T U M

I N S T I T U T F Ü R I N F O R M A T I K

Tagungsband des 4. Workshops zur Software-Qualitaetsmodellierung und -bewertung

Stefan Wagner, Manfred Broy, Florian Deissenboeck, Peter Liggesmeyer, Juergen Muench

TUM-I1104
Februar 11

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-02-I1104-0/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©2011

Druck: Institut für Informatik der
Technischen Universität München

Vorwort

Qualität ist seit Beginn der kommerziellen Entwicklung von Software ein wichtiges Thema in Forschung und Praxis und diese Bedeutung verstärkt sich noch weiter. Heutige Entwicklungen stellen zusätzliche Anforderungen an verschiedenste Qualitätsaspekte dar. Beispielsweise führt die Durchdringung von kritischen Systemen, wie Flugzeugen oder Automobilen, zu immer höheren Sicherheitsanforderungen an Software. Der starke Anstieg der durchschnittlichen Code-Größen und die Langlebigkeit von Software-Systemen machen die Wartbarkeit zu einer wichtigen Eigenschaft. Die Beherrschung von Software-Qualität stellt somit ein wichtiges Ziel im Software Engineering dar. Diesem Ziel steht aber die Komplexität und Vielschichtigkeit von Qualität gegenüber.

Es existiert eine große Zahl an unterschiedlichen Sichten und eine entsprechende Vielzahl an Herangehensweisen zu diesem Thema. Für die praktische Anwendung in der Software-Entwicklung stehen aufgrund dieser Vielfalt überwiegend nur Insellösungen zur Verfügung, die keine ganzheitliche Behandlung des Themas ermöglichen. Beispielsweise sind trotz der engen Verbindung Bewertungen von Zuverlässigkeit und Nutzbarkeit typischerweise nicht integriert.

Ein verbreitetes Vorgehen zur Bewältigung dieser Probleme stellt die Verwendung von Qualitätsmodellen und daraus abgeleiteter bzw. damit in Beziehung gesetzter Bewertungen dar. Ein solches Vorgehen wird sowohl in der Forschung untersucht, als auch bereits in der Praxis angewendet. Es hat sich aber oft gezeigt, dass Standards, wie die ISO 9126, nicht direkt anwendbar sind und eigene Qualitätsmodelle für spezifische Situationen erstellt werden müssen. Dies resultiert in teilweise sehr unterschiedlichen Ansätzen zur Qualitätsmodellierung und -bewertung. Ziel dieses Workshops ist es, diese Ansätze vorzustellen und zu diskutieren. Hierbei bauen wir auf die Erfahrungen und Ergebnisse vorherigen Ausgaben des Workshops (2008, 2009, 2010).

Organisation

Der Workshop SQMB ’11 wurde in Zusammenarbeit der Technische Universität München und des Fraunhofer IESE organisiert. Der Workshop fand im Zusammenhang mit der Konferenz SE 2011 in Karlsruhe statt.

Organisatoren

Stefan Wagner, Technische Universität München
Manfred Broy, Technische Universität München
Florian Deißenböck, Technische Universität München
Peter Liggesmeyer, Fraunhofer IESE
Jürgen Münch, Fraunhofer IESE

Programmkomitee

Andrea Baumann, Universität der Bundeswehr, München
Klaus Beetz, Siemens
Manfred Broy, TU München
Florian Deißenböck, TU München
Gregor Engels, Universität Paderborn
Wilhelm Hasselbring, Universität Kiel
Jürgen Knoblach, BMW
Peter Liggesmeyer, Fraunhofer IESE
Jürgen Münch, Fraunhofer IESE
Dietmar Pfahl, Simula Research Laboratory
Markus Pizka, itestra
Jochen Quante, Bosch
Kurt Schneider, Universität Hannover
Dirk Voelz, SAP
Stefan Wagner, TU München
Rolf Ziegler, SAP

Keynote

Der Keynote-Vortrag wurde von Prof. Dr. Jochen Ludewig (Universität Stuttgart) zum Thema „Qualität und Qualitätsmodelle“ gehalten.

Modeling Quality Information within Business Process Models

Robert Heinrich, Alexander Kappe, Barbara Paech

University of Heidelberg, Institute of Computer Science,
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{heinrich, paech}@informatik.uni-heidelberg.de
kappe@stud.uni-heidelberg.de

Abstract. Business process models are a useful means to document information about structure and behavior of a business process. However, they do not aim at expressing quality information relating to business processes. Organizations are interested in the measurement and modeling of quality information for the enhancement of quality of business processes and supporting IT systems. This paper presents the results of an extensive literature and tool survey on modeling quality information within business process models.

Keywords: Business Process Modeling, Business Process Quality, Quality Model, Notation, Tool.

1 Introduction

Business process modeling is widely used within organizations as a method to increase awareness and knowledge of business processes and to deconstruct organizational complexity [2]. A business process model typically visualizes activities and their dependencies, involved actors and their communication with one another and external parties. In some cases, process models also capture information about data and resources (e.g. software systems) involved in the process. Therefore, a business process model is a commonly used means to express structure and behavior of a business process. Current business process modeling notations do not aim to model quality information (QI) such as information about maturity or time behavior of a business process. Thus, it is difficult to capture quality requirements at the modeling stage which results in increased costs and delays in the further development [22] of business processes and involved IT systems. It is desirable to capture as much QI as possible while modeling a business process to provide a comprehensive view on quality. A (graphical) expression of QI together with information on functionality within a single model would increase the modeler's focus on quality in the early stage of modeling and therefore prevent negative effects on the development of business processes and supporting IT systems.

In contrast to software product quality, which for example is standardized in the ISO/IEC 9126 quality model [11], there is no common quality standard for business processes. Therefore, we are developing a comprehensive quality model for business

processes that is based on software product quality standards [6]. Moreover, we are developing a concept to present the QI of our quality model within a business process model. Hence, we analyzed related work.

This paper discusses the results of an extensive literature and tool survey to present the state of the art in modeling QI within business process models and point out deficiencies of current approaches. As we want to model QI graphically, we focus on graphical modeling notations. The paper is structured as follows: in Section 2 we present the background of this paper by describing our ongoing research on a comprehensive and practically relevant quality model for business processes. Section 3 discusses current approaches and in Section 4 we investigate how current tools support the modeling of quality within business process models. Section 5 concludes the paper and sketches future work.

2 Background

This section presents a summary of our work published so far in [6] to provide the background of this paper. We are developing the comprehensive Business Process Quality Meta-Model (BPQMM) [6], [7] using characteristics we transferred from software product quality standards like ISO/IEC 9126. We introduced a hierarchical structure of QI defined as follows. A *business process quality characteristic* is a category of business process quality attributes, for example maturity (see Figure 1). A *business process quality attribute* is an inherent property of a business process that can be distinguished quantitatively or qualitatively, for example the error density of an activity. A *business process quality measure* is a variable to which a value is assigned as the result of measurement, for example the number of detected errors per activity. In the following we use the term QI as a superset of characteristics, attributes and measures.

Business process quality refers to the components of a business process. Components are the activities of the process, the actors performing these activities, the objects handled and created by the process as well as the resources necessary for execution. As an activity can be subdivided into sub-activities, we consider a process itself as an activity. To each component of a business process we associated a set of quality characteristics. We took the ISO/IEC 9126 software product quality characteristics for resources and also adapted them for activities. For information objects we took the ISO/IEC 25012 [12] data quality characteristics. The actor characteristics we developed based on QI from practice. Figure 1 shows the BPQMM. The nodes correspond to the components and the characteristics are listed either within the node or on an edge between nodes. If the assessment of a characteristic depends on information of another component, we located it on the edge.

Some of the QI in our quality meta-model cannot be expressed within a business process model, for example *completeness of description* which is an attribute of the quality characteristic *understandability* (see [7] for details). However, we aim at modeling as much QI as possible together with information on functionality within a single model because having all information located at the same place facilitates the capturing of QI for the modeler.

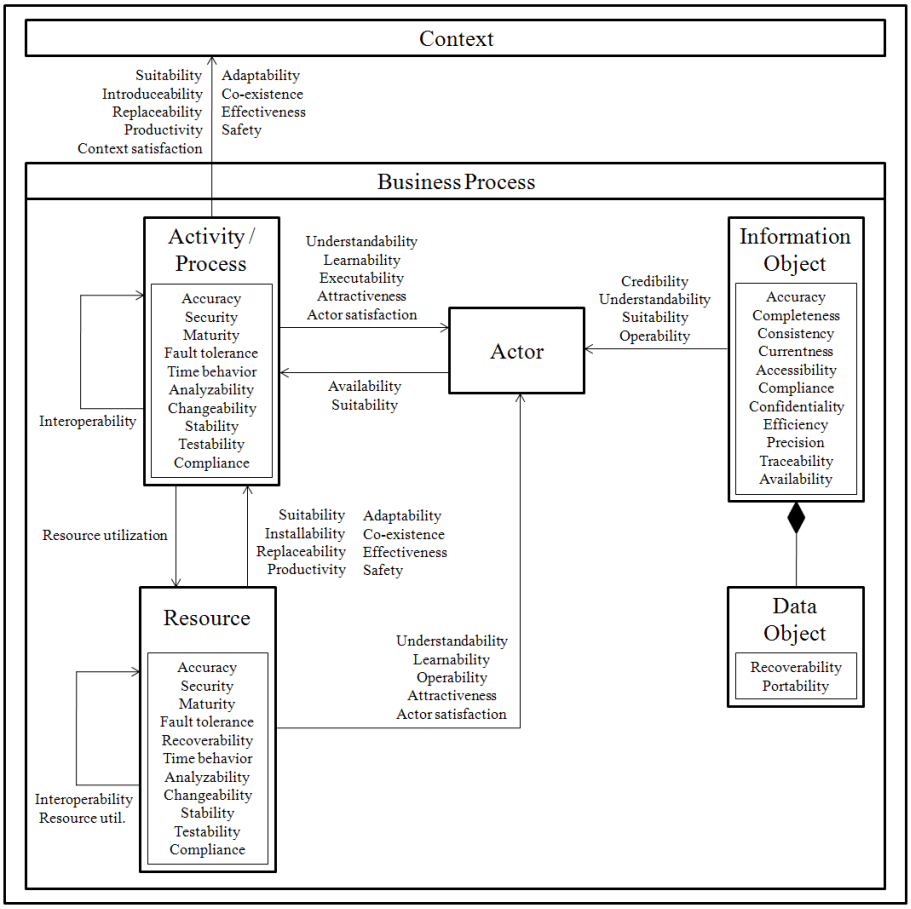

Fig. 1. Business Process Quality Meta-Model

3 Approaches to Modeling Quality Information

The approaches mentioned below are the result of an extensive literature research including the digital libraries of ACM (http://portal.acm.org), IEEE (http://ieeexplore.ieee.org) and SpringerLink (http://www.springerlink.com) because they give a reasonable confidence of covering the most relevant publications. Moreover, we utilized Google Scholar (http://scholar.google.com/) and the online catalogue of the university library which provides a variety of eBooks and eJournals. We used the following query: [‘business process (model)’ OR model OR graph OR diagram OR visualise OR visualize OR illustrate OR display OR picture OR depict OR represent OR capture] AND [‘quality information’ OR ‘quality characteristics‘ OR ‘quality requirements‘ OR ‘quality aspects‘ OR ‘quality properties‘ OR ‘quality attributes‘ OR ‘constraints’ OR ‘process characteristics’ OR ‘process properties’ OR

'non-functional requirements' OR 'NFR' OR 'goals' OR 'business rules' OR 'metric' OR 'measure' OR 'ratio'].

