Quality Model based on ISO/IEC 9126 for Internal Quality of MATLAB/Simulink/Stateflow Models

Wei Hu, Tino Loeffler, Joachim Wegener
{wei.hu, tino.loeffler, joachim.wegener}@berner-mattner.com
Berner & Mattner Systemtechnik GmbH
Gutenbergstr. 15, D-10587 Berlin, Germany

***Abstract*—In a model-based approach, models are considered as the prime artefacts for the software specification, design and implementation. Quality assurance for program codes has been discussed a lot, however equivalent methods for model quality assessment remain rareness. Assessing quality is of particular importance for technical models (e.g. MATLAB/Simulink/Stateflow models), since they are often used for production code generation. Our main contribution is a quality model based on ISO/IEC 9126, which defines the internal model quality as well as measures for the assessments. Our quality model shall not only show improvement potentials in model, but also provide evidence about quality evolution of a model.**

I. Introduction

In the last decade, model-based development has become common practice in a whole variety of branches and for a wide range of applications. In such an approach, models are considered as the prime artefacts for the software specification, design and implementation. In many domains (e.g. automotive and aviation), models become larger and more complex due to versatile functional and/or always stricter safety-related requirements. With growing size and complexity models become difficult to maintain and to alter, if they were not developed with adherence to distinct aspects of quality, such as readability, understandability and analysability. Further, executable models, such as MATLAB/Simulink/Stateflow models[1][2], are often used for (either manual or automatic) production code generation as well. Since debugging at a late development phase (i.e. after production code generation) is time consuming and costly, it is desirable to detect potential errors/faults at the earliest possible stage.

Surprisingly, although Simulink has established itself as de facto industry standard in model-based applications, there exist only few work concerning the quality, especially the non-functional quality of such models. Above all, there has not existed a clear definition of model quality yet. The state of practice regarding model quality is applying modelling guidelines. An example is the MAAB guidelines [1] for automotive applications, which defines various modelling patterns, naming conventions, rules of thumb (e.g. block usage, Simulink/Stateflow partitioning, subsystem hierarchies, ...). Such guidelines are useful for modellers as a reference for quality assurance task. However, neither are these guidelines related to any quality characteristics, nor do they provide any method/criteria for quality assessment. Existing model check tools (e.g. Model Advisor [2]) focus mainly on proper conditions and configuration settings and guideline conformance. Even though these tools are able to correct (some) rule violations automatically, they do not provide any indication about respective model quality either. Based on talks with modellers and feedbacks from automotive OEMs we notice that there is a huge need in rigorous model quality assessment methods.

Thus our research is aimed to analyse and to assess the internal (i.e. non-functional) model quality, especially the maintainability by means of static analyses. The purpose of this paper is to introduce a quality model which we have developed for Simulink models in automotive applications. Our quality model is derived from the international standard for software product quality ISO/IEC 9126-1 [3]. We focus on model maintainability and refined six well-chosen subcharacteristics. This paper will discuss how these quality attributes can be defined and decomposed regarding specialities of Simulink models, so that they can be measured and prepared for assessment.

The remainder of the paper is organised as follows: In Section II we give a brief overview of ISO/IEC 9126, after that, in Section III we share some observations on quality issues regarding practical Simulink models. In the following Section IV our refined quality model will be presented, whereat each quality characteristics is discussed in detail. In Section V our quality assessment prototype is briefly described. We treated related work in Section VI and conclude this work in Section VII.

II. ISO/IEC 9126

The standard ISO/IEC 9126 with title *Software engineering — Product quality* consists of four parts:

- ISO/IEC 9126-1: Quality model
- ISO/IEC TR 9126-2: External metrics
- ISO/IEC TR 9126-3: Internal metrics
- ISO/IEC TR 9126-4: Quality in use metrics

The part 1 [3] describes a two-part quality model for software product: a) internal quality and external quality, and b) quality

To appear in the 2012 IEEE International Conference on Industrial Technology (ICIT), Athens, Greece, 19-21 March 2012.

