 100% of the functional requirements. If 80% of the functional requirements are realized, this statement might make a project manager happy. However, it is worth bearing in mind that ‘easy’ requirements are often realized at the start before more complex problems are addressed due to efficiency reasons and lack of time. In this sense, the quantification of requirements is an essential factor in order to gain significant results with regard to the degree of realization on model level. A simple method that takes this case into account is to weight the individual requirements using numerical values or to categorize them into classes such as *easy*, *medium*, or *difficult* to implement. This can also be applied to the other factors listed above.

Model analysis (cf. Fig. 3, left) focuses on the following aspects:

(1) **Complexity measurement:** complexity calculation and assessment of modeling depth (subsystem hierarchy) by means of metrics. Relevant metrics are here pass/fail criteria, number of subsystems with pass/fail. An extension to this is to weight subsystems with regard to their complexity. An approach for this is described in [5].

(2) **Adherence to modeling guidelines:** here a guideline checking tool checks to which extent the model complies with company or project-specific modeling guidelines [7]. The number of reported pass/fail/not executed violations and messages is important here. We can expand this approach by measuring the guideline violations for each subsystem in relation to the complexity of the subsystems respectively.

Test management (cf. Fig. 3, right) has to answer a range of questions, such as: (1) How many tests are specified? (2) How many specified tests are implemented? (3) How many tests are successfully executed (goal: 100% passed)? (4) How many tests have revealed an error (fail)? (5) How many test cases cover (i.e. test) how many requirements?