As a result from 129 relevant matches we finally selected 9 publications that describe approaches which graphically represent QI within a process model. In the following we give a short description of 6 out of these 9 approaches and compare them in Table 1. We limit to these 6 approaches because [20] and [21] present QI similar to [13] and the approach in [19] allows to model arbitrary information and does not make regulations to the QI to be modeled, so it is not considered in comparison. Further details on the allocation of QI to characteristics can be found in [14].

In [22] the authors propose an extension of the Business Process Modeling Notation (BPMN) 2.0 meta-model with basic QI on time, cost and reliability (see Figure 2). This approach enables to capture QI quantitatively in tabular form as an extension of the activity model element (shown as rounded boxes).

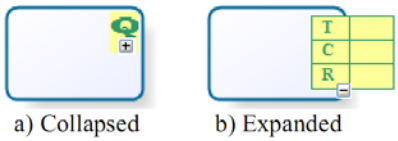

Fig. 2. Illustration of time, cost and reliability in [22].

Fig. 3. Extension of activity elements with performance information in [5].

The approach presented in [5] introduces a concept to present performance-relevant information within business process models using a mix of graphical and textual notation (see Figure 3). For each activity a set of performance indicators is calculated and visualized. The area of the circular icon at the lower left corner of the activity box is an average measure of the number of executions per month for the respective activity. The size of the dark pie of the circular icon is an average measure of the duration of the respective activity. The latest trends of throughput and duration are shown as arrows pointing upwards or downwards.

In [16] the author presents extensions to the BPMN, EPC and UML Activity Diagram for modeling process goals, cost, several QI on time and quality of activities. In the case of the UML Activity Diagram the QI is only represented textually inside the diagram. However, within a BPMN and EPC model the QI is also graphically represented inside the model by a circular icon besides the textual description, as shown exemplarily in Figure 4.

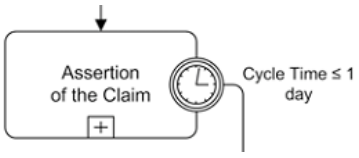

Fig. 4. An example of modeling QI in [16].

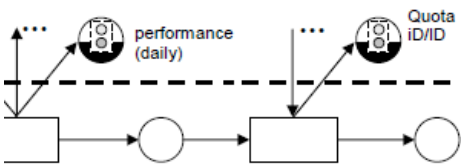

Fig. 5. Modeling performance indicators as places [18].

In [18] a concept for modeling performance indicators is proposed. It uses low-level Performance nets which extend traditional low-level Petri nets by the representation of performance indicators as places (see Figure 5). Once performance indicators are defined, they can be refined to machine-readable high-level Performance nets.

The graphical modeling notation Time-BPMN is described in [4] as an extension of the BPMN 1.2. Time-BPMN deals with the various temporal constraints and dependencies that may occur while characterizing real world business processes. Figure 6 presents an example of this notation by showing the activity "Complete Final Exam" which has a defined starting time, a Finish No Later Than constraint of 3 hours and a Start-to-Start dependency to the subsequent activity.

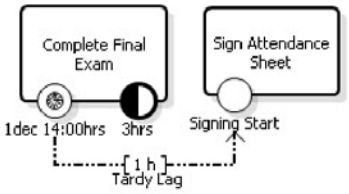

Fig. 6. Modeling temporal constraints and dependencies in Time-BPMN [4].

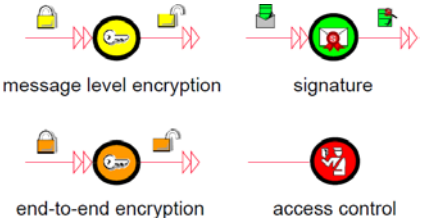

Fig. 7. Modeling artifacts for representing security information [13].

The approach described in [13] presents an extension to the ARIS SOA Architect that is capable of modeling security information. The extension enables the description of access control, data integrity, and confidentiality within a process model. In [13] the approach is applied to EPC models but it can also be applied to any other modeling notation. Figure 7 shows the security model symbols for each of the security properties. In [20] and [21] an approach to visualize similar security information is proposed. This approach is applied to UML 2.0 Activity Diagrams respectively BPMN models and uses a padlock symbol.

The approaches presented above extend existing process modeling notations by few or single QI such as time or security. However, they do not model a comprehensive set of QI as described in [7]. Altogether, regarding activities, the approaches contain QI covered by 7 of the 26 quality characteristics for activities in the BPQMM, namely time behavior, productivity, maturity, effectiveness, understandability, context satisfaction and security. Regarding information objects, QI covered by accuracy and confidentiality (2 of the 17 characteristics for information objects) can be expressed by the approaches. Regarding resources the approaches can only express security (1 of the 26 characteristics for resources). There is no actor characteristic expressible by the approaches. Note that we do not consider cost as a QI. In the following we summarize other important criteria used for comparison.

First we consider the granularity of the modeled information. The approach in [22] only allows the modeling of coarse-grained information within the diagram, e.g. time or reliability, whereas the other approaches allow the specification of much finer-grained information, e.g. waiting time or throughput of an activity.

Table 1. Comparison of current approaches on modeling quality within process models.

Approach/Criterion	Saeedi et al. [22]	Gulla [5]	Korherr [16]	Mevius [18]	Gagne et al. [4]	Jensen et al. [13]
Basic notation	BPMN	Arbitrary process modeling notation	BPMN, EPC, UML Activity Diagram	Petri nets	BPMN	EPC[1]
Way of extension	Graphical + textual	Graphical + textual	Graphical + textual	Graphical + textual	Graphical + textual	Graphical + textual
Expressible QI	Response time and reliability of activities	Performance indicators: average duration and number of executions per time unit (throughput) of activities	Cycle time, working time, waiting time, goals, complaints	Performance indicators for activities	Temporal aspects: time points, durations, temporal constraints and dependencies	Security aspects: access control for resources, encryption on message exchange, digital signature for information objects (integrity)
Covering quality characteristics from [7]	Time behavior, productivity, maturity (reliability[2]) (activity)	Time behavior, productivity, effectiveness (activity)	Time behavior, productivity, understandability, context satisfaction (activity)	Time behavior (activity)	Time behavior (activity)	Security (activity + resource) confidentiality, accuracy (information object)
Granularity of the Information	Coarse-grained	Fine-grained	Fine-grained	Fine-grained	Fine-grained	Fine-grained
Formality	Semi-formal	Semi-formal	Semi-formal	Formal	Semi-formal	Semi-formal
Maturity of the approach	New approach	Case study and tooling	New approach	Tooling	New approach	New approach

[1] Event-driven Process Chain

[2] According to ISO/IEC 9126 in [7] reliability is subdivided into the characteristics maturity and fault tolerance, and in the case of resources additionally recoverability. As in [22] for reliability only failures are considered, we only allocate maturity.

In [18] a formal approach is proposed while all others are semi-formal. A formal modeling including QI is necessary for analyzing, controlling, simulation and automation of the business processes.

The maturity of the approaches is another point we compare because there already might be hints on the appropriateness of the notation, the user acceptance or the benefit of applying the approaches in practice. However, as most of the approaches are rather new, there are no significant experiences to refer to. Only [5] conducts a case study including the prototypical implementation of the concept and [18] provides prototypical tooling.

4 Tools for Modeling Quality Information

Besides a literature survey on research approaches for modeling QI, we additionally analyzed current tools to understand the state of practice. Restricting our research only to business process modeling tools did not lead to satisfying results. These tools comply with a standardized process modeling notation (like BPMN or UML Activity Diagram) and as none of these notations allow the modeling of QI the tools do neither. Therefore, we extended our research to Business Process Management (BPM) systems and tools for Enterprise Modeling (EM). These tools usually include a process modeling component and additionally provide utilities to support other activities of BPM respectively EM, such as execution, monitoring, optimization or data modeling. The corresponding components are closely interconnected. Thus, the associated data is also more interconnected. That is the kind of data we want to visualize within business process models.

We finally analyzed[3] 42 BPM, EM and BPMN tools currently used in practice which we obtained from lists published by independent BPM-related organizations[4]. From these tools 16 enable the description of some kind of QI for business processes. However, we could not find a tool which satisfactorily enables the (graphical) modeling of QI within business process models as described above. Note that we do not consider prototypical tooling of approaches presented in Section 3 as we already discussed this QI in the previous section. Moreover, we are rather interested in tools used in practice than in research prototypes. In most cases QI can only be captured textually as a property of a model element in a tabular structure with predefined or free fields, or in separate views, which are not visible in the process modeling view. In few cases [1], [9], [17] the QI can be visualized as labels to the corresponding element and in [9] additionally with a freely selectable graphical symbol.

In Table 2 we compare five tools which allow the expression of QI. Due to the limited space we decided to present five representative tools which give an overview of the state of practice. Further tools can be found in [14]. For each tool we list the functional range, the supported process modeling notations, the expressible QI and the

[3] Analyzed means the tools were installed and executed, and associated white papers, tutorials and all sorts of published information material were consulted.

[4] An overview of current BPM- and BPMN-tools is available under http://www.bpm-netzwerk.de/content/software/listSoftware.do?view= respectively http://www.bpmn.org/BPMN_Supporters.htm

Table 2. Comparison of current tools from practice.

Tool/Criterion	ABACUS [1]	ADONIS [3]	Kern Process [15]	GRADE [10]	Horus [8]
Functional range	IT strategy, planning and EM	BPM	BPM	CASE	BPM
Modeling notation	BPMN	BPMN and own notation	Own notation	Own notation	Own notation
Expressible QI	- Process: processing time, frequency of execution, reliability, availability - Resource: utilization, business fit, reliability, availability, etc. - Actor: reliability, availability, etc.	Own notation: - Process: quantity of execution per time unit (throughput), tolerance waiting time (before cancelling) - Activity: processing-, waiting-, resting-, transport time	- Process: service time, customer satisfaction, employee satisfaction - Actor: qualification	- Activity: duration, goals	- Activity: quality (in %), error rate, processing- and transport time, execution frequency per time unit (throughput)
Covering quality characteristics from [7]	Time behavior, maturity, fault tolerance (because of reliability and availability[5]) (activity); resource utilization, suitability, maturity, fault tolerance, recoverability (because of reliability and availability) (resource); availability (actor)	Time behavior, effectiveness (activity)	Time behavior, productivity, context satisfaction, actor satisfaction (activity); suitability (actor)	Time behavior, productivity, understandability (activity)	Time behavior, maturity, effectiveness (activity)
Way of expression	Textual	Textual	Textual	Textual	Textual
Visibility within diagram	One property per model element (optional)	Not visible	Not visible	Not visible	Not visible

[5] According to ISO/IEC 9126 availability (just like reliability) is a combination of maturity, fault tolerance and recoverability.

covering characteristics of the BPQMM, and the way of expressing the QI.