[1]MATLAB, Simulink, and Stateflow are registered trademarks of The MathWorks, Inc.

[2]For simplicity, we use *Simulink model* to refer model that was built with the MATLAB/Simulink/Stateflow toolkit in the rest of this paper.

in use. For internal and external quality six characteristics (*functionality*, *reliability*, *usability*, *efficiency*, *maintainability* and *portability*) are specified and each characteristic is subdivided into several subcharacteristics (cf. Fig. 2). As supplementation to part 1 the remaining three parts (technical reports) provide a suggested set of software quality metrics from perspective external quality, internal quality and quality in use, respectively.

Since this quality model is rather generic, it may be applied to any software product by tailoring to a specific purpose [4]. However, the metrics defined in ISO/IEC 9126 "provide only guidance for *a posteriori* evaluation of characteristics based on effort and time spent on activities related the software product…are not measured on the system itself and lack predictive power" [5].

III. Observations on practical models

Simulink models in automotive applications can be roughly categorised into three groups: *physical models*, *behaviour models* and *implementation models*. The first two model groups are mainly used for simulation purpose while the latter is code-generation-oriented. Despite the different usages, we have observed some common issues by all kinds of models in practice. Several examples are showed in the following regarding three main model aspects (architecture, design, modelling):

- Model architecture
  - Subsystems contain too many[3] direct child elements (blocks, signal lines)
  - Excessive subsystem interfaces
  - High subsystem hierarchy (nesting) level
  - Marginal (or even no) description/comment for the modelled plant/functionality
- Model design
  - Copy&paste model parts
  - Hardly comprehensible/understandable data and control flow (e.g. due to excessive use of Goto/From connections)
- Modelling
  - Non-uniform naming conventions and block colourings

IV. Internal model quality

The importance of software maintainability has been widely acknowledged (e.g. [6], [5], [7]). As mentioned in the introduction section, Simulink models are likely to be used for automatic production code generation. Although a good code generator might be able to make certain optimisation during the process, negative impacts due to bad modelling style(s) are still non-negligible. Further, either for model review or for (dynamic) model test it is required that a model has to be well understandable, which is unfortunately not always the case (Fig. 1 shows such a "bad" example from the real world.).

[3]As a rule of thumb, "not too many" means that all blocks/lines/names of a subsystem should be clearly readable when printed on one page A4 paper.

Another key issue in especially automotive applications is variants management ([8], [9], [10]), the authors of [11] demonstrated that a powertrain control application with 218 algorithm model libraries might result in 3488 possible component realizations. Whether a model is modular built, i.e. whether a model is simply adaptable, plays an important role on the maintenance effort.

Based on above considerations, we developed a quality model containing six characteristics for assessing internal quality of Simulink models: *analysability*, *changeability*, *stability*, *testability*, *understandability* and *adaptability* (cf. Fig. 2). We have excluded several quality (sub-)characteristics defined in ISO/IEC 9126 from our quality model, either because those characteristics are not applicable to Simulink models or respective evaluations are out of scope of static analysis. In the following, each of these characteristics will be outlined, respective alignments in Simulink models are described, some concrete analysis methods are presented as well. Note that some characteristics are more or less correlated to each other, i.e. some modelling aspects such as subsystem partitioning have influence on all six characteristics. What we will describe are those main factors which we believe play key role on the respective characteristic.

A. Understandability

Whether a model is understandable or not, the answer may vary from person to person, because understanding is an internal process of humans. However, there do exist some common factors which have influences on understanding a Simulink model.

