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

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]

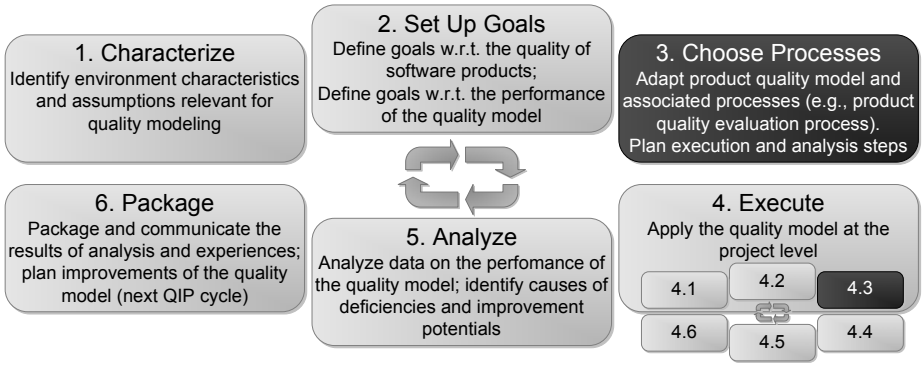

Fig. 1. Scope of the QM adaptation process

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.

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

- *(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).

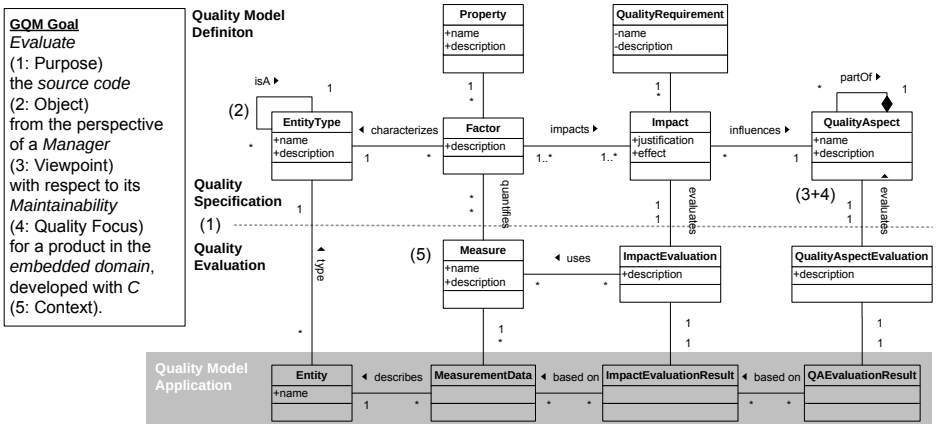

Fig. 2. Quality MM and area of goal-oriented QM adaptation

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 ∉ 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 ∉ 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 ∉ 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.

Adapting Quality Models for Assessments – Concepts and Tool Support

Reinhold Plösch[1], Harald Gruber[1], Gustav Pomberger[1], Christian Körner[2]

[1] Johannes Kepler University Linz, Altenberger Straße 69,4040 Linz, Austria
{reinhold.ploesch, harald.gruber, gustav.pomberger}@jku.at

[2] Siemens AG, Corporate Technology – SE 1, Otto-Hahn-Ring 6, 81739 Munich, Germany
christian.koerner@siemens.com

Abstract. Operational quality models, i.e., quality models that do not only structure and define quality by means of quality aspects but also contain a larger number of measures, facilitate quality measurement as the laborious task of defining measures can be omitted. An operational quality model often contains a large number of quality aspects and measures; therefore methodological support as well as tool support is necessary to adapt such a quality model to project-specific needs. Based on a general quality adaption framework we have developed a method for tailoring quality models. This method is supported by an accompanying tool. The application of the method and the tool in more than 40 projects proved the practicability of the approach. Nevertheless additional methodological support for deriving quality aspects or quality requirements from (business) goals would be desirable.