Similar to the research approaches, the tools only allow the documentation of few QI and are not suitable to model quality comprehensively. Altogether, regarding activities, we found QI covered by 8 of the 26 quality characteristics for activities, namely time behavior, productivity, maturity, fault tolerance, effectiveness, context satisfaction, actor satisfaction and understandability. For resources, we identified QI covered by 5 of the 26 characteristics, namely resource utilization, suitability, maturity, fault tolerance and recoverability, and regarding actors, we found QI covered by availability and suitability (2 of the 2 characteristics). Regarding information objects, the tools are not able to express any characteristic. The tools as well as the research approaches do not support the hierarchical structure of QI as described in Section 2. This means they do not distinguish between characteristics, attributes and measures. This may lead to confusion of the concrete values that are to be captured. For example if an approach or tool requires the specification of reliability, it is not clear which attribute or measure should be used to specify the concrete value.

5 Conclusion and Future Work

In this paper we presented the results of a survey on approaches from research and tools from practice that targets to provide state of the art in modeling QI within business process models. Furthermore, we compared these results to the Business Process Quality Meta-Model. There are some approaches that are able to express few or single QI. However, we could not find an approach that is able to express a larger set of QI which is necessary for capturing quality requirements or doing process simulation comprehensively. Also tools from practice do not enable the capturing of QI satisfactorily; in fact, most of the tools we analyzed were not able to capture QI at all. Some of the tools were able to capture few QI but mostly these are not visible in the process model view. Our survey showed that there is a gap between the capability of current approaches and tools to present QI and the set of QI we want to capture within a business process model.

The deficiencies identified in the survey motivate us to develop a comprehensive concept on how to model QI within a process model [14]. The approaches and tools discussed in this paper provide first ideas however we need to extend them to capture a relevant set of QI. As a next step we plan to apply our concept to real processes to ensure practicality. Furthermore, we plan to provide tool support as an extension of the Eclipse-based CASE tool UNICASE [23].

References

1. Avolution Pty Ltd: ABACUS 3.2, http://www.avolution.com.au/ (Last Access: 2010.11.24)
2. Bandara, W., Gable, G.G., Rosemann, M.: Factors and measures of business process modelling: Model building through a multiple case study, European Journal of Information Systems Vol. 14 No. 4, pp. 347-360 (2005)

3. BOC Information Technologies Consulting AG: ADONIS 3.9, http://www.adonis-community.com/ (Last Access: 2010.10.26)
4. Gagne, D., Trudel, A.: Time-BPMN, In: Commerce and Enterprise Computing, CEC '09. IEEE Conference on, IEEE Computer Society, pp. 361-367 (2009)
5. Gulla, J.: Using Models in Enterprise Systems Projects, In: Krogstie, J., Opdahl, A.L., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, Springer Berlin Heidelberg, pp. 107-122 (2007)
6. Heinrich, R., Paech, B.: Defining the Quality of Business Processes. In Engels, G., Karagiannis, D., Mayr, H.C., eds.: Modellierung 2010, Lecture Notes in Informatics Vol. P-161, GI, pp. 133-148 (2010)
7. Heinrich, R., Paech, B.: Business Process Quality - A Technical Report, Technical Report, Software Engineering Heidelberg (2010)
8. Horus software GmbH: Horus Business Modeler 1.2.1, http://www.horus.biz/ (Last Access: 2010.12.17)
9. IDS Scheer AG: ARIS Design Platform 7.1, http://www.idsscheer.com/en/ARIS_ARIS_Platform/3730.html (Last Access: 2010.11.21)
10. INFOLOGISTIK GmbH: GRADE 4.1, http://www.infologistik.com/grade/ (Last Access: 2010.11.21)
11. ISO/IEC 9126-1: Software engineering — Product quality — Part 1: Quality model, First edition (2001)
12. ISO/IEC 25012: Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model, First edition (2008)
13. Jensen, M., Feja, S.: A Security Modeling Approach for Web-Service-Based Business Processes, In: Engineering of Computer Based Systems, ECBS 2009, 16th Annual IEEE International Conference and Workshop on the, IEEE Computer Society, pp. 340-347 (2009)
14. Kappe, A.: Entwicklung und Umsetzung eines Konzepts zur Modellierung von Qualitätsinformationen in einem Geschäftsprozessmodell, Master Thesis, Software Engineering Heidelberg (2011)
15. Kern AG: Kern Process 2.6, http://www.kern.ag/ (Last Access: 2010.12.17)
16. Korherr, B.: Business Process Modelling: Languages, Goals, and Variabilities, VDM Verlag, Saarbrücken, Germany (2008)
17. MEMOCenterNG build 2010-10-18, University of Duisburg-Essen, Chair of Information Systems, http://www.wi-inf.uni-duisburg-essen.de/FGFrank/index.php?lang=de_&&groupId=1&&contentType=Project&&projId=19 (Last Access: 2010.12.19)
18. Mevius, M.: A Novel Modeling Language for Tool-based Business Process Engineering, In: Proceedings of the 2008 ACM symposium on Applied computing, SAC'08, ACM, pp.590-591 (2008)
19. Pavlovski, C.J., Zou, J.: Non-functional requirements in business process modeling, In: Proceedings of the 5th Asia-Pacific conference on Conceptual Modelling, (APCCM 2008), CRPIT, 79. Hinze, A., Kirchberg, M. (eds.) ACS, pp. 103-112 (2008)
20. Rodríguez, A. Fernández-Medina, E. Piattini, M: Security Requirement with a UML 2.0 Profile, ARES 2006, pp. 670-677 (2006)
21. Rodríguez, A., Fernández-Medina, E.,Piattini, M.: A BPMN Extension for the Modeling of Security Requirements in Business Processes, IEICE Trans. INF & SYST., Vol.E90-D No. 4, pp. 745-752 (2007)
22. Saeedi, K., Zhao, L., Sampaio, P.R.F.: Extending BPMN for Supporting Customer-Facing Service Quality Requirements, In: Web Services, IEEE International Conference on, IEEE Computer Society, pp. 616-623 (2010)
23. UNICASE, Technische Universitaet Muenchen, Chair for Applied Software Engineering, http://unicase.org/ (Last Access: 2011.01.11)

Evaluating a Quality Model for Software Product Assessments – A Case Study

Michael Kläs[1], Klaus Lochmann[2], Lars Heinemann[2]

[1] Fraunhofer IESE
67663 Kaiserslautern
michael.klaes@iese.fraunhofer.de

[2] Technische Universität München,
85748 Garching, Germany
{lochmann, heineman}@in.tum.de

Abstract. *Background*: Software quality models have been proposed as a means for describing the concept of quality. Most quality models take an abstract view on quality characteristics. Therefore, they are not able to integrate measurement tools and metrics for conducting quality assessments of real software systems. To solve this problem, we developed a quality meta-model defining the structure of quality models that are detailed enough to specify quality characteristics and their links to metrics and measurement tools. *Aim*: In this paper, we present our evaluation of this meta-model in terms of its usability for constructing quality models that are suitable for quality assessments of real software systems. *Method*: For conducting the study, we developed an initial "proof-of-concept" quality model on the basis of static code analysis tools. This quality model was used for conducting quality assessments of Java-based software systems. The results were analyzed regarding two criteria: (1) the diversification provided by the results and (2) the congruence of the results with an independently conducted expert-based evaluation of the systems. *Results*: While the difference in the assessment results between the various systems is rather small, a correlation with the expert evaluation could be proven. Furthermore, the study provided useful insights for further work and improvements. *Conclusions*: We conclude that quality models based on the Quamoco meta-model are, in principle, capable of being operationalized for the automated quality assessment of software systems.

Keywords: Quality Model, Software Quality Evaluation, Empirical Study

1 Introduction

Software quality is a crucial factor for the sustainable success of a software product. To understand and manage the complex and multi-faceted concept of software quality, a number of quality models have been proposed, such as [1][2][3][6]. Unfortunately, most models take an abstract view on quality characteristics and are not detailed enough to enable an operationalization in terms of a quality assessment of real software systems. Besides the approaches for quality modeling, a variety of largely isolated software tools exist for measuring specific metrics related to quality.

The Quamoco[1] project aims at closing the gap between the abstract definitions in existing quality models on the one hand and analysis tools on the other hand by providing a quality meta-model that allows creating quality models that are operationalized to assess the quality of software products. A quality meta-model defines the principal structure of quality models; this means it contains the knowledge about how quality can be modeled. The concrete quality models using this structure contain specific knowledge about what constitutes quality in a certain context (e.g., project, company) and are therefore specific for the environment where they are employed. For instance, thresholds for certain quality metrics can be different depending on the respective application domain. Providing an appropriate quality meta-model is considered as important for defining consistent models that are useful for their defined application purpose [2][4].

Problem. The Quamoco quality meta-model should allow modeling concrete quality models that are detailed enough to perform product quality assessments. However, from discussions with industry representatives and form reviewing existing work, we are aware of a number of expectations regarding quality models appropriateness for performing quality assessments. The key criteria we identified are:

- The model supports *reliable* assessments. This means that if an assessment for a specific product is repeated, we obtain the same or at least a similar result.
- The model provides *valid* assessment results. This means the assessment results are in concordance with the results obtained by other (independent) quality evaluations of the assessed products.
- The assessments based on the model help to answer relevant questions by decision makers, in particular *'Which product is better with respect to quality in general or with respect to a certain quality aspect?'* This means that the results provided by the model have to *differentiate* between products of different quality.
- The model allows performing assessments in a *cost-efficient* manner.

Contribution. As a 'proof of concept' for checking whether quality models based on the proposed meta-model can fulfill these criteria, we created a first concrete quality model incorporating code measures that can be automatically determined by existing tools. This model and its structure are briefly described in Section 3. Since the model contains only measures automatically collected by tools and their evaluation is done in a fully automated manner using evaluation rules predefined in the model, we obtain repeatable and, accordingly, reliable assessment results. The high degree of automation also leads to a minimal amount of manual activities required to perform an assessment for a specific product and thus results in high cost efficiency. The two remaining criteria that should be fulfilled by a model in order to be useful for quality assessments – the *validity* of the model-based assessments (Goal 1) and the model's *ability to differentiate* products of different quality (Goal 2) – are evaluated in an empirical study presented in Section 4.