- *Model hierarchy* The Simulink software offers a kind of blocks, i.e. *Subsystem*, which is able to group blocks as well as other subsystems together and build a hierarchy level. Although these hierarchies may be either solely graphical or both graphical and functional[4], it makes no difference from the viewpoint of human understanding. Partitioning a model into meaningful small subsystems will considerably improve its readability and also understandability. Unfortunately there exists no published approach for controlling or analysing such subsystem partitions. As state of the practice, the building of model hierarchies depends predominantly on modellers themselves. Mancoridis et al. proposed an approach for automatic recovery of the modular structure of a software system from its source code [12]. Their idea was to rebuild the module boundaries to increase cohesion within one module while at the same time reduce coupling between modules—an optimization problem of finding a best solution of high cohesion and low coupling for the modules. The basis of this analysis is a so-called Module Dependency Graph, the nodes of which denotes

[4]There are two kinds of subsystem blocks, virtual and non-virtual. Virtual subsystem blocks offer only graphical organisation of blocks, these subsystem borders will be flattened during model simulation and/or code generation. While non-virtual subsystems are both graphical and functional.

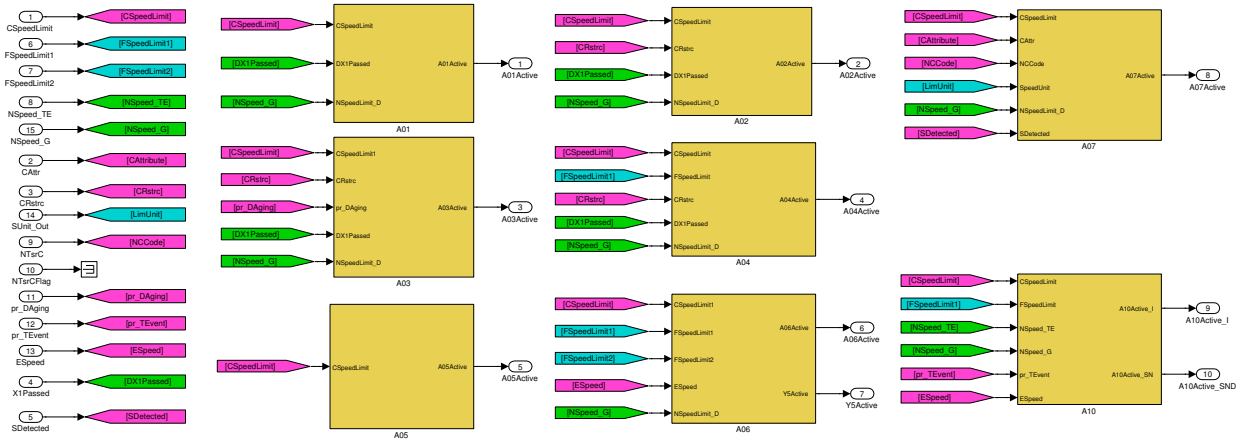

Fig. 1. A real world model from automotive applications (all signal names have been changed for confidential reason). Problems of this model include: 1) no description of the modelled functionality, nobody (probably except the modeller him- or herself) knows what does this module do; 2) many unorganised I/O signals (15 input and 10 output signals) lead to a complex module interface; 3) the I/O signals are not documented (e.g. meaning, unit and value range) that makes it difficult to conduct tests on them; 4) understanding data flow and dependencies between submoduls is difficult because of excessive use of Goto/From connections; 5) meanings of different block colourings are not described, which may lead to misinterpretation; 6) so many blocks in one model window that without zooming in they are hardly readable.

the software modules and the edges represent the dependence between the modules. Because Simulink models are based on a data-flow paradigm and consist of blocks and signal lines, which can be easily converted into nodes and edges, respectively, to build a dependency graph. For a more precise result, weights for different block types may be applied, data dimension of each signal line may come into consideration as well—a dependency graph with weighted nodes and edges. Having a dependency graph derived from model we might apply the established optimisation algorithms to search for a best partition of subsystems with high cohesion and low coupling. Compare the calculated subsystem partition with the existing one, we shall gain solid knowledge about the model hierarchy for assessment at a further step.