Keywords: ISO 9126, Adapting quality models, EMISQ method, SPQR tool

1 Introduction

It's the ambitious goal of software engineers to develop good software. What means “good”? Assessing a product, i.e., determining whether the product (software as well as any other consumer product) is good, has something to do with quality. The term quality may be considered from different viewpoints, leading to different interpretations and definitions. ISO 9126 [8], which follows the product-based and manufacturing-based approaches (as introduced in [4]), defines the term software quality as "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs".

This definition gives us an idea about software quality but is too vague for any practical application. To allow measuring software quality, a systematic approach has to be used to derive measures from general properties of software. Quality models provide this in a systematic manner. There are a number of different quality models known in literature. One example of a general model is the so called factor-criteria-metrics-model (FCM-model) [1]. According to this quality model, the quality of software is described by identifying various quality attributes, often called factors or quality aspects. In order to make each factor measurable, it is necessary to refine factors into quality criteria. Factors represent a more user-oriented view, while quality

criteria reflect a more software-oriented view. The refinement process takes place until quality indicators, i.e., measurable and assessable metrics, can be found for each quality criteria. There exist a number of specific quality models based on this general FCM-approach, e.g., the model defined by ISO 9126 [8], the FURPS model [5], the model defined by McCall [10], the quality model by Barry Boehm [2] or the SATC model [6].

The benefits of these approaches are well structured quality aspect hierarchies that facilitate reasoning about quality and ease finding measures for specific quality aspects. The model proposed by ISO 9126 defines a widely used quality model. Nevertheless, on the level of metrics (in terms of the FCM-model) ISO 9126 provides only some examples for metrics. These examples give good hints how to measure a quality aspect, but the set of measures provided is far from being comprehensive.

One important aspect of EMISQ ("Evaluation Method for Internal Software Quality") [11] was (and still is) the development of operational quality models. Operational in this context means providing a comprehensive set of measures for each quality attribute, with special attention on measures that can be provided automatically by static code analysis tools like PMD [15] or PC-Lint [14]. The current quality model provided by EMISQ contains more than 3,000 measures of 15 different tools. These measures are assigned to two quality models – one that is similar in structure to the ISO 9126 model and a second one that has its emphasis on technical topics and problems (e.g. memory issues, threading issues). This large number of measures makes it necessary to think about an adaptation process and a method as not all 3,000 measures can be applied for each project. Approaches like GQM [16, 17, 18] or SQUID [9] focus on building entire quality models from scratch and cannot be directly used for adaptation tasks, though the result of these approaches (especially the identified quality aspects) could of course be used as input for the tailoring process. Franch and Carvallo [3] propose a six step approach for building (adapting) a quality model based on the ISO 9126 quality model. Both, on the level of quality attributes and on the level of measures they allow the following principal operations:

- *Delete-Operations*: Remove quality attributes (or later measures) that are not suitable for the domain or the specific project.
- *Add-Operations:* Add quality attributes (or later measures) that were not included in the original (ISO 9126) quality model but that make sense in the context of the domain or project.
- *Modify-Operations:* Adapting of thresholds for measures.

Fig. 1 depicts the principal process for adapting the quality model. This general adaptation process needs to be refined in order to provide the necessary support for the adaptation. In our EMISQ method we provide more specific methodological support for this general adaption process (this is described in chapter 2). Chapter 3 describes how the tailoring approach provided by EMISQ is supported by our tool SPQR (“Software Project Quality Reporter”).

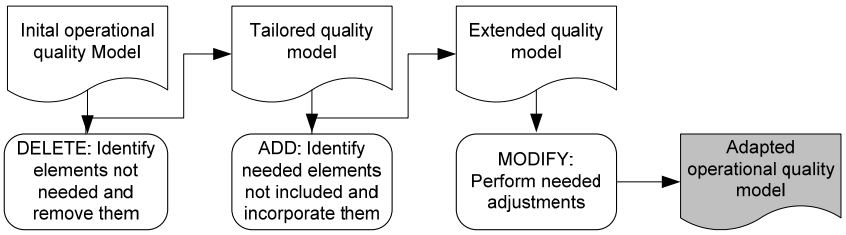

Fig. 1. General adaptation process

2 Tailoring in the EMISQ Assessment Method

Besides the quality model aspect, EMISQ is a methodology for systematically assessing the internal software quality. The assessment model is based on the ISO 14598 [7] standard but extends it significantly. The major differences are:

- Instead of specifying a quality model and finding measures for each project we have developed a pre-defined quality model with measures integrated from various static code analysis tools. This quality model may be tailored for a particular project. The step "Establish rating levels for metrics" of the ISO standard can be excluded completely as its activities are already considered in our quality model.
- Contrary to the ISO standard the evaluation of the quality attributes does not occur automatically after the measures are collected but needs an expert. An aggregation model describes how to sum up quality ratings from the sub-attribute level to the top level.
- The method is substantially supported by a knowledge base, i.e., by a set of structured evaluation guidelines, that guides the evaluation team and provides extra material like checklists, tool configuration files, presentation material, and document templates.
- Furthermore, the evaluation team can use a tool called SPQR (Software Project Quality Reporter) that is aligned with the EMISQ-method. This would of course also be possible when applying ISO 14598, but currently SPQR is tightly focused on the EMISQ-method.

The EMISQ evaluation model consists of eight main activities that are each divided up into 5 to 10 sub-activities. Fig. 2 depicts the main activities.

The first activity – “Establish Purpose of Evaluation” – is a major preparing activity for the adaptation process. The intention of this step is to identify and define the goals of the evaluation project, e.g., improving the maintainability of a software product or identifying stability and efficiency problems. It has to be found out whether these goals can be achieved by following the EMISQ method or whether additional measures (e.g. dynamic analysis of programs) have to be planned. The result of this activity is a signed project agreement and a set of the goals that have to be considered by the evaluation project.

The adaptation process is spread over three main activities (see Fig. 2). According to the defined goals, the types of products to be analyzed are selected (“Identify Types

of Products"). The purpose of this activity is to identify those software artifacts that are objects of the assessment, e.g. products, packages or subsystems. During this step all technical issues for the selected software artifacts that are important for the evaluation are documented.

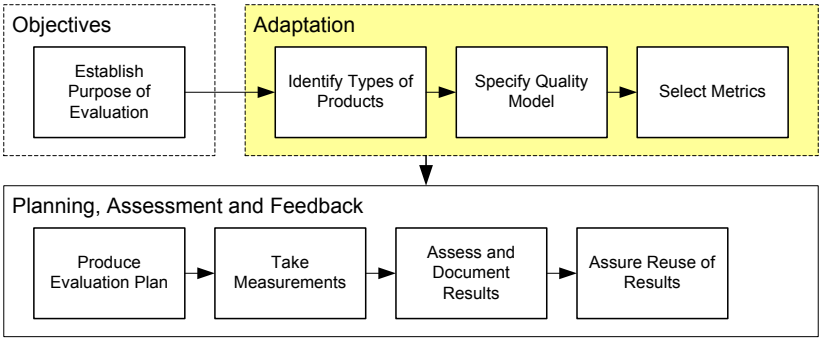

Fig. 2. EMISQ Process Overview. Activities with tailoring tasks are highlighted.

The core adaptation activities take place in the activities "Specify Quality Model" and "Select Metrics".

The adaptation process in "Specify Quality Model" starts by trying to map the identified goals to quality aspects. In EMISQ we support two different quality aspect hierarchies – one following ISO 9126 (with some enhancements) and, as a second choice, one that is organized by technical topics (e.g. Memory topics, Timing topics, Object-oriented programming topics). The respective quality aspects are identified by analyzing the goals and trying to link them with aspects that are suitable to support goal achievement. The formulation of goals (e.g. "The code should be highly portable") often gives hints for the selection of quality aspects, but obviously more support would be desirable here. Currently we are working on a better method support for this mapping.