[1] *http://www.quamoco.de*

2 Related Work

A large number of quality models have been proposed in the literature, for example, [1][2][5][6]. These quality models define the term *quality* by decomposing it into more concrete quality attributes. However, they typically remain on a high level of abstraction and do not define how an actual quality assessment can be conducted using them. There is work on trying to establish a connection between those high-level quality models and measurement tools. For example, [9] and [10] developed an experimental quality model that specifies aggregation formulas needed for aggregating concrete measurement results. A more comprehensive approach for using quality models for quality assessments is being developed by the research project Squale[2], where researchers are developing an explicit quality model and a tool for evaluating software products. The main difference to our approach is that Squale uses a fixed quality model, whereas in Quamoco, the quality model can be edited, with the explicit meta-model guaranteeing that the structure of the created models is interpretable by the assessment tool chain. Moreover, Squale is limited to automated measures while Quamoco allows the seamless integration of the results of manual analysis activities like inspections and reviews.

Most existing work regarding quality models focuses on defining quality on a high level of abstraction. Work on using quality models for assessing the quality of software products is much more limited. Moreover, empirical evidence on quality assessments using these quality models is largely missing.

3 The Quamoco Quality Model

The quality assessment approach relies on a quality model that defines elements for specifying and measuring quality and for evaluating and aggregating the measurement results. The quality model is based on an explicit meta-model, whose main parts are summarized in the following; for further details, please see [4]. The quality model defines a product model of the software as suggested in similar forms in the literature [3][5]. The product model describes **Entities** and **part-of** and **is-a** relationships between them. When describing the quality of source code, typical entities in the model include *Class* and *Expression*, where *Expression* is further refined by *Relational expression* and *Mathematical expression*, which are in an *is-a* relation with *Expression*. The entities are characterized by **Attributes**, resulting in **Factors**. A factor is the central part of the quality model and describes a property of the software product with an influence on quality. A typical factor in the quality model is, for example, *Correctness* of *Relational expression*, which describes that a relational expression is correct if its operands have compatible types, units, scales, etc.

While factors describe properties of the software product, **Quality aspects** in the quality model describe the quality characteristics that are in the focus of the analysis, like the "-ilities" of the ISO 9126 [6]. The influence of factors on quality aspects is modeled as **Impacts**. For each impact an explicit justification and direction must be

[2] *http:// www.squale.org*

provided in prose text. An impact may, for example, be "The *Correctness* of *Relational expression*s has a positive impact on *Reliability*, because incorrect comparisons of data may cause arbitrary failures at runtime". For conducting a quality assessment, the factors specified in the model must be quantified by **Measures**. A measure specifies which data have to be provided either by a tool or by manual inspection in order to provide an assessment of a factor.

An important purpose of operationalized quality models is to specify how the collected measurement values are aggregated, normalized, and transformed in evaluations in order to yield a quality assessment in terms of quality aspects. These aggregations/normalizations and mappings to an evaluation scale can be specified in a domain-specific language named **QIESL** (quality impact evaluation specification language), which can be used to specify rules for **Impact evaluation** and **Quality aspect evaluation** elements in the model. This language has a Java-like syntax and provides predefined functions for special purposes, e.g. a function for calculating a linear distribution, or a function for calculating the proportion of methods affected by a certain type of defect detected by a measure. The structure of this meta-model should make it possible to express the contents of existing quality models. In a previous study [4], we analyzed the generality of the quality meta-model and concluded that it is able to express the concepts of a large range of different quality models applied in practice.

Tooling. We developed tool support for creating and editing quality models using Eclipse/EMF. The quality model editor supports the user in building quality models consistent with the Quamoco meta-model by automatically checking corresponding modeling constraints and providing different views.

For conducting automatic quality assessments, we developed tool support based on the framework ConQAT[3]. It loads a quality model, takes the data of external static code analysis tools like FindBugs[4], and evaluates and aggregates the measurement results based on the QIESL formulas specified in the quality model.

Base Model. In order to evaluate the meta-model regarding its operationalizability, we developed a concrete quality model that is compliant to the meta-model. This quality model, though limited in size, shows how the meta-model concepts are applied in a meaningful way. The quality model describes 24 factors for the programming languages C/C++ and Java. In this paper, we focus on Java; therefore, only 10 factors are relevant. These factors have 11 impacts on quality aspects. For each impact, a QIESL formula was specified in the corresponding impact evaluation. The evaluation results produced by these formulas are used as a basis for the further discussions in this paper. Due to reasons of brevity, the contents of the base model are not discussed in detail here. However, a browsable representation of the base model can be found at the Quamoco Web Portal[5].

Figure 1 illustrates an excerpt of the quality model. The factor *Structuredness* of *Class* is measured based on a rule of the tool FindBugs. The impact describes that the factor has an influence on the aspect *Analyzability*. The QIESL formula connected with the impact specifies how the measurement results for the factor are interpreted with respect to their impact on *Analyzability*.

[3] *http://www.conqat.org*
[4] *http://findbugs.sourceforge.net*
[5] *https://quamoco.in.tum.de/webportal/version1.php*

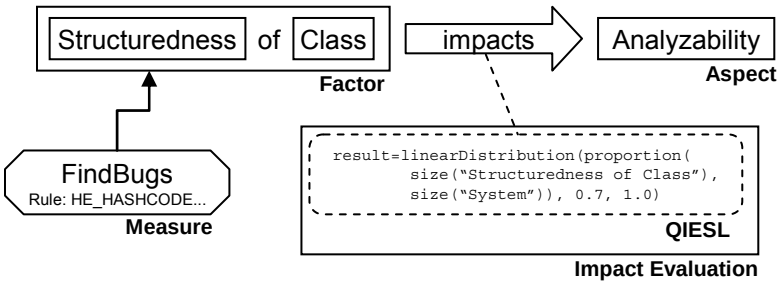

Figure 1: Excerpt of the Quality Model

4 Empirical Evaluation

This section describes the empirical evaluation of the appropriateness of the Base Model with respect to its application purpose, namely *assessing software quality*. We focus our investigation on the two aspects discussed in the introduction that have to be empirically evaluated. We first present the two corresponding study goals. In the following sub-sections, we derive criteria (and hypotheses) from these goals that will be check for the Base Model. For each criterion, we describe the evaluation procedure used and present and discuss the results obtained.

Goal 1 (Diversification). Evaluate whether the assessment results obtained by applying the Base Model provide a sufficient level of diversification between products with different quality levels to answer questions such as 'Which product is better with respect to quality in general or with respect to a certain quality aspect?'.

Goal2 (Validity). Evaluate whether the Base Model provides *valid* assessment results, meaning that the assessment results are in concordance with the results obtained by another independent and valid approach for assessing product quality.

4.1 Evaluation of Diversification

To the best of our knowledge, there are no commonly accepted criteria for an appropriate level of diversification in the area of software quality assessment. The diversification provided on the bottom level of the model, where the collected measurement values are mapped onto values on the evaluation scale (in our case a number from 1 to 6), strongly depends on the approach used to define the responsible mapping function.

Benchmarking-based approaches typically define specific evaluation functions by analyzing a sample of products and calculating some statistics for the resulting measurement values. If the sample is large enough and representative of the population of assessed products, these approaches can give any form to the resulting distribution and therefore can also provide a defined level of diversification (as far as the measurement results differ between the products in the considered population). In many cases, such approaches create a distribution that is similar to an equal or uniform distribution, by mapping the quantiles of the measurement data distribution onto the values of the evaluation scale (e.g., [11]).

Other approaches define the mapping between the measurement results and the resulting evaluations using the knowledge of experts or target values based on empirical studies or literature reviews as in the Base Model presented. In such cases, it is much more relevant to check the level of diversification provided because a sufficient level of diversification is not automatically assured by the approach.

In the area of operational research, where 'diversification' is a known concept, a series of measures are being discussed to determine diversification on a nominal or ordinal scale [8]. One of these measures commonly used is *entropy* (*E*), which originates from information theory and is defined as:

$$E = -\sum p_i \ln(p_i) \text{ for } i=1\ldots m, \text{ where } p_i \text{ is the probability to obtain scale level } i$$

E=0 means there is a probability of 100% to get the same assessment result for each product and, consequently, there is no diversification at all. On the other hand, a high value of E means the assessment results are well distributed across the scale levels. Since the maximum obtainable entropy depends on the number of levels offered by the scale, we can compute the *normalized entropy* (*e*) by dividing *E* with its maximum for a given number of scale levels (*m*), namely *ln(m)*, and obtain a value between 0 and 1:

$$e = -\ln(m)^{-1} \sum p_i \ln(p_i) \text{ for } i=1\ldots m$$

In order to use this equation to estimate the diversification provided by our evaluations, we have to approximate the probability values p_i for each scale level. We can do this by determining the ratio between the assessment results in the sample with level i (n_i) and the total number of results in the sample (n): $p_i = n_i / n$ for $i = 1 \ldots n$.

If we want to define a criterion for checking whether sufficient diversification is provided by the evaluations in the model, we first have to assume a certain kind of distribution of the assessment results. A maximal normalized entropy and thus diversification is provided by a perfect equal/uniform distribution of the results on the evaluation scale.

We consider a discretized normal distribution across the levels '1' to '6' with a mean of 3.5 and a variance of 1 (Figure 2) as the lower bound for an acceptable diversification. This means that around two-thirds of the assessments provide a '3' or '4' (<1σ distance) and around 5 percent a '1' or a '6' (>2σ distance). The corresponding normalized entropy is ~0.80 when measured for the total population. However, depending on the size of the sample used to estimate the p_i values, it would not be uncommon to obtain an *e* value of not more than 0.60 for this kind of distribution.

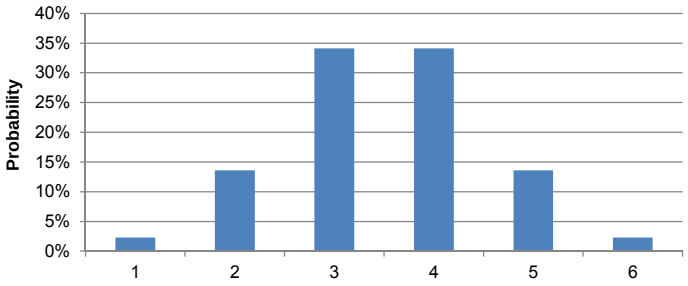

Figure 2: Normal distribution N(3.5,1) discretized on levels 1 to 6.

Hence, in the absence of other criteria, we use (as a rule of thumb) a threshold of ≤0.50 for samples between 10 and 15 assessed products as an indicator of inappropriate diversification. More accurate thresholds could be calculated by performing a simulation study using the actual sample size as input.

However, not only the entropy value but also the range of the evaluation results and their distribution over the evaluation scale should be considered, for instance by visualizing and checking them in a box plot chart.

Procedure: During the study, the Base Model was used to assess 13 software products written in Java. Since the model should be applicable for a broad range of software products, our test objects covered a range of open source projects different in type and size (JabRef 2.3, TV-Browser, RSSOwl, Log4j, Checkstyle, ConQAT, JabRef 2.5, Tomcat) as well as five closed source projects. Each product was assessed with respect to 8 factors influencing the product quality by determining the corresponding impact evaluation result using grades 1 to 6. For each impact evaluation, the distribution of the evaluation results was presented by a box plot and the normalized entropy (e) was calculated.

