Chapter 6
Model-Based Quality Management of Software Development Projects

Jens Heidrich, Dieter Rombach, and Michael Kläs

Abstract Managing the quality of artifacts created during the development process is an integral part of software project management. Software quality models capture the knowledge and experience regarding the quality characteristics of interest, the measurement data that can help to reason about them, and the mechanisms to use for characterizing and assessing software quality. They are the foundation for managing software quality in projects in an evidence-based manner. Nowadays, coming up with suitable quality models for an organization is still a challenging endeavor. This chapter deals with the definition and usage of software quality models for managing software development projects and discusses different challenges and solutions in this area. The challenges are: (1) There is no universal model that can be applied in every environment because quality is heavily dependent on the application context. In practice and research, a variety of different quality models exists. Finding the "right" model requires a clear picture of the goals that should be obtained from using the model. (2) Quality models need to be tailored to company specifics and supported by corresponding tools. Existing standards (such as the ISO/IEC 25000 series) are often too generic and hard to fully implement in an organization. (3) Practitioners require a comprehensive set of techniques, methods, and tools for systematically specifying, adapting, and applying quality models in practice. (4) In order to create sustainable quality models, their contribution to the organizational goals must be clarified, and the models need to be integrated into the development and decision-making processes.

J. Heidrich (✉) • M. Kläs
Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
e-mail: jens.heidrich@iese.fraunhofer.de; Michael.Klaes@iese.fraunhofer.de

D. Rombach
Technische Universität Kaiserslautern, Kaiserslautern, Germany

Fraunhofer Institute for Experimental Software Engineering (IESE), Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
e-mail: rombach@informatik.uni-kl.de; dieter.rombach@iese.fraunhofer.de

G. Ruhe and C. Wohlin (eds.), *Software Project Management in a Changing World*,
DOI 10.1007/978-3-642-55035-5_6,

6.1 Introduction

When it comes to managing and controlling software development projects, three dimensions are typically addressed (assuming a defined functionality and project scope): cost, time, and quality. A variety of measurement-based approaches have been developed in recent years for managing and controlling projects in terms of these three aspects. However, while well-established measures exist to quantify cost and schedule aspects (such as those described in Chaps. 3 and 4), quality is a less tangible concept. As a consequence, it is still a challenge today to objectively measure software quality in early stages of the development process. Yet, being able to manage and control software quality is an integral part of professional project management (PMI 2008).

The difficulties become obvious when we take the definition of quality as being the "totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs" (ISO 8402 1995). On the one hand, software systems are becoming ever more complex (e.g., in terms of size, algorithms, and interfaces) and heterogeneous (e.g., in terms of development platforms and languages). This makes it more difficult to systematically analyze their quality. On the other hand, quality is heavily dependent on the application domain, the stakeholders, the usage context, and the specific project environment. For example,

- **Application domain**: Safety-critical systems (such as automobiles or power plants) require far different quality characteristics than information systems (such as accounting software or web applications)
- **Stakeholders**: A top-level manager of an organization probably has a different understanding of the term "software quality" than technical software development personnel
- **Usage context**: Different mechanisms need to be considered when trying to predict the final software quality at early stages of the development process compared to measuring the actual quality of a software design or component
- **Project context**: Having contractor-subcontractor relationships in a project requires a detailed understanding of what quality of externally delivered artifacts is actually needed compared to doing in-house development only

Depending on these aspects, different characteristics are important, and different techniques, methods, and tools have to be used as part of the software development process to guarantee a certain level of quality.

Software quality models are a means for defining and operationalizing the term "software quality." The typical approach is to refine the abstract term into more concrete subconcepts down to a level of detail where concrete metrics and indicators can be assigned. Depending on the structure of such a model, mechanisms for evaluating/assessing software quality are included. They support the stakeholders of a quality model in interpreting the measurement data (e.g., by defining thresholds distinguishing between acceptable and not acceptable quality) and in aggregating

the results in order to allow them to come up with an evaluation/assessment of the different quality characteristics of interest.