Depending on the types of goals either management oriented quality aspects in the manner of ISO 9126 or technical topics will be selected. The EMISQ approach starts with fully developed operational quality aspect hierarchies and removes those quality aspects that cannot be justified by the identified goals. Our current quality model contains more than 3,000 measures from various static code analysis tools, which makes flexible support for the selection of quality aspects and measures absolutely necessary. Thus, according to the general adaption steps presented in chapter 1, we perform *delete*-operations on the level of quality aspects (measures are not yet considered). Sometimes a goal might impose a link on a general quality aspect like "Maintainability". In such a case it has to be cross-checked, whether all sub quality aspects (e.g., "Readability", Changeability") are to be included or whether the goal was formulated too imprecise. Fig. 3 depicts the adaptation process.

As shown in Fig. 3 all measures associated with quality aspects are part of the tailored quality model at this stage of the adaptation process. Typically we do not apply *add*-operations (see chapter 1) on the level of quality aspects as we start with a fully developed quality model.

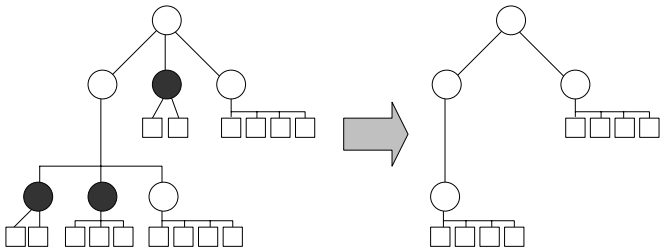

Fig. 3. Tailoring (deleting) of quality aspects. Circles denote quality aspects, squares denote measures. Quality aspects marked for deletion are indicated by dark grey color.

In the “Select metrics” activity the measures are adapted. In a first step, typically *delete*-operations (see chapter 1) are applied – with different selection criteria for deletion:

- *Programming Language*: Typically only measures of one programming language are used in a project. Nevertheless, depending on the complexity of a project, it could make sense to integrate measures of more than one programming language in the quality model, if different parts of the project are realized using different programming languages. In that case, the quality goals have to be invariant for these different parts.
- *Static Code Analysis Tool:* The selection of static code analysis tools is mainly driven by experience with tools in an organization and by software license conditions. Therefore those static code analysis tools are excluded that do not fit into the organization. Typically a static code analysis tool is focused on one programming language (e.g. PMD [15] provides support for the programming language Java, only). Nevertheless, there are some tools available (e.g. Sissy [16]) that provide multi programming language support. It therefore makes sense to decouple the selection of the programming language from the selection of the static code analysis tools.
- *Importance:* As a part of the quality model each measure has an associated importance attribute. The importance of a metric is defined for the relation between the measure and the quality aspect, i.e., if a measure is associated with more than one quality aspect, the importance might be different for each quality aspect. This importance attribute allows the selection of metrics in the range “very high” to “very low”. Usually, when starting a quality enhancement program based on static code analysis, only measures with “very high” importance are selected in order to keep the effort for quality enhancements low. Later in the project, additional metrics with lower importance might be added.
- *Trustworthiness:* Each measure in the quality model is attributed with a trustworthiness level (in percent). This number is derived by means of a special analysis technique, where we have manually inspected the results of static code analysis tools in order to find out so called false-positives, i.e., measure values that are misinterpreted as false while being correct. Details on the approach can be found in [13]. Depending on the importance of a quality

aspect, one might select for instance metrics with a lower trustworthiness, too, if it is an important quality aspect, and on the other hand will concentrate on metrics with high trustworthiness only, if the quality aspect is of lesser importance.

- *Key Metrics:* Key metrics have been identified by a group of experts and denote metrics that are considered to be important regardless of the type of software product. This attribute therefore reflects the experience of experts. Typically, key metrics are important for at least one quality aspect and have at least an average trustworthiness level.

These criteria can be combined flexibly and therefore give the opportunity of a fine-grained selection of metrics for the project-specific quality model. Additionally, individual metrics may be separately deleted from the quality model.