2.3. Tool Support for Quality Metrics and Key Figures

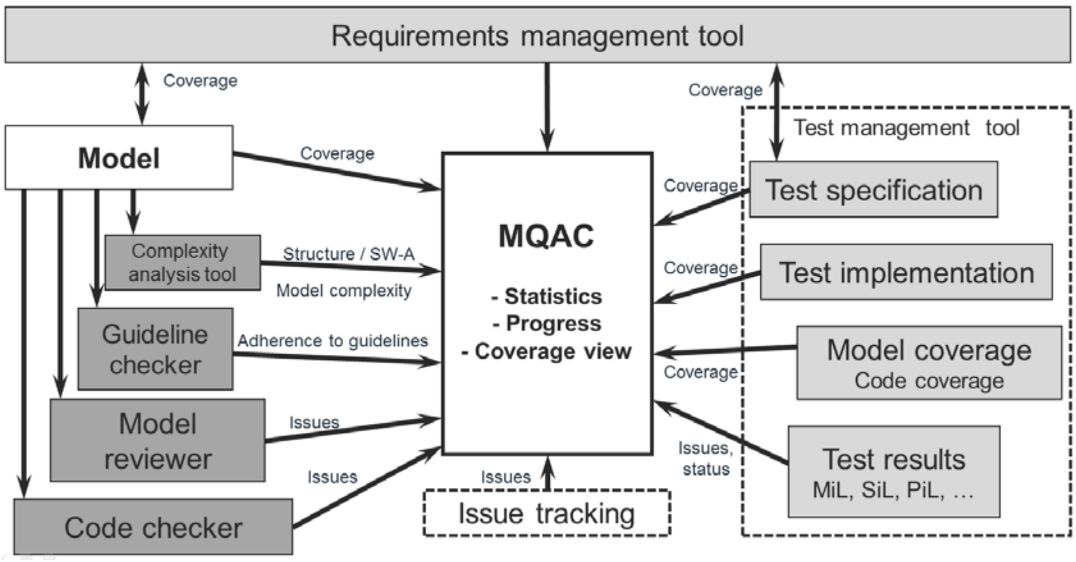


Figure 4: Tools delivering data for quality assessment

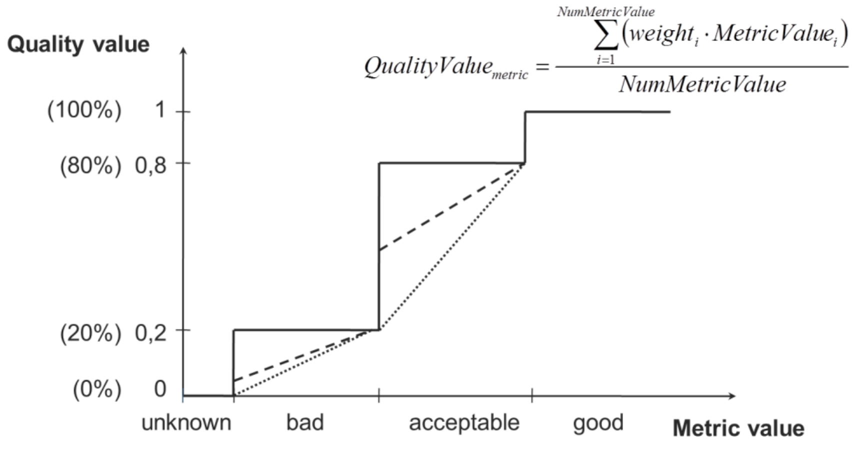
In Fig. **4** we see which tools and processes provide data for quality assessment of the software model. First of all it, we need to track how many functional requirements are fulfilled within the model, i.e. what proportion of requirements are implemented in the model?

Several metrics are necessary to answer these questions:

- ☐ Implemented requirements / requirements overall
- ☐ Goal: 100% of requirements to be implemented
- ☐ How far has model implementation progressed?
- ☐ An extension would be the weighting of individual requirements according to priority and the degree of requirements implementation

2.4. Quality Values and Weighting

A comprehensive quality assessment requires quality values, which are provided by different tools that are part of the model-based development tool chain. Based on this, the quality values need to be aggregated and evaluated in order to provide an overall assessment of the quality. This procedure takes place as follows:



$$QualityValue_{metric} = \frac{\sum_{i=1}^{NumMetricValue}\left(weight_i \cdot MetricValue_i\right)}{NumMetricValue}$$

Figure 5: Aggregation of metric values of a quality operation

In Fig. 5 we show our approach for calculating and aggregating quality values for individual artifacts as realized by our MQAC tool. The quality value of an artifact or a quality operation is calculated using the mean value of the metrics provided by a quality operation. The metrics can provide numerical values for different metric values such as *unknown*, *bad*, *acceptable*, or *good*. The metrics are then mapped to numerical values respectively: *unknown=0*, *bad=0.2*, *acceptable=0.8*, and *good=1*. This results in the discrete (solid) line shown in Fig. 5. Continuous deviation can be adapted if required (dotted and dashed line). Quality values for the quality operations are then calculated on the basis of the arithmetic mean value of the individual quality operations (cf. Fig. 5, top right formula). The metric value *MetricValue*$_i$ can also be weighted if necessary with *weight*$_i$ ≥ 0 and ≤ 1. This results in what is known as the *weighted mean value*.

