Goal-oriented Adaptation of Software Quality Models

Michael Kläs, Constanza Lampasona, Adam Trendowicz, Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{michael.klaes,constanza.lampasona,adam.trendowicz,juergen.muench}@iese.fraunhofer.de

Abstract. Objectively measuring and evaluating software quality has become a fundamental task. Many models support software product quality stakeholders in dealing with software quality. In this contribution, we present an approach for adapting software quality models and the challenges that emerge in this regard. We propose an adaptation process based on the use of a core quality model and on the existence of a meta-model that provides an essential structure for the base and for the derived adapted models. We show different solution ideas for obtaining a correct adapted quality model and performing goal-oriented, efficient adaptation.

1 Introduction

Given the increasing pervasiveness of software in our society and its growing complexity, it is essential to produce high software quality. The importance of satisfying customers' needs and keeping the software organization profitable have made the objective measurement and evaluation of software quality a fundamental task.

A myriad of software quality models (QMs) intend to support product quality stakeholders in dealing with software quality. Most of these models can be assigned to one of two strategies for modeling software quality [15], namely *fixed-model approaches* and *define-your-own-model approaches*. The former usually specify a prescriptive set of quality characteristics or metrics, whereas the latter use methods to guide the experts in the derivation of customized QMs. The applicability of fixed models is generally limited to contexts similar to the one in which the model was developed, in contrast to define-your-own approaches, which require intensive expert effort. A third option is represented by the so-called *balanced QMs*, which are based upon the idea of adapting a core model for specific domains and specific purposes [15]. This adaptation should be as much reproducible as possible and should therefore be guided by a detailed process. The use of a base QM and its systematic customization may support cost-effective handling of organizational or project quality needs. Furthermore, the fact that software quality for all products would be relying on the same elementary structure represents another potential advantage.

The concept of balanced models plays a central role in the German research project QuaMoCo, in which the work presented in this paper has been conducted. The project aims at developing a software quality standard with a degree of detail that should allow its direct operational application. Besides, the quality standard should have the

necessary flexibility to cover different technologies for software development. In this contribution, our objective is to present an approach for adapting QMs and the challenges that emerge in this regard.

Existing software quality models are deficient when it comes to their adaptation to the needs of a specific organization or project in a reusable, reproducible manner. Adaptations of QMs are generally based on ISO9126 [14]. Some common modifications are to define attributes [2] or to add quality characteristics for a very specific domain [17, 5].

The literature related to adaptation methods for software product QMs is rather meager. For adapting the ISO9126 QM to the domain of component-based software development, Andreou and Tziakouris [2] take into account the users: component developers, re-users, and end users. The outcome is a quality model especially for original software components. Calero et al. [7] used a general portal quality model for the creation of a special QM for eBanking: BPQM (eBanking Portal Quality Model). To achieve the specialization of the model, a survey was performed among domain experts. Unfortunately, these specific adaptations focus on the resulting adapted QMs and not on a reproducible customization process.

Other authors use *define-your-own-model* tools to refine their specific models. These customizations have a narrow focus and are difficult to transfer to other contexts. Andersson and Eriksson [1], for example, present a process for the construction of a QM founded on a basic QM with existing metrics (SOLE quality model). They illustrate how to customize the model to the specific needs of an organization, including how to identify quality factors and mapping them to metrics. The SOLE quality model [10] has the factor-criteria-metric [16] structure. Bianchi et al. [6] used GQM to refine a specific model. They center their research on QM reuse, i.e., which changes can be requested when a quality model is reused, how to verify that despite the changes made in the reused quality model, it remains suitable for its goals, and what the side effects caused by changing the metrics are on the quality model. Horgan and Khaddaj [13] propose an approach for model refinement based on expert knowledge.

Franch and Carvallo [11] present a very general process to build an ISO9126 QM. It should be kept in mind that we are explicitly referring to software product quality and not to process quality. Nevertheless, for the refinement of our tailoring idea, we will consider concepts related to software process adaptation and study their transferability to the customization of software product quality models.

2 Proposed Adaptation Approach

2.1 Adaptation Process Scope

In order to locate the adaptation of a QM within an organization's structure and processes, we use the Quality Improvement Paradigm (QIP) [4]. QIP defines an abstract six-step process for introducing and continuously improving a technology within an organization, where analogical cycles are applied at the organizational level and at the project level. We recommend embedding quality modeling and model application into

these cycles (Fig. 1). At the organizational level, the performance of a QM is the subject of continuous improvement. At the project level, the QM is used to evaluate and improve software product quality.