- *Model design* Deploying appropriate modelling instruments may greatly enhance understandability—naturally is to use Simulink blocks for numerical operations and use Stateflow for logical operations and scheduling tasks. An intuitive method is to identify whether a function is mainly logical or numerical operation related. A main problem with such an analysis on existing models is how to set boundary for a complete function, i.e. which blocks and signal lines belong to this function and which not. An applicable solution for this problem, is to use slicing techniques to analyse data flow and control dependence of the blocks [13]. Based on the dependence analysis results we may separate one function from the others, then we are able to determine whether a function was modelled with appropriate blocks.
- *Documentation* This is a often neglected aspect in practice. Because many modellers think that a Simulink model itself is already the documentation (due to its graphical nature). Unfortunately, this is not the case (cf. Fig. 1). An undocumented model may be more difficult to understand as a well commented source code program on the same algorithm. Ideally, each Simulink subsystem should be documented/commented (just as one is required to do so for each source code function). Thus here a measure similar to metric *comment rate* for conventional programming language can be applied.

There are also further aspects relating to model understandability, such as *signal line grouping*, *signal routing*, *naming conventions*, *block layouts*. Except that naming convention is more or less defined within a company or a project, to assess other three aspects empirical methods are required, e.g. considering the number of blocks, number of signals and model size.

B. Analysability

A model is easily analysable means that the effort required to diagnose and localise model parts and error/fault causes is minimum. It is assumed that the more complex[5] a model the more difficult is it to analyse.

- *Model complexity* (rather general description than concrete metrics, which shall be topic of next paper) We are mapping several established complexity metrics for conventional programming languages—Halstead metric [14], McCabe's metric [15] and Henry & Kafura's metric [16]—to assess model complexity. We use Halstead metric to calculate model *computational complexity*; Use McCabe's metric to analyse model *structural complexity*; Use Henry & Kafura's metric to determine model *informational complexity*. All three complexities are combined together to give an overall assessment for a whole model or for a certain subsystem.

[5]In this work, we focus on system complexity, i.e. complexity related to model structured design. The complexity of a algorithm itself is out of consideration.

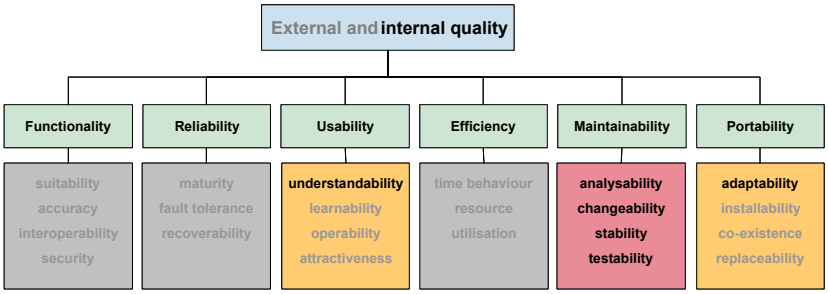

Fig. 2. Breakdown of the external and internal quality into 6 main characteristics according to ISO/IEC 9126-1. Each characteristic is further subdivided into several sub-characteristics. In this paper, we focus on all 4 subcharacteristics of maintainability, i.e. analysability, changeability, stability, and testability, plus understandability and adaptability within usability and portability, respectively.

- *Traceability* Simulink models and/or libraries are often versioned. It is important not only to document model changes separately (e.g. in a separate document), but also to note down the changes in model directly, which will ease a model analysis. Unfortunately, this analysis of this aspect mostly can only be carried out semi-automatically, e.g. via search for keywords.

C. Changeability

Changeability describes the effort needed to make change on model or to debug. Besides aforementioned well-defined model hierarchy, two main influence factors may be observered:

- *Model clone* A vital influence on model change effort, is played by duplicated model parts, i.e. copy&paste model block groups. What makes a changing harder is that model clones may have variants, i.e. there may exist similar yet not 100% identical model parts. To indentify such exact and approximated model clones exist already several approaches (e.g. [17] [18]), however, how to reduce the number of "false positives" remain an open research problem. As of an assessment, generally speaking, the number of exact model clones should be minimized, as well as approximated model clones that up to certain similarity grade.
- *Data management* How data are defined (i.e. either as signal or as parameter) and where data are defined (i.e. either in *Base Workspace* or in *Model Workspace* or in *Mask Workspace*) have different consequence on changeability, these effects shall be analysed qualitatively in a empirical way.

