Balancing Upfront Definition and Customization of Quality Models

Michael Kläs, Jürgen Münch

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

Abstract. The selection and customization of quality models for the development of software systems or software-intensive systems and services is a challenging task. Due to the nature of software development, quality models such as reliability models or defect models need to be adapted to specific project goals and environment characteristics. Currently, there is a tremendous deficit in understanding the selection and customization of appropriate quality models. Quality models that balance upfront definition and customization are widely missing. This article describes two types of quality models, i.e., fixed models and define-your-own models, gives an initial overview of possible variabilities in quality models, and finally sketches elements for the definition of balanced quality models.

Keywords: Software Quality, Quality Assurance, Project Management, Quality Management, Quality Standards, Quality Definition, Quality Measurement

1 Introduction

In contrast to production engineering, software development usually produces individual results. In addition, the software development process is, to a large extent, a creative, human-based activity that varies from project to project. However, quality assurance in software development is often performed in a way similar to that in production engineering: Usually, it is assumed that each project is the result of the same process. In consequence, many software quality models are fixed quality models without systematic customization support. Quality models that can be tailored to the individual goals and environment characteristics of software development projects are widely missing.

Software quality can be seen from a multitude of perspectives and is relevant in nearly all application domains. In most cases, software quality is attached to products, i.e., artifacts such as design documents, code modules, or the resulting system. Definitions of *quality* provided in standards like ISO 8402 [1] usually focus on a product-based or user-based view: “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs”. ISO 14598 [2] provides a very general definition of a *quality model* in the context of information systems: ‘the set of characteristics and relationships between them which provides the basis for specifying

quality requirements and evaluating quality'. Following this definition, a quality model can be considered as an operationalization of the term quality.

Although most quality models presented in the literature or defined in standards are models for product quality (e.g., [6], [3]), they can also be attached to processes (e.g., maturity models, process adherence models), resources (e.g., server availability model, qualification model), or projects (e.g., milestone slippage model).

2 Fixed versus Define-Your-Own Models

In general, there are two main directions in software quality modeling [4]: On the one hand, there are *fixed-model approaches*, which usually prescribe important quality characteristics as a subset of the relevant quality focus in a published model. Sometimes, predefined measures for the quality characteristics are also given. On the other hand, *define-your-own-model approaches* support the derivation of relevant quality characteristics, i.e., they provide concepts and procedures on how to derive custom-tailored quality models.

Prominent examples of fixed-model approaches are the models proposed by McCall [5], Boehm [6], and ISO9126 [3]. Literature typically concentrates on them when discussing quality models. The models proposed by McCall [5] and Boehm [6] both present key characteristics of quality identified from a user perspective. ISO9126 [3] was developed in the early 1990s as an attempt to consolidate the many views of software quality. Further fixed model approaches are, e.g., the FURPS/FURPS+ model developed by Hewlett-Packard [7], [8], the DGQ model [9], and the SATC Software Quality Model [10] developed at NASA.

According to [11], it is not realistic to assume that one can provide a prescriptive set of necessary and sufficient quality characteristics to describe the quality requirements of any project; therefore, they propose specifying how quality characteristics should be specified rather than specifying a fixed set.

A prominent example of such a define-your-own-model approach is the SQUID approach [11]. It provides guidance in the customization of a quality model and aims at clarifying the link between process and product quality, but provides no detailed guidance regarding the measurement of quality characteristics.

To some extent, the GQM approach [15] can also be seen as an approach that supports the custom-tailored definition of quality models. The primary objective of the GQM approach is to provide a framework for defining measurement goals and deriving measures that help to answer these goals. Further define-your-own model approaches are presented, e.g., by Dromey [12] and Gilb [14].

3 Balanced Quality Models

Support of an appropriate level of upfront definition and opportunities for customization is an important criterion for the applicability of a quality model. On the one hand, it is in general not possible to define quality characteristics and measures that are valid for any purpose in any context. On the other hand, very generic define-

your-own approaches require very high skills and significant efforts for creating quality models. We state the following hypothesis: For specific domains and specific purposes, custom-tailored, so-called *balanced quality models* can be constructed by adapting a core model that captures the underlying quality concept in the specific domain. The adaptation needs to be guided by a detailed process so that it is reproducible. A prerequisite for defining such a balanced quality model is, besides other issues, a deeper understanding of the variable elements of quality models and their degree of variability. Therefore, below we sketch variabilities in quality models and potential impact factors on these variabilities.

Popular quality models such as [3], [5], [6], [7], and [11] have in common that they decompose quality into characteristics and sub-characteristics in a tree-like-fashion to make them ultimately measurable. Usually, this kind of approach is called Factor Criteria Metric approach [13]. Therefore, the (1) *decomposition of quality into characteristics* and the (2) *definition of metrics for characteristics* can be considered as the two basic elements of any quality model. They are sufficient if the model is used for characterizing quality.