Major QM adaptations should take place in step 3 (Choose Processes) at higher organizational levels such as a whole organization, a business unit, a domain, or a projects portfolio. At the project level, QM adaptation takes place in the analogical step (4.3) and should be limited to minor adjustments, e.g., driven by project-specific quality requirements, without changing the QM structure, in order to preserve conformance of quality evaluations across software products created at the project level. Therefore, an adaption process has to support three logical activities: reducing, extending, and adjusting a QM.[1]

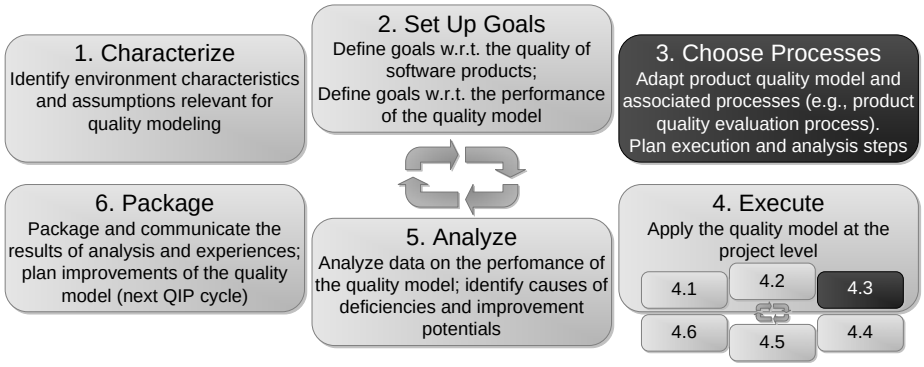

Fig. 1. Scope of the QM adaptation process

2.1 Requirements

Based on the requirements for the definition and application of QMs stated by practitioners and scientists within the QuaMoCo project, we condensed three major requirements with respect to the adaptation process:

- *(R1) Correctness* – An adapted QM must be syntactically correct in that it remains conformant to the underlying meta-model (MM) and to the defined consistency rules.

[1] QIP refines the first step of the PDCA approach [9] (Plan) into three more detailed ones, with the remaining three PDCA steps being used very similarly by the QIP [11]. For readers who may be more familiar with the PDCA approach, this means that the quality adaptation process can be thought of as a part of the first step within PDCA. Analogically, the DMAIC cycle [18] could be mapped to the concepts of QIP, and probably, step 3 of QIP would correspond to some activity in the Define step of DMAIC.

- *(R2) Goal Orientation* – The adaption of a base model should be driven by organizational needs and capabilities. In particular, organization-specific and project-specific software quality objectives should be considered.
- *(R3) Efficiency* – This is concerned with the overhead (e.g., personnel, time, and budget) needed for adapting a QM. Acceptable overhead would differ depending on the organizational level (e.g., more overhead will be allowed for adapting a QM at the level of the whole organization, where such adaptation has a larger scope and is performed relatively rarely).

One major challenge of defining a QM adaptation process is to make it independent of a particular model, i.e., to define a set of adaptation rules that will be universally applicable to any model conformant with the QuaMoCo MM.

2.2 Solution Idea for R1 (Correctness)

Assuring syntactical correctness requires an MM describing the structure of the QM and some consistency rules that should be fulfilled by the QM in order to be compliant with the MM. We show the adaptation process on the QuaMoCo MM (Fig. 2) [8]. The two fundamental constructs in this MM are the *Quality Aspect* tree, which describes and refines the quality characteristic of interest (e.g., maintainability is broken down into the sub-aspects analyzability, stability, changeability, and testability) and the *Factors*, which capture the product-related factors with the major influence on the Quality Aspects defined (e.g., complexity of source code). A Factor consists of an *Entity Type* (e.g., source code) and a *Property* that characterizes it (e.g., complexity). Both provide important information for defining appropriate Measures, with a *Measure* referring to rules for determining the actual value of a Factor's occurrence. Based on the Measures, an *Impact Evaluation* can be specified to determine the concrete impact of the Factor on the Quality Aspects (i.e., a rule mapping the measurement results to a specific value on the evaluation scale). The general tendency whether a factor improves or inhibits a specific Quality Aspect is defined and justified by the corresponding *Impact* (e.g., high complexity inhibits the analyzability of the product). In order to get an overall statement about the quality of interest, the evaluation results of different Impacts and of different subordinate Quality Aspects have to be combined according to rules defined by *Quality Aspect Evaluations* (e.g., analyzability and stability evaluations are combined into an overall maintainability statement). One approach for assuring the consistency of an adapted model would be to adapt the QM first, and then check its consistency based on the MM and consistency rules. Another approach for assuring consistency *during* the adaptation of a QM would be to define a limited set of basic operations that allow transforming a QM from one consistent state into a new consistent state. We propose a compromise between these two options: We allow intermediate inconsistencies and defining rules for a set of basic transformations to highlight inconsistencies and to explain what has to be done to return the QM back to a consistent state. Such basic transformations can be *DELete*, *ADD*, or *MODify* a specific model construct (see Table 1). For instance, if we want to add a new Measure for an existing Factor (i.e., ADD(Measure)), we have to check all Impact Evaluations defined for the Factor and include the Measure in the Impact Evaluation description.