We also provide *add*-operations for measures, i.e. depending on the goals it might be useful to add measures. Adding measures takes place when measures of tools should be used that are not yet part of our operational quality model. Additionally, some tools allow the specification of additional metrics, e.g., by using a special query engine (PMD [15] is an example of a tool that allows this). Finally, metrics that cannot be retrieved automatically but only by means of code inspections or code walkthroughs can be added.

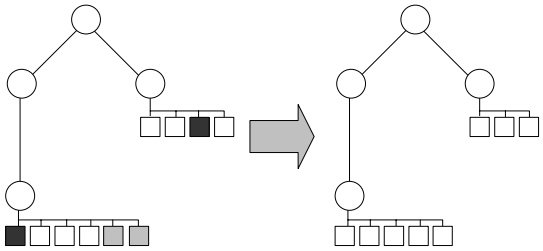

Fig. 4. Adaptation of metrics – deleting and adding metrics. Starting with the quality model tailored by quality aspects (see Fig. 3) we delete metrics (marked dark grey) and add metrics (marked light grey).

In a last step, *modify*-operations are applied onto measures. This basically means that thresholds for metrics are adjusted according to the goals of the project – quality aspects that are considered to be very important will have stricter thresholds for the associated metrics than quality aspects that are of less importance.

Typically we apply this adapted quality model on the project in order to fine-tune the selection of measures. This is essential as there are always project situations were some measures – that are reasonable in principal – cannot be used for the project, as they are in conflict with established and documented programming practices of the project. So, we typically apply additional *delete*-operations on metrics after a first measurement task before finishing the adaptation process.

3 Tool Support for the Tailoring Process

Tool Support for EMISQ is provided by the Software Project Quality Reporter (SPQR) [12] which is implemented as set of Eclipse plug-ins. SPQR is capable of

importing data from various static code analysis tools for associating this data with the quality model. Flexible filtering and browsing facilities allow detailed inspection of the data by quality experts and provide integrated access to the source code under investigation. Furthermore the tool provides support for documenting quality measures and for documenting the ratings of the quality experts. Finally the tool allows exporting a preliminary code quality report as Microsoft Word document. This is the core functionality needed by quality experts. In addition, plug-ins are available to maintain the central quality model and to tailor this quality model to the specific needs of each evaluation project. In this chapter we present the tailoring capabilities in more detail.

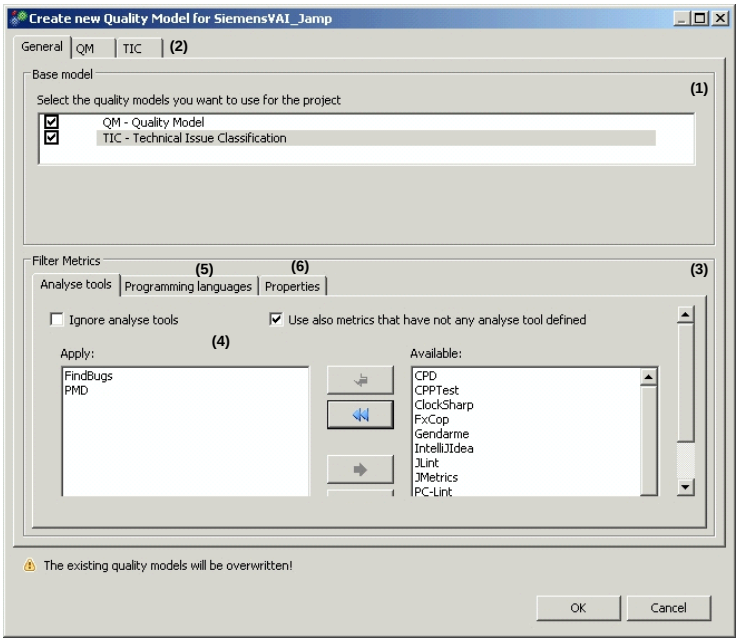

Fig. 5. Adapting quality models with SPQR – Overview

The tailoring process starts with a fully operational quality model. The main user interface for adapting this quality model is depicted in Fig. 5. The following numbers refer to the numbers in the image above.