Results: Figure 3 compares the discretized normal distribution that was used as a baseline with the results of a selection of impact evaluations. In total, five impact evaluations in the Base Model such as 'Technical name x Conformity', which is presented in Figure 3, rated all products with the best grade '1' resulting in e = 0. Despite not using grades '5' and '6', the evaluation for 'Class Comment x Consistency' provided good diversification across the remaining grades resulting in an acceptable entropy value (e=0.72). In general, we could observe that the results of most evaluations tend towards the lower half of the scale (i.e., grades '1' to '3').

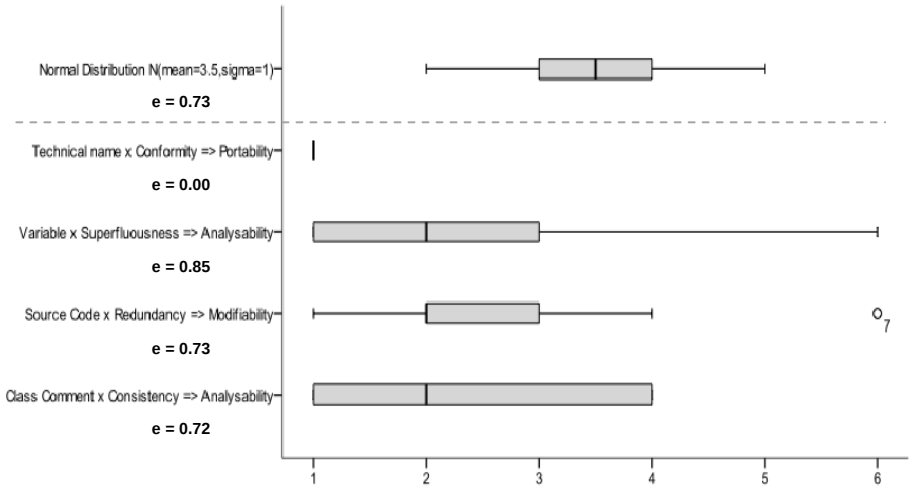

Figure 3: Discretized normal distribution N(3.5,1) compared to the results of selected impact evaluations for n=13 product assessments.

Interpretation: In general, we see three possible reasons for the lack of diversification provided by certain impact evaluations. (1) The assessed products may be all very similar (in our case excellent) with respect to a factor without diversification (e.g., 'Technical name x Conformity'). (2) The factor might not be sufficiently operationalized by the measures collected. This can mean that the measures do not cover all relevant aspects of the factor or that they measure only issues that rarely occur in practice. (3) The evaluation function used to map the measurement values to a specific grade is inappropriate (i.e., it does not differentiate sufficiently between good and bad results). Since we assessed a broad range of different products, it is likely that either the chosen measures are not sufficient for covering the factor or the mapping (including the normalization) does not differentiate sufficiently. To precisely identify the possible reasons, more tests are necessary. The missing diversification provided by certain factors seems not to be a general problem with the meta-model since other factors provide sufficient diversification. However, the results already indicate that the sensitivity of various impact evaluations has to be increased in order to provide better differentiation between products of varying quality on the factor level.

Threats to Validity: Since we could not determine the quality of the assessed products with respect to each factor independently of the evaluations provided by the model, the major validity threat is that the 13 assessed products might be too similar with respect to certain factors to provide good differentiation for each factor without making the assessment results instable due to oversensitive evaluation functions.

4.2 Evaluation of Assessment Validity

In order to evaluate the validity of the model-based quality assessments, we need an independently obtained criterion for product quality that we can compare with our assessment results. Since no measurement data were available that directly measure the quality or the quality aspects of interest for the assessed products, we used as the independent criterion an expert-based quality rating provided in the 'Linzer Software-Verkostung' [7] for a set of five open source products. The rating is a combination of ratings provided independently by nine experienced Java experts.

In the IEEE standard [12], several *validity criteria* are proposed for validating software quality metrics. Most of them assume that the collected measures and the independent criterion both use an interval or ratio scale. However, while the results of the Base Model assessments are provided as a value characterizing the product quality between 1 (best possible) and 6 (worst possible), the assessment results of the Linzer Software-Verkostung are provided on an ordinal scale as a ranking from best (1) to worst (5) product. Consequently, we had to limit our investigation to the validity criterion '*consistency*' [12], which can be applied on interval scale data. In our case, it will characterize the concordance between a product ranking based on the assessments provided by our model and the ranking provided independently by a group of experts. This means that ***we determine whether the Base Model can accurately rank the set of assessed products with respect to their quality*** (as perceived by experts).

Following [12], we measure consistency by computing the *Spearman's rank correlation coefficient* (r) between both rankings, where a high positive correlation means high consistency between the two rankings. Since we want to check whether a poten-

tially observed positive correlation is just due to chance or is a result of using an appropriate quality model, we state the (alternative) hypothesis H_A with alpha = 0.05:

H_A: There is a positive correlation between the ranking provided by the Base Model (BM) and the ranking provided by the "Linzer Software-Verkostung" (LSV).

$$\mathbf{r\ (\ ranking_{BM},\ ranking_{LSV}\ ) > 0} \quad [\ \text{i.e., } H_0\text{: } r\ (\ ranking_{BM},\ ranking_{LSV}\ ) \leq 0\ ]$$

Procedure: During the study, the Base Model was used to assess the quality of five open source products for which results of the Linzer Software-Verkostung were available: JabRef 2.3, TV-Browser, RSSOwl, Log4j, and Checkstyle. We had to limit our evaluation to these five products since the remaining seven products used for the evaluation of diversification were not part of the Linzer Software-Verkostung. Furthermore, there was no other independent validity criterion that we could use.

For the sake of simplicity, we assumed in the Base Model that each factor has the same relevance for the overall perceived product quality. Thus, the aggregated assessment result for each product corresponds to a weighted sum with an equal weight for each factor. Taking the sum, we implicitly assume the same distance between the different grades (1 to 6). This means that the difference in quality between a product with grade 1 and a product with grade 2 is assumed to be equal to the difference in quality between a product with grade 2 and a product with the grade 3.

In a final step, the assessed products were ordered by the results for their overall quality provided by the Base Model and compared with the ranking provided by the Linzer Software-Verkostung.

Results: Figure 4 shows the assessment results using the Base Model and the resulting product ranking as well as the ranking of the Linzer Software-Verkostung. The calculated Spearman's rho correlation is ***r = 0.975***, which is close to a perfect correlation of 1. Hypothesis $\boldsymbol{H_A}$ can also be ***accepted*** on a high level of significance (p=0.002) meaning that there is a significant positive correlation between the ranking provided by the Base Model and the ranking provided by the Linzer Software-Verkostung.

Assessed Product	Result using BM	Ranking by BM	Ranking by LSV
Checkstyle	1.00	1	1
Log4j	1.22	2	2
RSSOwl	1.44	3	3
TV-Browser	1.44	3	4
Jab-Ref 2.3	1.89	4	5

Figure 4: Comparison of the assessment results and 'Linzer Software-Verkostung'

Interpretation: Despite the partly missing differentiation of the assessment results on a lower level of granularity (i.e., with respect to specific factors), the assessments of the overall product quality turn out to be consistent and thus valid when compared to an independent criterion for quality, in this case provided in the form of an expert-based assessment. This indicates that the Quamoco meta-model can be used to specify quality models that provide valid automated quality assessments of software systems. Although this conclusion is supported by a very high and statistically significant correlation, there are some threats to validity that need to be considered.

Threats to Validity: The most relevant threats we see are (1) we cannot guarantee that the criterion chosen for the validation, namely the expert-based quality rating (of 9 experts reviewing each product), adequately represents the quality of the products, (2) the generalizability of our results is limited by the fact that the number of assessed products (5 systems), their type (Java, Open Source), and the factors considered in the assessment are limited.

5 Conclusions and Future Work

In this paper, we investigated the question of whether the meta-model proposed by the Quamoco project can be used to define quality models that are appropriate for quality assessments of real software systems. In order to answer this question, we developed an initial quality model (Base Model) using the proposed meta-model and evaluated in a case study whether the quality model can diversify software products of different quality on the level of quality factors. Moreover, we compared the results of the assessment with an independent quality ranking of the products performed by experts.

In the model, we identified five factors for which we had to reject our assumption that the model is able to differentiate between products of different quality with respect to these factor. This leads us to the conclusion that the sensitivity of the affected impact evaluations needs to be increased. However, our results also indicate that the model provides a quality assessment in line with the findings of an independent expert group. We found a very high and also statistically significant correlation between the manual quality ranking and the results of the automated assessment based on the quality model.

We conclude that the Quamoco meta-model can be used to specify quality models that provide valid automated quality assessments of software systems. As future work, we plan to refine the impact evaluations in order to achieve better results with regard to diversification among software systems. Moreover, we plan to extend the quality model to include more quality characteristics and measurements.

Acknowledgments

This work was supported in part by the German Federal Ministry of Education and Research (Quamoco Project, No. 01 IS 08 023 C). We would also like to thank Sonnhild Namingha for reviewing a first version of this article.

References

[1] Boehm, B. W. (1978): Characteristics of Software Quality, North-Holland.

[2] Kitchenham, B.; Linkman, S. G.; Pasquini, A.; Nanni, V. (1997): The SQUID approach to defining a quality model. In: Software Quality Journal, Vol. 6, Issue 3, pp. 211–233.

[3] Deissenboeck, F.; Wagner, S.; Pizka, M.; Teuchert, S.; Girard, J.-F. (2007): An Activity-Based Quality Model for Maintainability. In: Proc. of the 23rd Int. Conf. on Software Maintenance (ICSM 2007).

[4] Kläs, M.; Lampasona, C.; Nunnenmacher, S.; Wagner, S.; Herrmannsdörfer, M.; Lochmann, K. (2010): How to Evaluate Meta-Models for Software Quality? In: Proc. of the joined Int. Conf. on Software Measurement. IWSM/MetriKon/Mensura 2010, Shaker, pp. 443-462.

[5] Dromey, R. G. (1995): A model for software product quality. In: Software Engineering, IEEE Transactions on, Vol. 21, Issue 2, pp. 146–162.

[6] ISO/IEC. 9126-1 (2001): Software engineering – Product quality – Part 1: Quality model

[7] Plösch, R. (2010): Software-Verkostung: Ein neuer Ansatz zur Validierung von Software Qualitätsmanagement-Werkzeugen. http://www.ipo.jku.at/dokumente/upload/Software%20Verkostung%20Impulsvortrag.pdf.

[8] Troutt, M. D.; Acar, W. (2005): A Lorenz-Pareto measure of pure diversification, European Journal of Operational Research, Vol. 167, Issue 2, pp. 543-549, ISSN 0377-2217, DOI: 10.1016/j.ejor.2004.02.022.