Table 1. Exemplary consistency rules for basic transformation operations

Construct	DEL([Construct])	ADD([Construct])	MOD([Construct])
Measure	*Modify* descriptions of all impact evaluations that use the measure, in order to remove the deleted measure from the impact evaluation.	For each factor to which the measure added belongs, *modify* related impact evaluation descriptions in order to include the new measure.	*Check* descriptions of all impact evaluations that use the measure, in order to align them with the modification of the measure (if required).
…	…	…	…

2.3 Solution Idea for R2 (Goal Orientation)

We suggest describing the objective of the adapted QM using the GQM goal template [3]. The GQM goal template is a means for specifying a goal in a structured way. Our aim is not to derive a measurement plan using the GQM approach, but to use the GQM template to formulate the quality modeling goal with respect to which adaptation should be performed. The template describes the entities to be considered (*Object*), from which perspective we consider them (*Viewpoint*), the quality aspects of interest (*Quality Focus*), the adapted QM intention (*Purpose*), and the environmental characteristics influencing the QM (*Context*) (for an example goal, see Fig. 2).

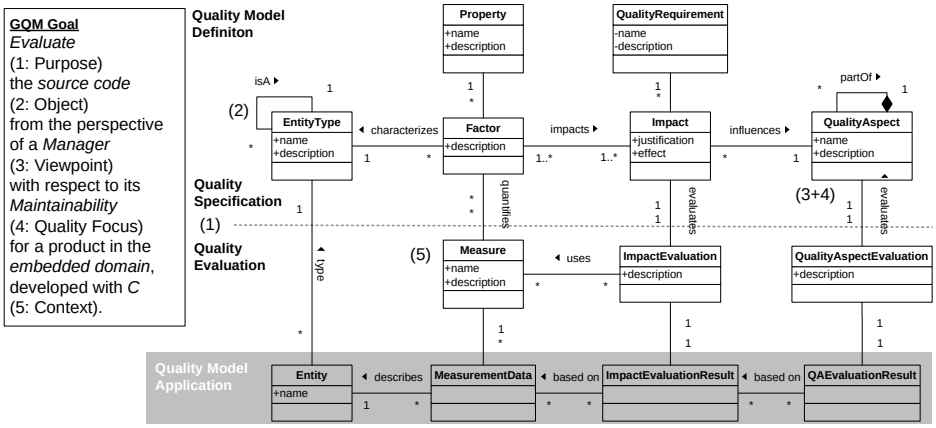

Fig. 2. Quality MM and area of goal-oriented QM adaptation

A goal-oriented approach allows us to simplify the adaptation process such that each basic transformation is assigned elemental rules specifying which QM elements need to be adapted and how. For instance, the following goal-oriented rules can be defined for reducing the QM, i.e., removing all its parts that are not relevant for achieving the particular goal: (1) For the purpose of quality *evaluation*, all constructs defined by the MM are required. For the purpose of quality *specification*, we can remove irrelevant elements by applying the rule: DEL(Construct ∈ {Measure, Impact Evaluation, QualityAspectEvaluation}). (2) All Factors in which the entity type does not correspond to the Object specified in the QM's goal can be removed from the QM by applying the rule: DEL(EntityTypes ∉ instanceOf(Object)). For example, if we are interested in *source code*, this operation removes requirements- and design-related Factors from

the QM. (3) The Viewpoint identifies relevant Impacts by scoping specific Quality Aspects. Impacts that are irrelevant regarding a particular Viewpoint can be removed, together with associated Impact Evaluations, by applying the rule: DEL(QualityAspects $\notin$ partOf(Viewpoint)). (4) If we are only interested in a specific Quality Aspect (e.g., maintainability, but not reliability), this would further reduce the relevant scope of the adapted QM: DEL(QualityAspects $\notin$ part of (QualityAspect)). (5) Finally, we have to remove all Measures that cannot be collected in our context, e.g., for *C* source code this could be object-oriented metrics: DEL(Measure m with m.applicableFor.language $\notin$ context.language).