(1) By default the ISO 9126 quality hierarchy (QM-Quality Model) as well as the defect oriented classification (Technical Issue Classification) are available. If the defined goals for a project clearly focus on one view only, the unnecessary view can be excluded quickly (by the particular checkbox) .

(2) For each quality aspect hierarchy those aspects that are not needed in a project can be excluded (see Fig. 6, where the aspects Portability and Security are deleted from the quality model).

(3) The deletion of metrics from the quality model can be done by different criteria – either by means of analysis tools, programming languages or additional properties.

(4) For the current project the tools FindBugs and PMD are used. The other static code analysis tools (and therefore the measures associated with them) are excluded from the quality model.
(5) Deleting metrics for specific programming languages is done in a similar way as deleting tools (see (4)).
(6) The deletion of metrics can also be done via the properties trustworthiness and importance as well as by considering key metrics as described in chapter 3 (see Fig. 7).

Pressing the ok-Button (see Fig. 5) generates a tailored quality model. The quality model editor (there is no figure for this in this paper) can be used to manually add new metrics and to modify the thresholds of metrics and therefore fully support *add* and *modify* operations as presented in chapters 1 and 2 on the level of measures.

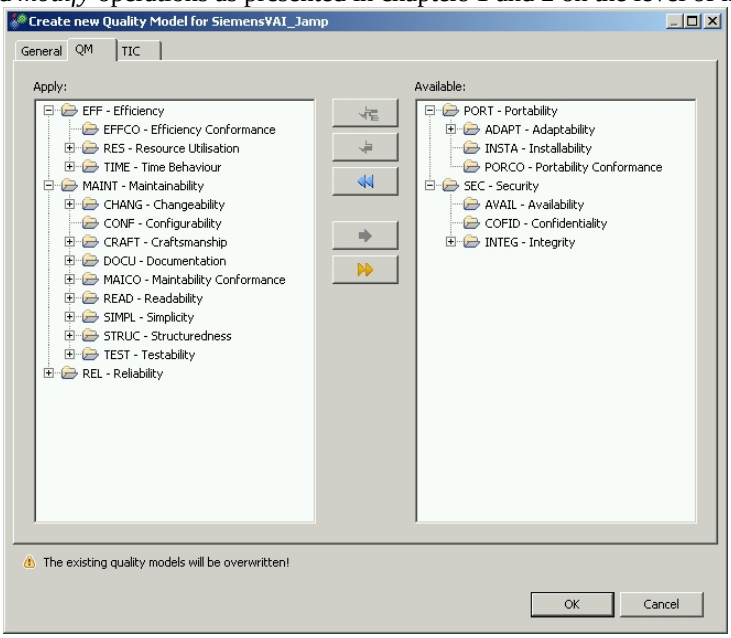

Fig. 6. Deleting quality aspects

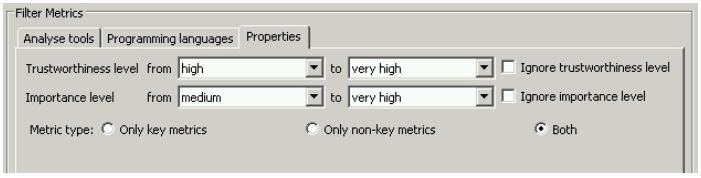

Fig. 7. Deleting measures by properties

At the end of the adaptation process an adapted quality model is available. SPQR generates the appropriate tool configuration data – depending on the measures and tools included in this adapted quality model.