[9] Schackmann, H.; Jansen, M.; Lichter, H. (2009): Tool Support for User-Defined Quality Assessment Models. In: Proc. of MetriKon 2009, Shaker.

[10] Marinescu, C.; Marinescu, R.; Mihancea, R. F.; Ratiu, D.; Wettel, R. (2005): iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design. In: Proc. of 21st IEEE Int. Conf. on Software Maintenance - Industrial and Tool volume (ICSM '05).

[11] Gruber, H.; Plösch, R. (2010): On the validity of benchmarking for evaluating code quality. In: Proc. of the joined Int. Conf. on Software Measurement. IWSM/MetriKon/Mensura 2010, Shaker, pp. 463-481.

[12] IEEE. 1061 (1998): Standard for a Software Quality Metrics Methodology.

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.

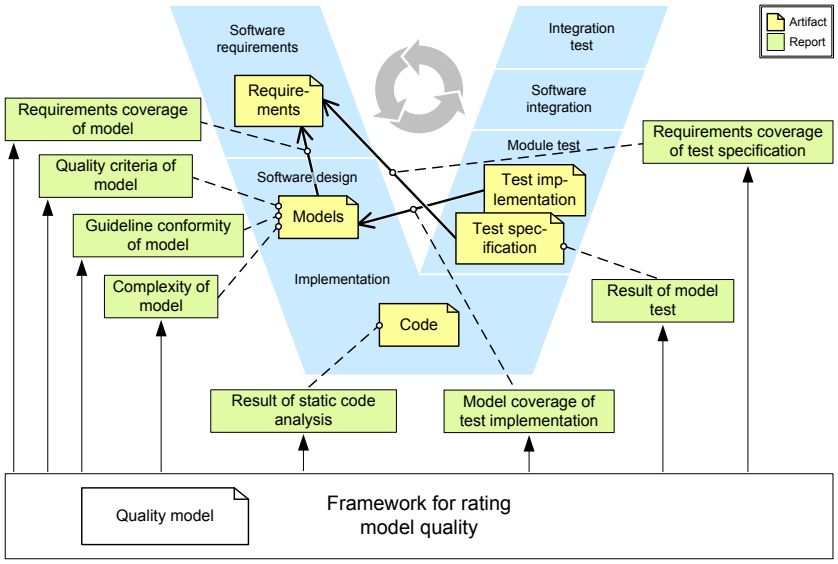

Fig. 1. Development process with artifacts and analysis for quality rating

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.

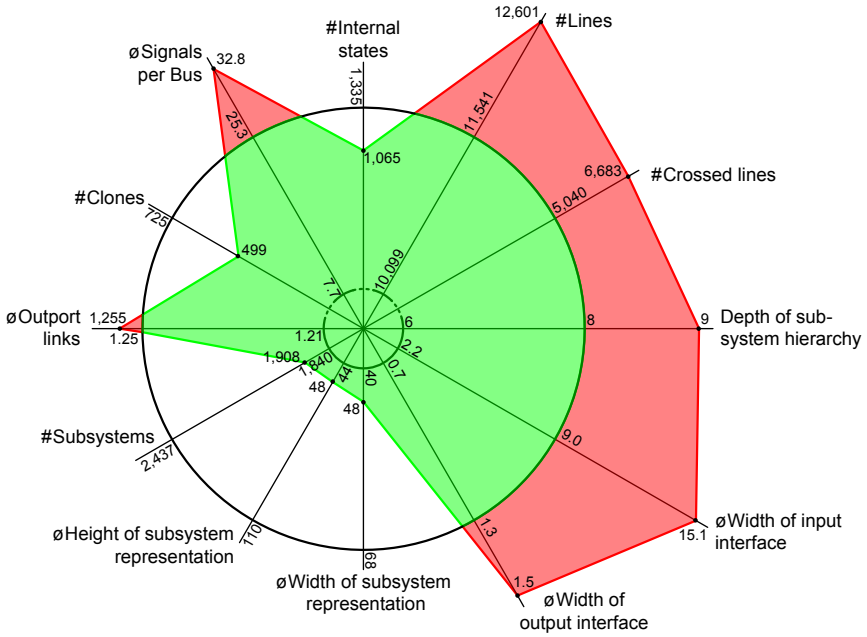

Fig. 2. Measured values and permissible values

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.

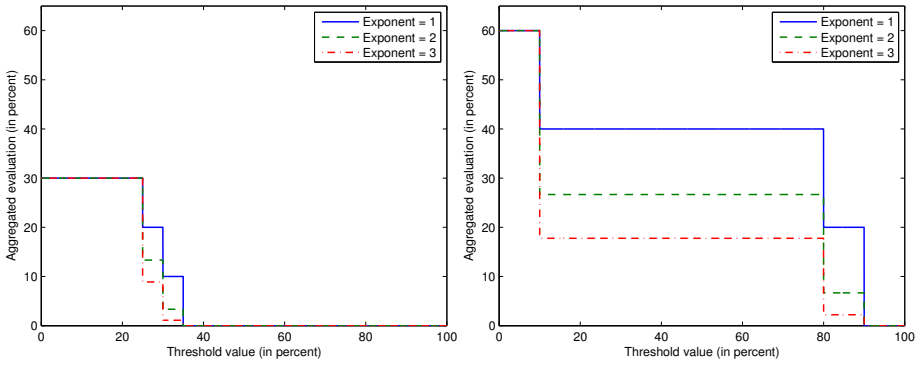

Fig. 3. Aggregation of $e_1 = [25\%, 30\%, 35\%]$ and $e_2 = [10\%, 80\%, 90\%]$

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.

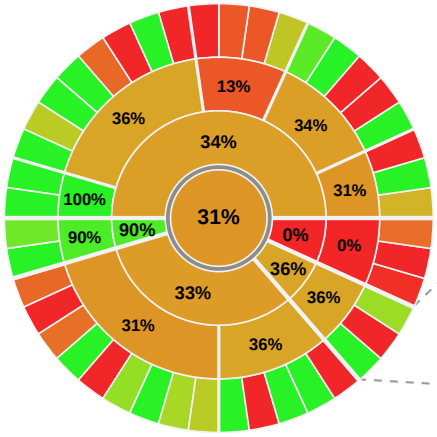

Fig. 4. Complete model quality rating

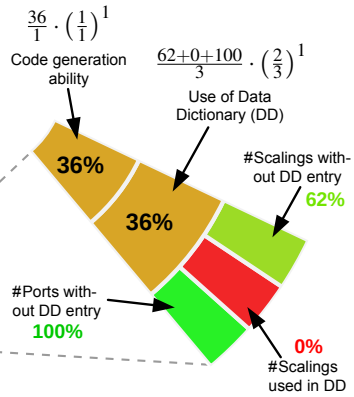

Fig. 5. Rating of a factor

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.

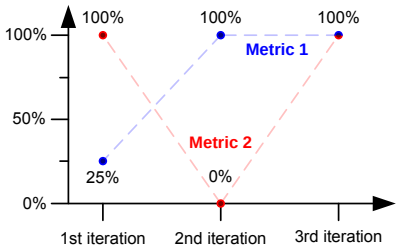

Fig. 6. Relative trend representation

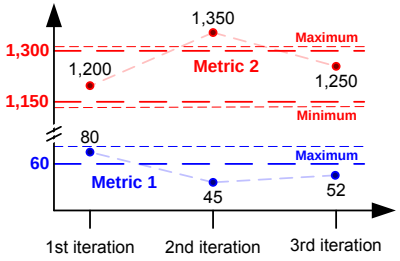

Fig. 7. Absolute trend representation

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.

Supporting the Adaptation of Software Quality Models – An Empirical Investigation

Constanza Lampasona, Michael Kläs

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz-1, 69663 Kaiserslautern, Germany
{constanza.lampasona, michael.klaes}@iese.fraunhofer.de

Abstract. Measuring and evaluating software quality are fundamental challenges for software engineers. Assessing software quality is hard to accomplish in practice not only because existing quality models are not easily applicable, but also because adjusting them to the needs of one's own organization and projects requires intensive effort. In this paper, we present a study on applying a flexible but rigorous adaptation process for quality models. The goal-oriented adaptation process is based on the existence of a quality meta-model that provides a structure for adapted models. In order to obtain first empirical insights, we compare adaptations guided by the tool-supported adaptation process with (ad-hoc) adaptations using a tool that can be used to create and edit quality models. In the study, we investigated the *formal consistency*, *appropriateness*, and *efficiency* of exemplary quality model adaptations. One important study result is that the quality models obtained applying the tool-supported process are considerably more *consistently* and *appropriately* adapted than the ones obtained by following an ad-hoc approach. Further, we could observe that model adaptation is significantly more *efficient* when applying the adaptation process.

Keywords. Empirical study, goal-oriented adaptation, tailoring process.

1 Introduction

The measurement and evaluation of quality are fundamental challenges for software engineers. A lot of models exist that concretize the concept of software quality by defining sub-concepts that are parts of software quality. This refinement typically results in tree-like structures of concepts and sub-concepts. The leaves of the tree are (or are supposed to be) measurable concepts that contribute to software quality.

Although existing models can help to systematically concretize the concept of quality, they also show deficiencies. Some are very specific, which limits the scope of their applicability. Others are abstract, which leads to a large overhead when concretizing them in order to make them applicable. Moreover, concretizing a quality model often requires intensive expert effort. This is especially true since quality models are rarely provided together with a detailed process or method for their adaptation.

Building quality models based on adapting a core model for specific domains and

specific purposes [14] may reduce these problems by achieving a balance between fixed models and models developed from scratch. Relying on this concept of *balanced* models, we propose a method for adapting quality models, including a general adaptation process based on a structure defined by a quality meta-model [12].

Many existing quality models are adaptations of other models, e.g., [2, 5, 1, 4, 10]; however, these customizations are narrowly focused and difficult to transfer to other contexts. Franch and Carvallo [9] present a general process for building an ISO9126 model. Plösch et al. [16] present a method for adapting operational quality models.

We have formerly discussed a potential solution idea for performing goal-oriented, efficient adaptations and obtaining correctly adapted quality models [13]. In this contribution, we present the general process for adapting quality models, which is a core part of our adaptation method, and a study in which we investigated the impact of the implemented adaptation process with regard to (1) the efficiency of adaptation and the (2) formal consistency and (3) appropriateness of the quality models obtained by applying the process.

In the following sections, we provide an overview of the adaptation process and describe the study design. Then, we present and analyze the study's results. Finally, we summarize our current work and sketch planned research directions.

2 Goal-oriented Adaptation of Software Quality Models

Balanced models are a fundamental concept in the German research project Quamoco (www.quamoco.de) in which our work has been conducted. The project aims at developing a software quality standard with easily operationalizable models to cover different technologies for software development. In order to arrive at an operationalized quality model applicable for a concrete environment in a repeatable way, we developed an adaptation process for quality models.