One of the most popular and extensive software quality models is published in the ISO/IEC standard 9126-1 (2001) and the corresponding standard for software product evaluation ISO/IEC 14598-1 (1999). It distinguishes three different views on product quality:

- *Internal Quality* describes the characteristics of intermediate artifacts of the development process required to satisfy internal quality requirements.
- *External Quality* describes externally visible quality characteristics of artifacts of the development process required to satisfy external quality requirements. They typically specify the quality of the product delivered to the customer.
- *Quality in Use* defines quality characteristics from the perspective of a user, which are of importance when the software product that was delivered to the customer is actually used.

The assumption underlying the definition of these views is that having high-quality processes helps to obtain artifacts with good internal quality; creating artifacts with high internal quality supports achieving a final product with good external quality; and a product of high external quality leads to good user experience with the product, that is, good quality in use (ISO/IEC 9126-1 2001).

For all three views on software quality, the standard provides models defining a breakdown structure of quality into subconcepts. For example, internal and external quality is broken down into functionality, reliability, usability, efficiency, maintainability, and portability. Each of these high-level quality characteristics is broken down into corresponding subcharacteristics. The models for internal and external quality essentially comprise the same quality characteristics, but use different metrics for actually measuring product quality in the end.

Recently, the successor to these two standards has been published, the ISO/IEC 25000 "Software Product Quality Requirements and Evaluation (SQuaRE)" (ISO/IEC 25000-1 2005), which comprises a whole series of standards related to quality management addressing models for product quality (external and internal quality) and quality in use (2501n series), measuring product quality (2502n series), and evaluating/assessing product quality (2504n series). Moreover, standards have been added that address how to define and systematically derive quality requirements, that is, nonfunctional requirements of software products (2503n series).

The ISO/IEC standards specify a set of quality models that try to be generic enough to be applicable for all kinds of software systems. Indeed, models proposed to be universal, such as those described in the ISO/IEC standards, stay on a high-level of abstraction, making it difficult to instantiate and apply them. In literature, a variety of more specific software product quality models can be found for different application domains, stakeholders, usage purposes, and project environments (Kläs et al. 2009). However, in practice, it is quite difficult to sustainably apply these models for managing the quality of the outcome of a software development project (Wagner et al. 2010b) (overview):

- **Challenge 1**: There is no universal quality model that can be applied in every environment. In practice and research, a variety of different quality models exists. Finding the "right" model requires a clear picture of the goals that should be obtained from using the model in a software development project. Goals may range from getting a common understanding of important quality characteristics when specifying requirements to determining the maintainability of the source code to getting early indicators for potentially failing performance or reliability targets. In order to quantitatively manage the fulfillment of the nonfunctional requirements of a product under development, having the right model is essential. Otherwise, there is a high risk of a lot of effort being spent on collecting data of limited use for the success of the project
- **Challenge 2**: Quality models need to be tailored to company specifics and supported by corresponding tools. Existing standards (such as the ISO/IEC 25000 series or the predecessor ISO/IEC 9126) are too generic and hard to fully implement in an organization. Moreover, it is difficult to come up with reliable measures and evaluation criteria. Nevertheless, quality models as proposed in respective standards may be a good starting point. However, if models are not further tailored, there is a high risk that either the models will not find all quality issues, which will in turn lead to bad product quality, or the models will find too many issues, which will be hard to prioritize and comprehensively address in the project
- **Challenge 3**: Cost-effective application of quality models requires a comprehensive framework to facilitate their specification, adaptation, and practical usage for software product evaluation/assessment. Without such a framework that can be used right out of the box, more effort will typically be spent on thinking about how to construct and implement such quality models from scratch than on actually thinking about the content of the models itself and about how product quality can effectively be managed in a development project
- **Challenge 4**: In order to create quality models that are sustainably used for decision-making, their contribution to and value for the organizational goals have to be clarified, and the models need to be integrated into development and decision-making processes (e.g., by defining appropriate quality gates). Without this integration effort, model-based quality management remains a project-specific effort largely dependent on the mechanisms the project manager wants to use for managing product quality. In order for models to be used effectively, the organization as a whole needs to implement a framework to think about software quality and how this contributes to their overall strategy across multiple development projects

This chapter deals with the definition and usage of software quality models for monitoring and controlling the quality of artifacts produced during the software development life cycle. It discusses the different challenges mentioned above in more detail and illustrates solutions based upon first-hand experience from different industrial applications.