D. Testability

Testability represents efforts needed to validate a model that was changed.

- *Module coupling* When all modules are isolated from each other, we will have an ideal model testability. The more one module depends on the other, the harder is it to test. To assess the module couplings we are to consider connectivities between the modules, i.e. inter-connections between subsystems. We adapted the proposed approach in [12] to Simulink models.
- *Module interfacing* A speciality in Simulink models is that signals may inherit properties such as *sample time*, *data type*, *signal dimension* from a predecessor block via back propagation. It is also possible to define signal properties explicitly. This "modelling flexibility" might result in undesirable model behaviour and makes it harder to test. We believe that each subsystem should have explicitly defined inports and outports. However, how far our assumption is applicable (or better saying, acceptable by modellers) in practice is still an open question.

E. Adaptability

Adaptability indicates efforts needed to adapt a model to different specified environments. This is especially meaningful for the above mentioned variants management. Besides the aforementioned *model hierarchy* and *data management* it is common practice to build those basic and/or variant functionalities as libraries or as model references. For this special use case we suggest a qualitative analysis based on empirical methods.

- *Model hierarchy*
- *Data management*

F. Stability

Stability is the robustness of a model against undesirable effects due to model modification. In other words, the model change impact should be hold as low as possible to achieve a great stability against model changes.

- *Change impact* To analyse change impacts it is inevitable to study data-flow and control dependence in model. To achieve this, we apply slicing techniques, as supposed in [13].

To apply above introduced quality model to Simulink models a set of metrics that capture different aspects of each quality characteristics are required. Since we concentrate on static analysis techniques for models, we do not include metrics that are only retrievable via dynamic tests in our calculation. Besides, by defining metrics we give preference to analyses which require no more input information than that one can get from a Simulink model directly. Some sample metrics are listed in Table I.

Characteristics	Metrics
Understandability	number of blocks
	number of signal lines
	number of signal line crossings
	number of Goto/From block pairs
	$1 - \frac{\text{number of blocks with annotation}}{\text{number of blocks should have annotation}}$
	blocks layout size/available display area
Analysability	subsystem complexity
	documentation rate of module change
	$\frac{\text{number of improper configured logical blocks}}{\text{number of all logical blocks}}$
	$\frac{\text{number of improper configured relational blocks}}{\text{number of all relational blocks}}$
Changeability	$\frac{\text{number of cloned blocks}}{\text{number of all blocks}}$
Testability	coupling degree between subsystems
	$\frac{\text{number of explicitly defined inports}}{\text{number of all inports}}$
	$\frac{\text{number of explicitly defined outports}}{\text{number of all outports}}$
Adaptability	$\frac{\text{number of referenced models}}{\text{number of all subsystems}}$
	$\frac{\text{number of linked library blocks}}{\text{number of all subsystems}}$
Stability	average slice size per input signal

TABLE I
LIST OF SAMPLE METRICS

V. QUALITY ASSESSMENT PROTOTYPE

To facilitate automatic quality assessment we have developed a prototype. Basic idea of our prototype is 1) to collect both explicit and implicit information of object Simulink model; 2) to calculate various metrics; 3) to assess the calculation results. Since Simulink model files (*.mdl) apply a text-based format, most model information (we call this *explicit model information*) is saved as plain-text and can be retrieved via a text scanner (so that a MATLAB installation is not necessary). For this task we adopted an open source Simulink library[6] that provides a Java parser for MDL files. Remaining model information (we call this *implicit model information*), such as contents of bus signals, propagated data type, may be queried via Java MATLAB interface. For a successful information collection it is required that object Simulink model must be compilable as well as an installation of MATLAB/Simulink/Stateflow. The calculation of metrics as well as respective quality assessment run then automatically.