4 Experience and Further Works

The tailoring process described in this paper was applied to more than 40 evaluation projects worldwide with success. The principal idea of starting with an operational quality model proved to be good. Typically, gathering of goals and mapping these goals to quality aspects is done by means of one or two workshops. The workshop is used to identify business and quality goals and to map them to quality attributes, i.e., to gain a common understanding of the relation between goals and quality attributes. The fine-tuning of the quality model on the level of metrics is typically driven by the importance of a quality attribute and by the availability of tool support (or the license costs for them). A comparable quality model (i.e., with comparable accuracy) built from scratch would require considerable more effort (in the projects we have in mind this would mean another addition 2 person weeks at least). Additionally, having an operational model facilitates the communication as examples for measures of a quality aspect can be shown – this often clarifies problems or ambiguities in the definition of a quality aspect and therefore leads to more accurate quality models. In all projects the tailoring of the quality model takes place by means of analysis tools, as there is often experience available with specific tools, or organizations want to start with open source tools, only. Trustworthiness levels or importance of metrics are hardly used in projects, while the selection of key metrics is typically done when starting with quality management.

On the method level additional support is needed for deriving quality aspects from (business) goals. Currently, there is hardly any method support available which means that only highly experienced quality experts can adapt the quality model.

References

1. Balzert H.: Lehrbuch der Software-Technik – Software Management; Software-Qualitätssicherung, Unternehmensmodelierung, Spektrum Akademischer Verlag, 1998
2. Boehm B., Brown J.R., Kaspar H., Lipow M., MacLeod G.J., Merrit M.J., "Characteristics of Software Quality", 1978
3. Franch X., Carvallo J.P..: Using Quality Models in Software Package Selection, IEEE Software, Vol. 20, No 1, IEEE Computer Society Press, 2003
4. Garvin D.A.: "What does Product Quality Really Mean?, in: Sloan Management Review, 1984, pp 25-43
5. Grady R.B., Caswell D.L.: "Software Metrics: Establishing a Company-Wide Program", Prentice Hall, 1987
6. Hyatt L., Rosenberg L.: "A Software Quality Model and Metrics for Identifying project Risks and Assessing Software Quality", Proceedings of 8th Annual Software Technology Conference, Utah, April 1996
7. ISO/IEC 14598: Information Technology – Software Product Evaluation; ISO/IEC, 1999
8. ISO/IEC 9126-1:2001: Software engineering - Product quality - Part 1: Quality model (2001)
9. Kitchenham B., Linkman S., Pasquini A., Nanni V.: The SQUID approach to defining a quality model., Software Quality Journal, Vol (6), No 3, September 1997, Springer Netherlands, 1997
10. McCall J.A., Richards P.K., Walters G.F.: "Factors in Software Quality", Rome Air Development Center, 1977
11. Plösch R., Gruber H., Hentschel A., Körner Ch., Pomberger G., Schiffer S., Saft M., Storck S.: The EMISQ Method - Expert Based Evaluation of Internal Software Quality,

Proceedings of 3rd IEEE Systems and Software Week, March 3-8, 2007, Baltimore, USA, IEEE Computer Society Press, 2007
12. Plösch R., Gruber H., Pomberger G., Saft M., Schiffer S.: Tool Support for Expert-Centred Code Assessments, Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation (ICST 2008), April 9-11, 2008, Lillehammer, Norwegen, IEEE Computer Society Press, 2008
13. Plösch R., Mayr A., Pomberger G., Saft M.: An Approach for a Method and a Tool Supporting the Evaluation of the Quality of Static Code Analysis Tools, Proceedings of SQMB 2009 Workshop, held in conjunction with SE 2009 conference, March 3rd 2009, Kaiserslautern, Germany, published as Technical Report TUM-I0917 of the Technical University Munich, July 2009
14. Product information about PC-Lint can be obtained via http://www.gimpel.com
15. Product information about PMD can be obtained via http://pmd.sourceforge.net
16. Product information about SISSy can be obtained via http://sissy fzi.de
17. Rombach H.D., Basili V.R.: Quantitative Software-Qualitätssicherung; In: Informatik Spektrum, Band 10, Heft 3, Juni 1987, S 145-158
18. Solingen R., Berghout E.: The Goal/Question/Metric Method; McGraw Hill Verlag, Berkeley, 1999
19. Solingen R.: The Goal/Question/Metric Approach; In: Encyclopedia of software Engineering, two-volume set, 2002