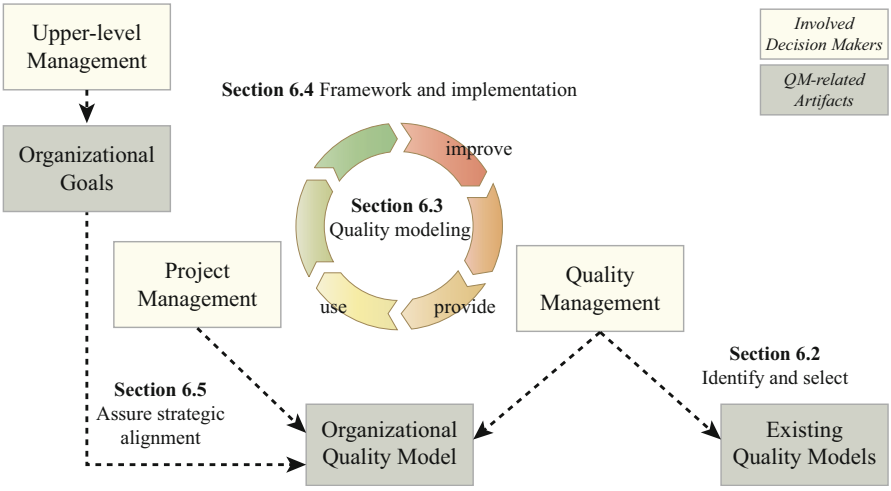

Fig. 6.1 Overview of the structure of Chap. 6

Figure 6.1 gives an overview of the content of this chapter and embeds it into the context of project and quality management. The order in which companies are typically confronted with the introduced challenges determines the order of the subsequent sections: Sect. 6.2 deals with the selection of suitable quality models for a specific organization (Challenge 1), Sect. 6.3 deals with a process for building custom-tailored models that can be used for evaluating product quality during the software and system development process (Challenge 2), Sect. 6.4 introduces a recently finalized, comprehensive framework for specifying and applying quality models in practice (Challenge 3), and Sect. 6.5 deals with integrating quality models into organizational goals and strategies (Challenge 4). Section 6.6 summarizes the chapter and gives an outlook to future work.

6.2 Selecting the Right Quality Models

In practice and research, a variety of different quality models exists that is designed for different application domains, stakeholders, usage contexts, and project environments. The difficult question is: Which quality models are relevant and can be applied in a given environment? To answer this question, a clear picture of the underlying goal to be achieved by a quality model is required. In order to state this goal more precisely, some key questions have to be answered, such as

- Which artifacts of the development process should be analyzed (e.g., requirements document, architecture, design, or code)?

- For which purpose should the model be used (e.g., measuring the product quality of some artifacts or predicting the quality of the final software product delivered to the customer)?
- Which quality characteristics are of interest (e.g., quality in general or some subcharacteristics, such as maintainability or reliability)?
- Which stakeholders want to use the analysis results (e.g., project managers, suppliers, developers)?
- In which environment and context is the model to be applied (e.g., in the development of safety-critical embedded systems or web-based services)?

A systematic literature review and state-of-the-practice survey (Kläs et al. 2009) revealed 79 product quality models[1] ranging from general standards related to software product quality, such as IEEE 1061 (1998), ISO/IEC 9126-1 (2001), ISO/IEC 25000-1 (2005), IEC 61508-1 (2010), ISO/IEC 14598-1 (1999), or ISO/IEC 15939 (2007) via domain-specific standards such as MISRA (1995), ECSS (1996), or UKMD (1997) to quality models from academic research, such as Cavano and McCall (1978), Boehm (1978), Dromey (1998), or Avizienis et al. (2001). It is beyond the scope of this chapter to discuss these models in general. However, for illustrating purposes, Fig. 6.2 gives an overview of the quality models identified and their year of creation, distinguishing between official and de facto standards, models that are actually applied in practice, and models having a more scientific background. Furthermore, it separates models addressing quality in general (dark gray boxes) from models addressing specific quality characteristics (light gray boxes).