In this chapter, we describe the scope of quality model adaptation and the general steps needed to achieve it, such as identifying an adequate reference model to be customized and the necessary changes to be performed (what has to be modified, when and how).

2.1 Scoping Quality Model Adaptation

Software quality models may exist and be applied at different levels:

Public level: The models at this level are universally available, they may be intended for general use (e.g., ISO9126 [11], Quamoco base model), or for some specific domain (e.g., IEC 61508 [6] for safety in embedded systems, EN 60601-1-4 [7] for medical device embedded systems). This level is comparable to the broad *industry level* described by Fitzgerald [8]. Most of the models at this level are very generic; they are usually not operational and need to be customized. Using and tailoring these models could be useful for showing adherence to some standard.

Organization level: At this level, quality models focus on satisfying the interests of a specific organization. They can focus on the whole organization, a business unit, or a project portfolio. They are typically more specific than quality models at the public level and intended to provide a common basis for project-specific model

tailoring. At this level, public models can be refined for a particular organization and organization models can be further refined for specific parts of the organization.

Project level: At this level, quality models are put into operation; they are applied to specify and assess quality. Here, quality model adaptation should be limited to minor adjustments driven by project-specific quality requirements, without drastically changing the structure of the organization's quality model. This helps to preserve the comparability of quality evaluations across software products. At the project level, organization-wide models are further refined for a particular project.

The reuse of a quality model for adaptation is, in essence, more efficient than creating a new model from scratch for each project. We recommend tailoring models stepwise: for organizations and for projects. This means that the adaptation process proposed here can be used to adapt a public model for an organization and then to refine that organization model for specific project needs.

2.2 Adapting a Quality Model

The main steps for adapting a quality model are: (1) identifying a reference quality model as a basis for the adaptation, (2) sorting out irrelevant content, (3) performing adjustments, and (4) testing the adapted quality model.

2.2.1 Identifying a Reference Quality Model

The first thing to be done is to define the goal of the quality model that should result from the adaptation.

In order to define the goal, the organization/project needs with respect to software quality and context information are used. That is, it is necessary to identify the circumstances under which the quality model will be used. In order to describe the goal in a structured way and not to forget important aspects, we use an adapted GQM goal template, which is typically used to define measurement goals [3]: (1) *Object*, (2) *Purpose*, (3) *Viewpoint*, (4) *Quality Focus*, and (5) *Context*.

To define a goal, these questions must be answered:

1. ***What are the elements that are used to define, measure, or assess the product quality?*** For example, product documentation, source code, requirements, design, build process, test suite, etc. This information is the *object* in the goal.
2. ***For which purpose do I need the quality model?*** Following the classification of application purposes for quality models proposed in [16], the current Quamoco meta-model considers two different *purposes*: specification and evaluation of quality. Specification means that quality is described, but neither quantified nor measured. For the purpose of evaluation, quality is quantified, measured, and compared to defined assessment criteria to check the fulfillment of those criteria.
3. ***From which perspective is quality described or evaluated?*** Are there specific management requirements? What are the agreements with the customer? Must practices established in the organization be considered?
4. ***Which quality attributes of the software product are covered with this model?*** Quality can focus on general properties, such as reliability or maintainability, or specific aspects can be considered, such as globalization, learnability, or training.

5. ***What is the context of the software products to be addressed by the model?*** *Context* may include many different things. Are there things that are mandatory within the organization? Which domain should be covered by the quality model (e.g., railway, medical devices, embedded systems, information systems)? Which methodologies, practices, or technologies should be supported (e.g., component-based software development, agile development, open-source software, custom development, C++, Java, or automatic measurement tools)?

The answers to these questions together describe the goal of the required quality model, which will make it easier to focus on the key elements of the adapted quality model. The documented goal can be used later, for example when the model is inspected, to corroborate that it actually fulfills the stated goal.

Now the goal is used to look for a model and adapt it to the needs of the project or organization. We call this model on which the model adaptation is based, the reference model. Finding the right reference model consists in finding the model whose attributes best fit to the defined goal.

2.2.2 Sorting out Irrelevant Information

Once a reference model is chosen, the actual adaptation can start. First, elements are discarded that are not needed in the final model. Only quality model components in the reference model that are relevant for the new model are taken. In this way, unnecessary components of the quality model are eliminated at the beginning. Such components may be used for specific perspectives, such as the management view on quality, or for quality aspects such as internationalization, which are not of interest in the model. They can also be used for artifacts such as user documentation or design that should not be considered in the model, or for measures that cannot be collected since they are not applicable in the context of the model, e.g., measures for Java code in a model for applications in C. Sometimes, specific elements in the model can be partially reused but need some adjustments. Such elements should stay in the model and be marked for detailed inspection and modification in the next step. The parts selected to remain in the model are the basis for further adjustments.

2.2.3 Further Adjustments

After sorting out irrelevant information, the model obtained might not be consistent or operational anymore. The removal of model components triggers further adaptation tasks. These tasks help to bring the model back to a consistent, operational state. Some adaptation tasks can be automated. Other tasks will require user interaction, as they are based on user decisions.

Accomplishing all adaptation tasks will lead to a consistent model customized to the user's needs. Elements are incrementally deleted, added, or modified in the model until no further adaptation tasks are requested. The extent to which these operations are used depends on the suitability of the reference model. At this point, the quality model has been successfully adapted and can satisfy the defined goal.

New elements can be defined and added to the model or elements from models can be added to the adapted model. That is, individual elements from other models can be taken and reused in the new model.

What should be documented?

Goal of adapted quality model: The goal is a compact manageable description of the quality model. If the appropriateness of the model with respect to the goal is put into question during tailoring, the need to complete the model in order to archive the goal must be documented as well as the fact that the model is not complete.

Deviations from reference model: Operations on elements that are used to be mandatory for the organization and are no longer being considered in the sub-organization or project and the reasons of non-inclusion must be listed. Eventually, an agreement should be signed approving these changes. The manager responsible for the organizational model can take these changes and their justifications as one source for changes when maintaining the organization's quality models.

2.2.4 Testing the Adapted Quality Model

In this step, the adapted quality model needs to be applied to a small sample of test objects, i.e., the adapted model is piloted. This will lead to final acceptance of the model or indicate the need for further adaptations. Performing this step is mandatory when adapting a quality model, but how to perform it in detail is not in the scope of the adaption process.

3 Study Goals, Design, and Performance

The adaptation process as the object of the investigation was implemented as an add-on extending the functionality of an existing quality model editor. In the study, we compared quality model adaptation using only the quality model editor (Editor) and quality model adaptation applying the Quamoco adaptation process implemented by the add-on (Adaptation Assistant).

3.1 Study Goals

In the study, we wanted to investigate the quality of the proposed Quamoco adaptation method; in particular, the following question should be answered: 'Does the implemented adaptation process support its three major goals?'

- *Formal Quality Model Consistency*: Adapted quality models are syntactically correct, i.e., they conform to the structure defined by the quality meta-model.
- *Quality Model Appropriateness*: Adapted quality models are correct and complete with respect to their goals, i.e., they are suitable for use with the Object, Purpose, Viewpoint, Quality Focus and Context.
- *Efficiency of Adaptation*: The adaptation of quality models can be performed in an effort-efficient manner.

3.2 Operationalization

We collected subjective judgments about the achievement of the three major goals associated with the adaptation method by asking the participants closed questions:

Perceived_consistency: Does the participant consider the quality model obtained to be syntactically correct? This is the subjective assessment by the participants of

formal quality model consistency.

Perceived_appropriateness: Does the participant consider the quality model obtained to be appropriate with respect to its goal? (i.e., the model is complete and correct with respect to its goal). This is the subjective assessment by the participants of *quality model appropriateness.*

Perceived_efficiency: Does the participant think that the adaptation can be performed efficiently? This is the subjective assessment by the participants of e*fficiency of adaptation.*

For these variables, we used a 7-point Likert scale: {1: strongly disagree, 2: disagree, 3: somewhat disagree, 4: neither agree nor disagree, 5: somewhat agree, 6: agree, 7: strongly agree}. Additionally, we allowed the answer "I do not know".

Besides evaluating the goals based on the perception of the participants, we also want to evaluate them in a more objective way. Since it is difficult to objectively determine the degrees to which the three major goals are fulfilled directly, we address them indirectly by identifying the minimum set of model elements that need to be adapted (i.e., added, modified, or deleted) in order to obtain a consistently and appropriately adapted quality model. This allows us to define measures for the *completeness* and *correctness* of the performed adaption and use the measurement results as a more objective indicator for the model's *consistency* and *appropriateness*: a more completely and correctly adapted model is more consistent and appropriate.

Completeness: We say that a quality model is completely adapted if all of its elements are adapted that needed to be adapted to obtain a model that is consistent with the structure described by the meta-model and appropriate for addressing its goal. We measure this concept using two base measures: the total number of elements that should be adapted in the quality model based on the provided adaptation scenario and the number of elements in the quality model that were adapted by the study participant:

$$\text{completeness} = \frac{\text{number of adapted elements that should be adapted}}{\text{number of elements that should be adapted}}$$

Correctness: We say that a quality model is correctly adapted if all of its elements that should be adapted are correctly adapted with respect to the quality model goal. This means that we measure the degree of correctness as the percentage of correctly adapted elements with respect to the quality model goal:

$$\text{correctness} = \frac{\text{number of correctly adapted elements}}{\text{number of elements that should be adapted}}$$

Efficiency: We measured efficiency in a more objective way by relating the number of correctly adapted elements and the time needed for the adaptation:

$$\text{efficiency} = \frac{\text{number of correctly adapted elements}}{\text{time required for adaptation}}$$

3.3 Hypotheses

During the study, we tested the following hypotheses:

H_{Sub1} (Perceived consistency): The participants consider the quality models obtained using the Adaptation Assistant syntactically more correct than the quality models obtained using the Editor:

H_{1Sub}: μ(perceived_consistency(AA)) > μ(p_con(E)), i.e., H_0: μ(p_con(AA)) ≤ μ(p_con(E))

H_{Sub2} (Perceived appropriateness): The participants consider the quality models obtained using the Adaptation Assistant more complete and correct with respect to their goals than the quality models obtained using the Editor:

H_{2Sub}: μ(perceived_appropriateness(AA)) > μ(p_app(E)), i.e., H_0: μ(p_app(AA)) ≤ μ(p_app(E))

H_{Sub3} (Perceived efficiency): The participants consider the adaptation to have been more efficiently performed using the Adaptation Assistant than using the Editor:

H_{3Sub}: μ(perceived_efficiency(AA)) > μ(p_eff(E)), i.e., H_0: μ(p_eff(AA)) ≤ μ(p_eff(E))

H_1 (Completeness): The adapted quality models obtained using the Adaptation Assistant (AA) are more completely adapted than the adapted quality models obtained using the Editor (E):

H_1: μ(completeness(AA)) > μ(completeness(E)) , i.e., H_0: μ(comp(AA)) ≤ μ(comp(E))

H_2 (Correctness): The adapted quality models obtained using the Adaptation Assistant are more correctly adapted than the adapted quality models obtained using the Editor:

H_2: μ(correctness(AA)) > μ(correctness(E)) , i.e., H_0: μ(corr(AA)) ≤ μ(corr(E))

H_3 (Efficiency): Quality model adaptation is more efficiently performed when using the Adaptation Assistant than when using the Editor:

H_3: μ(efficiency(AA)) > μ(efficiency(E)), i.e., H_0: μ(eff(AA)) ≤ μ(eff(E))

3.4 Participants and Context

The target population comprises people working as software quality managers in a company or in similar positions where part of their job is to adapt, set up, or maintain software quality models.