If the model is to be used to derive development guidelines or to identify areas of possible improvements, factors influencing the quality (characteristics) have to be known at least on a qualitative level (e.g., avoiding redundancy in code increases final product maintainability, performing a fault tree analysis increases product robustness, developer's experience reduces product cost). Therefore, (3) *influencing factors* are required in quality models used for improvement.

If the model is to support the estimation of quality characteristics for planning a software project (e.g., cost estimation) or for controlling a project (e.g., prediction of product reliability), the relation between the influencing factors and the influenced quality (characteristic) have to be quantitatively modeled. Therefore, a (4) *quantitative model of influencing factors* is required in models applied for prediction.

Finally, if the model is to be used to plan overall quality and specify quality requirements (i.e., setting target values for the quality characteristics), the dependencies between different quality characteristics have to be known. Since quality characteristics can influence each other (e.g., an increased functionality can reduce the performance or usability of a product), tradeoff decisions must usually to be performed. Therefore, (5) *dependencies between quality characteristics* are an element in models applied in overall quality planning.

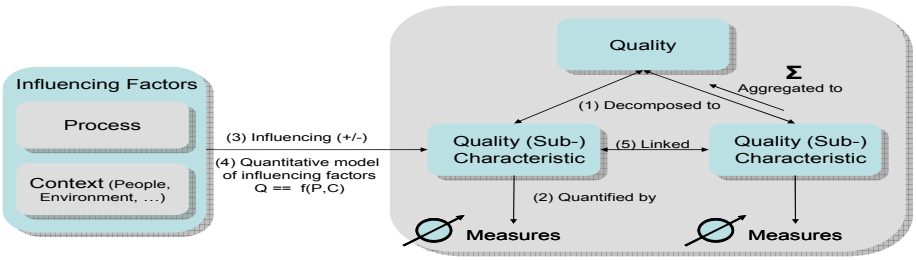

Fig. 1. Elements that can be customizable in a quality model

Fig. 1 gives an overview of the described concepts of a quality model. The mentioned variation points and influencing factors should not be seen as a comprehensive and complete list. There is rather an open research question of how to describe

customizable quality models that can be custom-tailored efficiently by practitioners in their own environment.

4 Conclusion

A balanced software quality model needs to be context- and purpose-oriented and should be derived from a domain-specific core model by using a fine-grained customization process. The definition of such models requires a much deeper understanding of the variabilities of quality models. As a vision, widely accepted domain-specific software quality standards that allow for systematic customization should be created.

5 References

1. ISO 8402, Quality management and Quality assurance - Vocabulary
2. ISO/IEC 14598 12207 International Standard, Standard for Information technology -- Software product evaluation -- Part 1: General overview
3. ISO/IEC 9126 International Standard, Software engineering – Product quality, Part 1: Quality model, 2001.
4. Norman E. Fenton and Shari Lawrence Pfleeger, Software Metrics - A Practical and Rigorous Approach. International Thomson Computer Press, 2nd edition ed., 1996.
5. James A. McCall, Encyclopedia of Software Engineering. Volume 2., ch. Quality Factors, pp. 1083--1093. John Wiley Sons, 2002.
6. Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. MacLeod, and Michhhael J. Merritt, Characteristics of Software Quality. North Holland Publishing Company, 1978.
7. Robert B. Grady and Deborah L. Caswell, Software Metrics: Establishing a Company-Wide Program, Prentice-Hall, 1987.
8. Robert B. Grady, Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, 1992.
9. Deutsche Gesellschaft für Qualität, Software-Qualitätssicherung. No. DGQ-NTG-Schrift 12-51, VDE-Verlag Berlin, 1986.
10. Larry Hyatt and Linda Rosenberg, A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, in Proceddings of the 8th Annual Software Technology Conference, 1996.
11. Barbara Kitchenham, Steve Linkman, Alberto Pasquini, and Vincento Nanni, The SQUID-Approach to Defining a Quality Model, Software Quality Journal, vol. 6, no. 3, pp. 211-233, 1997.
12. Geoff Dromey, A Model for Software Product Quality, IEEE Transactions on Software Engineering, vol. 21, pp. 146--162, Feb. 1995.
13. J.A.McCall, P.K.Richards, G.F.Walters, "Factors in Software Quality", RADC TR-77-369, 1977. Vols I,II,III', US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055, 1977.
14. Tom Gilb, Principles of Software Engineering Management, Addison-Wesley, 1988.
15. Victor R. Basili: Software Modeling and Measurement: The Goal/Question/Metric paradigm, Technical Report CS-TR-2956, Department of Computer Science, University of Maryland, College Park, MD 20742, 1992.