2.4 Solution Idea for R3 (Efficiency)

To achieve efficient QM adaption, existing (MM-conformant) QMs should be reused as an adaptation base instead of building a QM from scratch. Further, a base model should cover diverse concrete content, because QM reduction can be more efficiently automated than QM extension. However, including only universally valid content in such a model would lead to a nearly empty model without reuse potential. Thus, the adaptation approach should allow identifying potential reuse candidates.

2.5 Connection of Solution Ideas

In an initial tailoring step, the quality model is reduced to the elements needed for the specific quality modeling goal (R2). We increase reduction efficiency by focusing on the key elements of the GQM goal and by automating the respective basic operations (R3). In further adaption steps, the model can be modified and extended with basic operations, while consistency is assured by associated consistency rules (R1).

3 Summary and Future Work

This paper explains the need for an adaption process for QMs, presents fundamental requirements identified, and sketches an approach that addresses these requirements. The consistency of the adapted QM is covered by the definition of basic operations and corresponding consistency rules; further, the approach explains how to integrate the relevant goals into the adaptation process and addresses the efficiency of adaptation through automation and reuse. The next steps will be the refinement of the approach, i.e., the explicit description of a process into which the ideas presented here will be embedded. An empirical evaluation in an industrial environment is also planned, as is a tool for supporting the customization process. A special challenge is seen in the appropriate use of context information.

4 Acknowledgements

Parts of this work have been funded by the BMBF project QuaMoCo (grant 01 IS 08 023 C). We gratefully acknowledge the contributions and input of Reinhold Plösch, Dominik Kirchler, and Jens Heidrich.

5 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. R. (1992): Software Modeling and Measurement. The Goal/Question/Metric Paradigm. University of Maryland - Dept. of Computer Science: (Computer Science Technical Report Series). - NR CS-TR-2956. - NR UMIACS-TR-92-96.
4. Basili, V. R.; Caldiera, G.; Rombach, H. D. (2002): Experience Factory. In: Marciniak, John J. (Ed.): Encyclopedia of Software Engineering. 2nd Ed. New York: John Wiley & Sons, Vol. 1, pp. 511–519.
5. Behkamal, B.; Kahani, M.; Akbari, M. K. (2009): Customizing ISO 9126 quality model for evaluation of B2B applications. In: Inf. Softw. Technol., vol. 51, no. 3, pp. 599–609.
6. 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.
7. 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.
8. Deissenboeck, F. H. (2009): Quamoco Report: Quality meta model-WP1.3 v1.0 2009-08-17.
9. Deming, W. E. (1986): Out of the crisis. Massachusetts Institute of Technology, Cambridge, Mass.
10. Eriksson, I.; Törn, A. (1991): A model for IS quality. In: Softw. Eng. J., vol. 6, no. 4, pp. 152-158.
11. Franch, X.; Carvallo, J. P. (2003): Using quality models in software package selection. In: IEEE Softw., vol. 20, no. 1, pp. 34–41.
12. Hamann, D. (2006): Towards an integrated approach for software process improvement. Combining software process assessment and software process modeling. Techn. Univ., Diss.--Kaiserslautern, 2005. Stuttgart: Fraunhofer-IRB-Verl. (PhD Theses in Experimental Software Engineering, 19).
13. 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.
14. ISO/IEC 9126-1:2001: Software Engineering - Product Quality - Part 1: Quality Model.
15. Klaes, M.; Muench, J. (2008): Balancing upfront definition and customization of quality models. In: Software-Qualitaetsmodellierung und -bewertung SQMB'08, pp. 26–30.

16. McCall, J. A.; Richards, P. K.; Walters, G. F. (1977): Factors in Software Quality. Concept and Definitions of Software Quality: Final Technical Report Springfield: National Technical Information Service (NTIS), Reportnr. RADC-TR-77-369 (I, II and III).
17. Sharma, A.; Kumar, R.; Grover, P. S. (2008): Estimation of quality for software components: an empirical approach. In: SIGSOFT Softw. Eng. Notes, vol. 33, no. 6, pp. 1–10.
18. Tayntor, C. B. (2002): Six Sigma Software Development. Boca Raton: Auerbach Publications, 2002.