We conducted the study during one of the Quamoco project workshops. The participants were partners of the Quamoco consortium experienced in working with quality models. In addition, they had experience with the Quamoco meta-model and the corresponding *Editor*. They had only rudimental knowledge regarding the Quamoco adaptation process and no experience with the *Adaptation Assistant*.

To prepare the participants for the study, we presented the adaptation process together with examples. After that, we introduced the functionality and use of the Adaptation Assistant as well as an example adaptation using the tool.

3.5 Materials and Procedures

For the study, we provided the participants with the following input:

- Two quality model application goals that should be used by the participants to find the most appropriate reference model.
- Two pools of quality models from which the most appropriate reference model should be selected by the participants on paper and in the adaptation tool.
- Two adaptation scenarios including practical adaptation task descriptions.
- Two example quality models that should be adapted by the participants.

During the study, the participants assumed the role of a software quality manager in the scenarios and were asked to perform the following tasks:

- *Finding the most suitable reference model*: The participants had to select a reference model from a pool of quality models for a defined goal of quality model application. Most suitable means that this model meets most of the concepts required by the quality model goal.
- *Producing an adapted quality model*: The participants had to execute the practical adaptation tasks.

These tasks were performed by the participants twice: once with one scenario and the Editor and once with a second scenario and the Adaptation Assistant. We chose this kind of cross-design with two different adaptation scenarios (Table 1) in order to deal with the low number of participants, but get the design-inherent learning effects low.

After each adaptation, the participants provided their feedback by filling out a questionnaire, which asked them to subjectively rate the formal consistency of the quality model, the appropriateness of the obtained quality model, and the efficiency of the adaptation (for the Adaptation Assistant and for the Editor, respectively). After the execution of both scenarios, the entire work-space of each participant was collected and saved for subsequent analysis.

Table 1: Study Design

	Editor	Adaptation Assistant
Group 1*	Adaptation Scenario A	Adaptation Scenario B
Group 2*	Adaptation Scenario B	Adaptation Scenario A

*Both groups had the same number of randomly assigned participants.

4 Study Results and Interpretation

Descriptive Statistics: Table 2 shows the mean, median, and standard deviations (stdev) for the eight adaptations performed during the study, separated into applications of the Editor (baseline) and the Adaptation Assistant:

Table 2: Study Results

	Editor			Adaptation Assistant		
	mean	median	stdev	**mean**	median	stdev
Completeness (in %)	**15.00**	15.78	6.76	**78.55**	76.52	7.46
Correctness (in %)	**8.93**	9.98	3.87	**70.34**	69.17	8.82
Efficiency (elements/minute)	**0.37**	0.41	0.18	**2.90**	2.84	0.54
Perceived Consistency*	**3.50**	3.50	2.38	**6.00**	6.00	0.82
Perceived Appropriateness*	**2.00**	2.00	0.82	**5.75**	6.00	0.50
Perceived Efficiency*	**1.25**	1.00	0.50	**5.75**	5.50	0.96

*measured using a 7-point Likert scale with 1: strongly disagree, 2: disagree, 3: somewhat disagree, 4: neither agree nor disagree, 5: somewhat agree, 6: agree, 7: strongly agree.

Hypotheses: As our sample is not large enough to assume a normal distribution, we applied non-parametric one-sided Wilcoxon signed-rank tests with alpha=0.05.

- H_{Sub1}: perceived_consistency(AA) > p_con(E) was *accepted* with p=0.032,
- H_{Sub2}: perceived_appropriateness(AA) > p_app(E) was *accepted* with p=0.033,
- H_{Sub3}: perceived_efficiency(AA) > p_eff (E) was *accepted* with p=0.034,
- H_1: completeness(AA) > completeness(E) was *accepted* with p=0.034,
- H_2: correctness(AA) > correctness(E) was *accepted* with p=0.034,
- H_3: efficiency(AA) > efficiency(E) was *accepted* with p=0.034.

Threats to validity: The two major threats to the validity of our results are the small sample size and the potential learning effects.

Small sample size: The participants were chosen due to their experience in quality modeling in general and with the quality meta-model as well as with the Editor in particular. Therefore, they are (although a convenience sample) more representative of the target population (i.e., professionals performing quality model adaptations as part of their job) than, for example, students studying computer science. However, these selection criteria led to a small number of participants who executed the scenarios.

Potential learning effects: Although the participants were not requested to follow a

particular process for adapting the first model using the Editor and were confronted with two different adaptation scenarios, they may have learned from the first adaptation, which may have positively influenced their performance during the second adaptation using the Adaptation Assistant.

Further threats are that only a limited time frame was available for the study participants to conduct the adaptation tasks and that the attitude of the participants toward the well-known Editor or the Adaptation Assistant may have influenced the result.

Interpretation: Not only could hypotheses H_1 to H_3 be accepted, but the magnitude of the improvement using the tool-supported Quamoco adaptation process also seems to be high when compared to performing the adaption without explicit adaptation support using only the *Editor*. The effect was perceived by the participants and could be measured by analyzing the adapted models. Therefore, although several threats to the study's validity exist, we conclude that the proposed tool-supported adaptation process can increase the efficiency of adaptation tasks and the quality of their results in terms of consistent and appropriate models. Further, the study results indicate that typical quality model adaptations are difficult to handle adequately without a tool-supported adaptation process. The main reason for these results appears to be that even at first glance manageable adaptation tasks result in many subsequent sub-tasks that must be performed in order to assure the completeness and correctness of the adapted model. In part, these sub-tasks are hard to identify without support due to the complexity of a typical quality model and even harder to remember until they can be resolved due to their large number, especially if there is no process providing guidelines through the adaptation.

5 Summary and Future Work

Adapting models is important to get quality models that fit the specific needs of a concrete environment without building each model from scratch. However, the adaptation of a quality model is a complex and therefore error-prone task. Our study indicates that the quality of the adapted model can be significantly improved when using a well-defined and tool-supported adaptation processes such as the one developed in the Quamoco project. Not only were the consistency and appropriateness of the adapted quality model significantly improved, but so was the efficiency of performing the adaptation tasks.

In a next step, the adaptation method including the adaptation process and the rules for identifying the required adjustment tasks should be transferred to an updated quality model structure and get evaluated in an industrial field study in order to ensure its applicability for quality models and adaptation tasks in practice.

Acknowledgments

Parts of this work have been funded by the BMBF project Quamoco (grant no. 01 IS 08 023 C). We gratefully acknowledge Jens Göddel for his contributions.

References

1. Andersson, T.; Eriksson, I. V. (1996): Modeling the quality needs of an organization's software. In: HICSS '96: Proceedings of the 29th Hawaii International Conference on System Sciences Volume 4: Organizational Systems and Technology. Washington, DC, USA: IEEE Computer Society, p. 139.
2. Andreou, A. S.; Tziakouris, M. (2007): A quality framework for developing and evaluating original software components. In: Inf. Softw. Technol., vol. 49, no. 2, pp. 122–141.
3. Basili, V.; Weiss, D. (1984): A methodology for collecting valid software engineering data. In: IEEE Transactions on Software Engineering, vol. 10(3), pp. 728-738.
4. Bianchi, A.; Caivano, D.; Visaggio, G. (2002): Quality models reuse: experimentation on field. In: COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment. Washington, DC, USA: IEEE Computer Society, pp. 535–540.
5. Calero, C.; Cachero, C.; Córdoba, J.; Moraga, M. (2007): PQM vs. BPQM: studying the tailoring of a general quality model to a specific domain. In: Advances in Conceptual Modeling – Foundations and Applications, pp. 192–201.
6. E DIN IEC 61508-3:2006-07: Functional safety of electrical/electronic/programmable electronic safety-related systems.
7. EN 60601-1-4:1999: Medical electrical equipment - Part 1-4: General requirements for safety - Collateral standard: Programmable electrical medical systems.
8. Fitzgerald, B.; Russo, N. L.; O'Kane, T. (2003): Software development method tailoring at Motorola. In: Commun. ACM, vol. 46, no. 4, pp. 64-70.
9. Franch, X.; Carvallo, J. P. (2003): Using quality models in software package selection. In: IEEE Softw., vol. 20, no. 1, pp. 34–41.
10. Horgan, G.; Khaddaj, S. (2009): Use of an adaptable quality model approach in a production support environment. In: Journal of Systems and Software, vol. 82, no. 4, pp. 730–738.
11. ISO/IEC 9126-1:2001: Software Engineering - Product Quality - Part 1: Quality Model.
12. Kläs, M.; Lampasona, C.; Nunnenmacher, S.; Wagner, S.; Herrmannsdörfer, M.; and Lochmann; K. (2010): How-to evaluate meta-models for software quality? In: Abran, A.; Büren, G.; Dumke, R.R.; Cuadrado-Gallego, J.J.; Münch, J.: Applied Software Measurement - Proceedings of the joined International Conferences on Software Measurement (IWSM/MetriKon/Mensura 2010); pp. 443-462.
13. Kläs, M.; Lampasona, C.; Trendowicz, A.; Münch, J. (2009): Goal-oriented adaptation of software quality models. In: Technische Universität München. Institut für Informatik: Tagungsband 3. Workshop zur Software-Qualitätsmodellierung und -bewertung. SQMB' 10. München, 2010; pp. 4-11.
14. Kläs, M.; Münch, J. (2008): Balancing upfront definition and customization of quality models. In: Technische Universität München. Institut für Informatik: Software-Qualitätsmodellierung und -bewertung. SQMB'08 - Workshop-Band. München, 2008; pp. 26-30.
15. Kläs, Michael ; Heidrich, Jens ; Münch, Jürgen ; Trendowicz, Adam:
    CQML Scheme: A Classification Scheme for Comprehensive Quality Model Landscape. In: EUROMICRO 2009. Proceedings of the 35th EUROMICRO Conference Software Engineering and Advanced Applications. Los Alamitos: IEEE Computer Society, 2009, 243-250: Ill., Lit.
16. Plösch, R.; Gruber, H.; Körner, C.; Pomberger, G.; Schiffer, S. (2010): Adapting quality models for assessments - Concepts and tool support. In: Proceedings of SQMB 2010 Workshop, held in conjunction with SE 2010 conference, February 22nd 2010, Paderborn, Germany, published as Technical Report TUM-I1001 of the TUM.