We have conducted several case studies with real world models from automotive branches. The quality assessment results are not only plausible but also positively accepted by respective modellers. We shall present and discuss the case study results in detail in another paper.

VI. RELATED WORK

Following McCall et al. [19] and Boehm et al. [20] various approaches to software quality model have been proposed (e.g. [21], [22], [23], [24], [25]). Most of these quality models are based on a hierarchical structure consisting of factors, criteria and metrics. Dromey's quality model [26] defines a set of structural forms, a set of quality-carrying properties and a set of high-level quality attributes as well as interrelationships between them.

[6] http://conqat.cs.tum.edu/index.php/Simulink_Library

The quality model defined in ISO/IEC 9126 has been applied to various domains regarding quality evaluation. Kanellopoulos et al. [27] propose a methodology for source code quality and static behaviour evaluation of a software system. Bansiya and Davis [28] introduce a hierarchical model for the assessment of high-level design quality attributes in object-oriented designs. Heitlager et al. proposed a practical model for measuring software maintainability in [5], which discussed several problems with the measure *Maintainability Index* and presented a selection of measures and guidelines for aggregating and rating them. Zeiss et al. [29] present an adaptation of the ISO/IEC 9126 quality model to test specifications and show its instantiation for test specifications written in the Testing and Test Control Notation (TTCN-3).

Certain quality aspects of Simulink models have been studied as well. Stürmer et al. [30] applied Halstead metric [14] to calculate model complexity and acclaimed that their calculation results coincided with expert expectation values. Several approaches for clone detections for Simulink models may be found in [17], [18] and [31], which demonstrated satisfied clone coverage rates. A quality assessment approach for Simulink models may be found in [32], in which a FCM (Factor-Criteria-Metrics) framework for automatic model quality rating was proposed. Highlight of this approach is an extensible framework for model quality assessment, whereat the factors and criteria and metrics are arbitrarily expandable.

VII. SUMMARY AND OUTLOOK

Based on the ISO/IEC 9126 we have developed a quality model to assess internal quality of Simulink models. Focusing on maintainability, we outlined model analysis methods regarding analysability, changeability, stability, testability, adaptability and understandability. We also mapped several established measures for source code to Simulink models. The proposed quality model and metrics have been realised in a prototype and tested and refined with real-world models.

For further developments we are carrying out experiments on large models to validate the measures and to examine the scalability of our approach. We also look forward to develop automatic clustering techniques for Simulink models, so that we may give suggestions for model refactoring in future.

Quality assessment for Simulink models may not only be able to reveal weak points in models in a early development phase, in particular it may be integrated in the development process for a continuous observation of internal quality during the project development phase, it may also provide evidence about quality evolution of a single component.

ACKNOWLEDGMENT

This work is supported in part by the Investitionsbank Berlin (IBB) within the EFRE program.

REFERENCES

[1] MathWorks Automotive Advisory Board, "Control algorithm modeling guidelines using matlab, simulink, and stateflow," July 2007, version 2.1.
[2] The MathWorks, Inc., "Model advisor," http://www.mathworks.com.

[3] *ISO/IEC 9126-1:2001(E): Software Engineering - Product Quality - Part 1: Quality Model*, ISO/IEC Std., 2001.

[4] H. Jung, S. Kim, and C. Chung, "Measuring software product quality: A survey of iso/iec 9126," *Software, IEEE*, vol. 21, no. 5, pp. 88–92, 2004.

[5] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*. IEEE, 2007, pp. 30–39.

[6] S. Wake and S. Henry, "A model based on software quality factors which predicts maintainability," in *Software Maintenance, 1988., Proceedings of the Conference on*. IEEE, 1988, pp. 382–387.

[7] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J. Girard, "An activity-based quality model for maintainability," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, 2007, pp. 184–193.