If a suitable quality model or a set of quality models needs to be identified, a classification scheme is needed for distinguishing among the different types of existent models. The scheme presented in Kläs et al. (2009) discusses the following dimensions:

(a) **Structural components**: How is the quality model structured? What structural components are supported? For example, models can include structures for refining quality characteristics, for measuring quality characteristics, for evaluating measurement data, for aggregating results, etc.
(b) **Quality modeling goal**: What is the goal addressed by the quality model? In particular, who will use the model, for which purpose, to address which aspects of quality, and on what kinds of artifacts?
(c) **Model instantiation**: Does the quality model only prescribe a structure for specifying quality models (called metamodel) or does it also include an instantiation of that structure [i.e., a specific quality model such as those described in ISO/IEC 25010 (2011)?]

[1] A product quality model is a conceptual or mathematical model addressing one or more relevant characteristics of certain types of work products (such as requirements, design, code, documentation, or the final product) with the objective of better understanding and dealing with these characteristics (e.g., by specifying or quantifying them or correlating them with others).

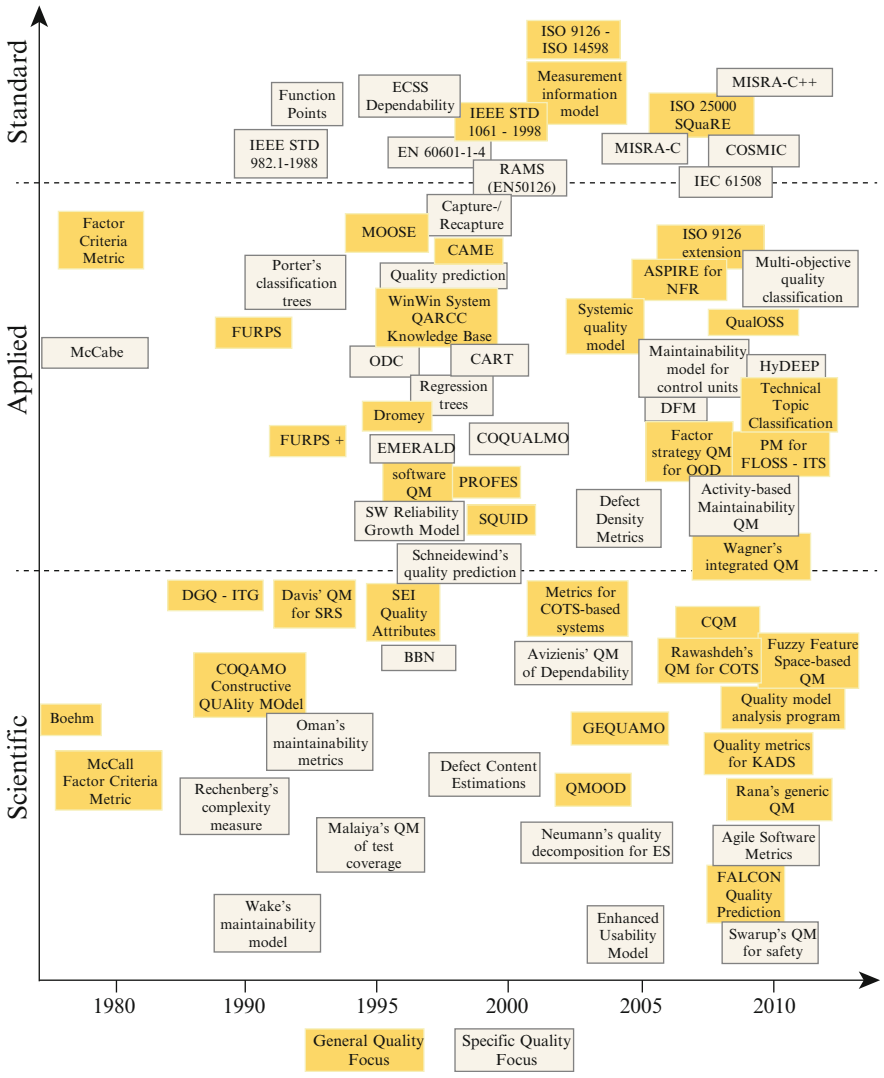

Fig. 6.2 Quality model landscapes

(d) **Method for instantiation/adaptation**: Is a procedure provided to instantiate the provided metamodel or adopt/tailor the presented quality model to specific needs?