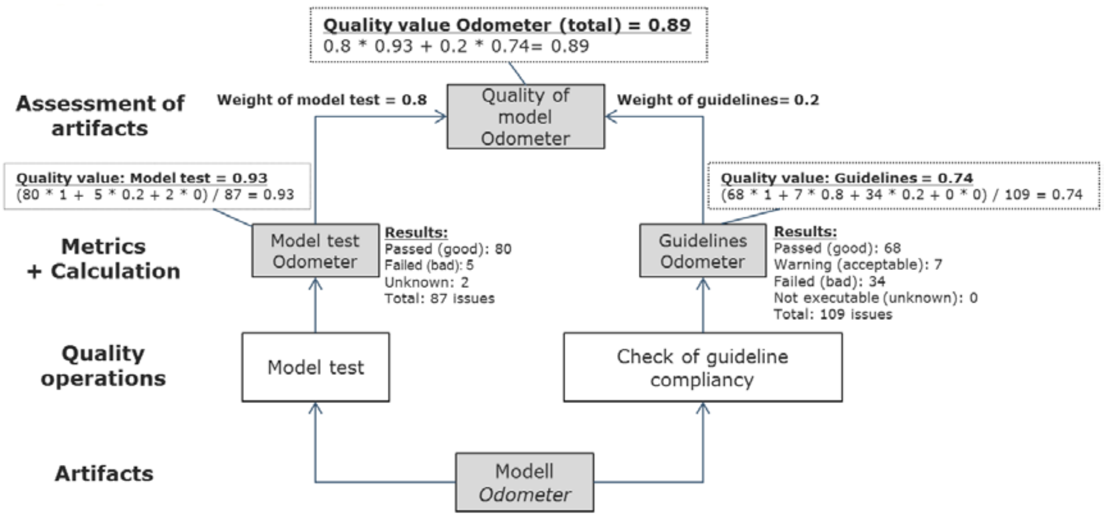


Figure 6: Aggregation of metric values of a quality operation

The approach for calculating quality values is now illustrated by assessing the quality of a project in which a Simulink and TargetLink Odometer model is developed (cf. Fig. 6). An *Odometer* is a driver assistance function from the automotive domain. This model contains different functions, which support the driver by giving him information about the actual distance driven, mileage, and overall travel time. In this example, only two quality operations are carried out for this model: model testing and modeling guideline checking. The model testing is carried out using the tool *MTest classic*[1], which performs functional testing (MiL, SiL) on model level. A TargetLink-implemented model (test object) is tested in a back-to-back test against the Simulink requirements models (test reference). Guideline

[1] http://www.mtest-classic.com

checking is carried out using MES' *Model Examiner*[2] [7]. The Model Examiner checks the TargetLink model against MAAB, MISRA AC TL, and TargetLink known problems, as well as MES functional safety-specific guidelines [8].

Table 1: Mapping of quality values to metric values

Metric value	Quality operation: model testing		Quality operation: guideline checking	
	Result	Value	Result	Value
Unknown	Unknown	0	Not executed	0
Bad	Failed	0.2	Failed	0.2
Acceptable	-	-	Warning	0.8
Good	Passed	1	Passed	1

The results of both quality operations are as follows:

(1) **Model testing:** *87* test cases were executed. *80* test cases provided a *pass*, *5* a *failed*, *2* an *unknown*[3] result. The results are mapped to the quality values and numerical values in Fig. 5 respectively: *passed=good (1)*, *failed=bad (0.2)*, *unknown=unknown (0)*. The mapping is also shown in Table 1. The overall quality value for the quality operation *model testing* was calculated using the formula in Fig. 5, resulting in a value of *0.93* for model testing.

(2) **Guideline checking:** A project-specific selection of 109 modeling guidelines was checked. *68* guidelines provided a *pass*, *7* a *warning*, *34* a *failed*, and *0* were *not executed*. The results are also mapped to the quality values and numerical values in Fig. 5 respectively: *passed=good (1)*, *warning=0.8*, *failed=bad (0.2)*, *unknown=unknown (0)* resulting in a quality value of *0.73* for guideline checking.

To assess the project overall, the decision was made to weight model testing with 0.8 and the results of guideline checking with 0.2. This results in a total quality of the Odometer model (project) at time t_i of 0.89. In this case, model testing was given a higher weight than the guideline checking for project-specific reasons. It is, however, quite common that all quality operations are weighted with the identical weight of 1. In safety-relevant projects it can be necessary to weight specific quality operations higher than others, e.g. coverage of safety requirements.

The methodology shown above can be expanded to aggregate the quality of individual artifacts or versions of a model over the duration of the project to provide an overall assessment of the project. The basic principle of aggregation is simplified and illustrated in Fig. 7 (cf. Fig. 2). The *1..n* software models (artifacts), which are part of a software project, are safeguarded by the quality operations *model testing* and *guideline checking*. Other quality operations such as reviews, code analyses, and other artifacts such as the generated code, the requirements specification, etc. are not shown in Fig. 7. However, the basic principle for aggregating quality values is identical and need not be described here in any greater detail.

As shown in the example for the Odometer model above (cf. Fig. 6), metrics of the individual quality operations are aggregated in one assessment. In so doing, not only the quality of artifacts, but also the quality of a quality operation itself can be assessed (e.g. quality of model testing). The quality

[2] http://www.model-engineers.com/en/our-products/model-examiner.html
[3] *Unknown* results may occur e.g. when a reference signal is not available.

assessment results from the weighted mean value of the individual quality operations. An assessment of many versions of an artifact can be used for tracking project quality during all the project phases (cf. Fig. 7, top right, project versions *v54* ... *v67* respectively).