[8] S. Thiel and A. Hein, "Modeling and using product line variability in automotive systems," *IEEE software*, pp. 66–72, 2002.

[9] A. Pretschner, M. Broy, I. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 55–71.

[10] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proc. Future of Software Engineering FOSE '07*. IEEE Computer Society, 2007, pp. 37–54.

[11] K. Butts, D. Bostic, A. Chutinan, J. Cook, B. Milam, and Y. Wang, "Usage scenarios for an automated model compiler," in *Embedded Software*. Springer, 2001, pp. 66–79.

[12] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner, "Using automatic clustering to produce high-level system organizations of source code," in *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*. IEEE, 1998, pp. 45–52.

[13] W. Hu, J. Wegener, I. Stuermer, R. Reicherdt, E. Salecker, and S. Glesner, "MeMo – methods of model quality," in *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII*, Germany, Feb. 2011.

[14] M. Halstead, *Elements of software science*. Elsevier Science Inc., New York, NY, 1977.

[15] T. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, pp. 308–320, 1976.

[16] S. Henry and D. Kafura, "Software structure metrics based on information flow," *Software Engineering, IEEE Transactions on*, no. 5, pp. 510–518, 1981.

[17] F. Deissenboeck, B. Hummel, E. Juergens, B. Schaetz, S. Wagner, J. Girard, and S. Teuchert, "Clone detection in automotive model-based development," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 603–612.

[18] H. Nguyen, T. Nguyen, N. Pham, J. Al-Kofahi, and T. Nguyen, "Accurate and efficient structural characteristic feature extraction for clone detection," *Fundamental Approaches to Software Engineering*, pp. 440–455, 2009.

[19] J. A. Mccall, P. K. Richards, and G. F. Walters, "Factors in software quality. volume i. concepts and definitions of software quality." GENERAL ELECTRIC CO SUNNYVALE CALIF, Technical Report, 1977.

[20] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt, *Characteristics of software quality*. North-Holland Pub. Co., 1978.

[21] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, pp. 483–497, 1992.

[22] W. Florac *et al.*, *Software quality measurement: A framework for counting problems and defects*. Citeseer, 1992.

[23] V. Caldiera and H. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, vol. 2, pp. 528–532, 1994.

[24] M. Jorgensen, "Software quality measurement," *Advances in engineering software*, vol. 30, no. 12, pp. 907–912, 1999.

[25] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," *Conceptual modeling: Foundations and applications*, pp. 363–379, 2009.

[26] R. Dromey, "A model for software product quality," *Software Engineering, IEEE Transactions on*, vol. 21, no. 2, pp. 146 –162, feb 1995.

[27] Y. Kanellopoulos, P. Antonellis, D. Antoniou, C. Makris, E. Theodoridis, C. Tjortjis, and N. Tsirakis, "Code quality evaluation methodology using the iso/iec 9126 standard," 2010.

[28] J. Bansiya and C. Davis, "A hierarchical model for object-oriented design quality assessment," *Software Engineering, IEEE Transactions on*, vol. 28, no. 1, pp. 4–17, 2002.

[29] B. Zeiss, D. Vega, I. Schieferdecker, H. Neukirchen, and J. Grabowski, "Applying the iso 9126 quality model to test specifications," *Software Engineering*, pp. 231–242, 2007.

[30] I. Stuermer, H. Pohlheim, and T. Rogier, "Berechnung und visualisierung der modellkomplexitaet bei der modellbasierten entwicklung sicherheitsrelevanter software," *Automotive–Safety & Security*, pp. 69–82, 2010.

[31] N. Gold, J. Krinke, M. Harman, and D. Binkley, "Issues in clone classification for dataflow languages," in *Proceedings of the 4th International Workshop on Software Clones*. ACM, 2010, pp. 83–84.

[32] J. Scheible, "Ein framework zur automatisierten ermittlung der modellqualitaet bei eingebetteten systemen," in *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VI*, Germany, Feb. 2010.