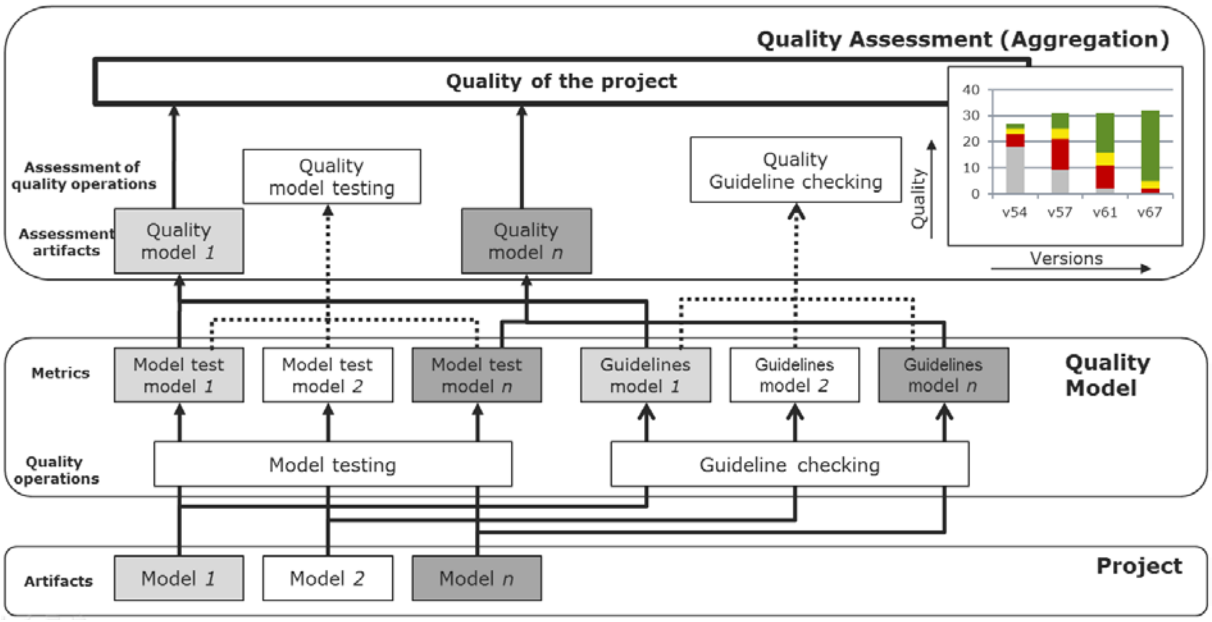
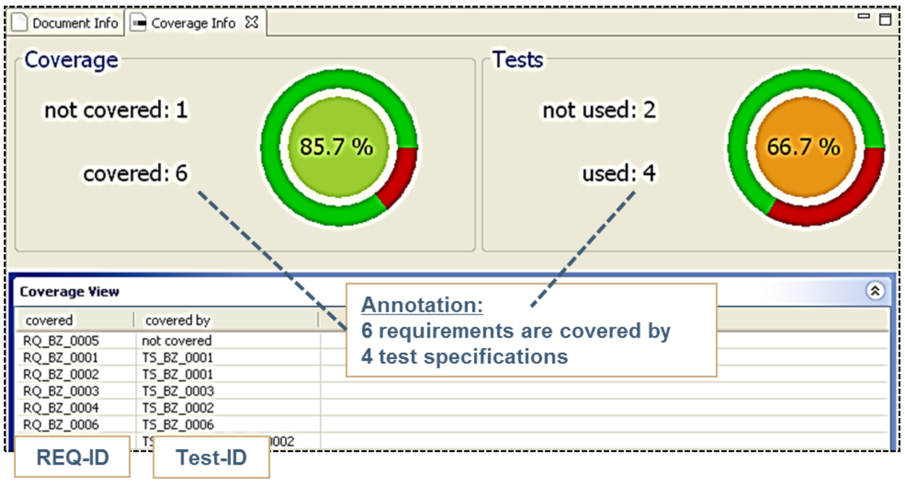


Figure 7: Aggregation of quality values for the quality assessment

2.5. Coverage of Artifacts

Metrics calculation and the quality assessment of the quality operations are however only one side of the story. One other factor must be considered for the purpose of assessing the overall quality of the project. From our point of view, the relation (i.e. coverage) of the artifacts to one another is essential for quality assessment. By *coverage*, we mean coverage of individual artifacts by quality operations. In order to do this, it is important to know how many requirements are covered by how many test specifications, how many requirements are realized within the model, etc. All this is relevant to getting an overview of the relationship between the artifacts. Moreover, coverage shows the percentage of (function) realization. As already discussed, it is essential to know how many requirements are actually implemented and tested for the purposes of quality assessment.



covered	covered by
RQ_BZ_0005	not covered
RQ_BZ_0001	TS_BZ_0001
RQ_BZ_0002	TS_BZ_0001
RQ_BZ_0003	TS_BZ_0003
RQ_BZ_0004	TS_BZ_0002
RQ_BZ_0006	TS_BZ_0006
	TS 002

Annotation:
6 requirements are covered by
4 test specifications

REQ-ID
Test-ID

Figure 1: Coverage view of artifacts (here: relation between requirements and test specification)

Fig. 8 shows the coverage view realized by our MQAC tool. The view shows how many requirements are covered by test specifications. In this case, 6 requirements are covered by 4 test specifications out of a total of 7 requirements. The IDs of the requirements and test specifications are automatically collected from external artifacts[4]. This view makes it easy to see that one requirement is not covered by a test case and that two test cases are not used in order to achieve the full requirements coverage.

3 Conclusions

In this paper we presented an approach for assessing the quality of model-based software projects. This approach is based on artifacts from the development process being safeguarded by quality operations. These operations provide (weighted) metrics, which are then used for the quality assessment of the artifacts. Apart from the quantitative collection of data from the quality operations, our approach also employs qualitative measures as part of the quality assessment. These measures include the degree of realization of requirements and the coverage of requirements by test cases. The Model Quality Assessment Center (MQAC) is capable of aggregating these metrics and providing a compact overview of all selected quality goals. The tool is already in use by selected German automotive OEMs for evaluation in serial production projects.

4 References

[1] MathWorks, (product information) http://www.mathworks.com/products, 2011.

[2] dSPACE: TargetLink – Production Code Generator at http://www.dspace.com, 2011.

[3] Broy, M., Kirstan, S., Krcmar, H., Schätz, B.: "What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?", in Rech, J., and Bunse, C. (Hrsg.) Emerging Technologies for the Evolution and Maintenance of Software Models, pp. 343-369, 2011.

[4] Fey, I., and Stürmer, I.: "Quality Assurance Methods for Model-based Development: A Survey and Assessment", SAE World Congress, SAE Doc. #2007-01-0506, Detroit, 2007. Also appeared in SAE 2007 Transactions Journal of Passenger Cars: Mechanical Systems, V116-6, Aug. 2008.

[5] Stürmer, I., Pohlheim, H., Rogier, T.: „Calculation and Visualization of Model Complexity in Model-based Design of Safety-related Software", (in German) in Keller, B. et. al. (Hrsg.), Automotive - Safety & Security, Shaker, pp. 69-82, 2010.

[6] Scheible, J., Kreuz, I.: „A Quality Model for the Automated Assessment of Model Quality for Embedded Systems", (in German) in Proc. of the 8. GI Workshop, Automotive Software Engineering, Leipzig, Germany, 2010.

[7] Stürmer, I., Stamatov, S., Eisemann, U.: "Automated Checking of MISRA TargetLink and AUTOSAR Guidelines", Proc. of SAE World Congress 2009, SAE Doc. #2009-01-0267, Detroit (USA), April, 2009.

[8] Model Engineering Solutions GmbH, „MES Strong Data Typing Toolbox – Guidelines and Checks", V 1.3, available at: http://www.model-engineers.com/en/our-products/model-examiner/sdt-toolbox.html

[4] Requirements are specified in IBM Rationale DOORS; the test specification in MS Excel.