(e) **Dissemination**: What is the degree of dissemination of the quality model? For example, is it only used in a scientific context, is there some evidence that it is actually applied in practice, or is it an official or de facto standard?

(f) **Tool support**: Is the quality model tool-supported in terms of specifying and adapting the model as well as actually applying the model to software products?

For example, there may exist commercial tool support, only prototypical tool support, or no tool support at all.

Based on the work in Kläs et al. (2009), the following subsections give further insights into what structural components of models look like (dimension a), discuss the specification of quality modeling goals (dimension b), and describe quality model landscapes as a means for systematically selecting suitable models (based on dimensions a–f).

6.2.1 Structural Components

Figure 6.3 depicts a graphical illustration of typical structural components of quality models. Mainly depending on the application purpose, different components are required by different quality models.

In general, two different parts of a model can be distinguished. The left side shows components related to the quality focus; that is, the quality characteristics of interest. The right side shows the components related to so-called variation factors, that is, factors that explain variances when analyzing quality characteristics. For example, the maintainability of a piece of software (as one characteristic in the quality focus) may be influenced by the experience of the developers or the amount of reuse that was performed (as two potential variation factors).

The relationships between quality focus characteristics and variation factors may be expressed qualitatively or quantitatively. For example, a quality model may specify that developers with higher experience produce software that is easier to maintain (quantitative expression), or it may specify that by increasing the average experience of the developers by 1year, maintenance costs can be reduced by 10 % (quantitative expression). Most quality models rely on qualitative relationships or do not specify variation factors at all. The reason for that is that in order to be able to make such statements, a very thorough understanding of the impact relationships is required. From building cost estimation (Trendowicz et al. 2006) and defect prediction models (Kläs et al. 2010b) (prediction), we know that the variation factors themselves and especially the quantitative relationships highly depend on the particular organization for which such a model is built.

The quality focus may be further refined by a breakdown structure of concepts (typically quality characteristics). For example, ISO/IEC 25010 refines maintainability into modularity, reusability, analyzability, modifiability, and testability. These subconcepts may then be quantified using concrete metrics that can be used for actually measuring the subconcepts. For example, ISO/IEC 25021 (2012) defines concrete metrics for measuring the subconcepts of maintainability. In addition, a quality model may define evaluation criteria for interpreting the measurement data and evaluating/assessing the concepts of interest. For example, ISO/IEC 25040 (2011) defines a quality evaluation reference model and guide. Finally, a quality model may specify a procedure for aggregating the evaluation

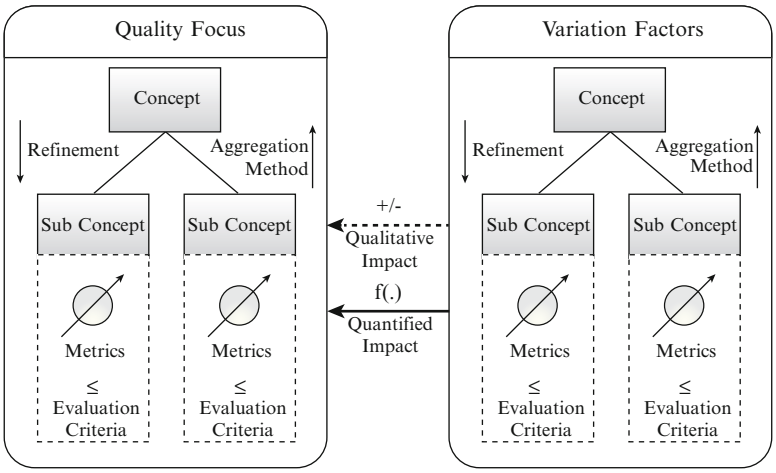

Fig. 6.3 Conceptual elements of quality models [cf. Kläs et al. (2009)]

results of the measurement data, resulting in an overall statement regarding the main concept (across the different refinement levels). For example, if a unique evaluation scale is defined for all concepts, the aggregation may be as simple as the weighted average across the results of all relevant subconcepts, or it may make use of advanced techniques from the field of multi-criteria decision analysis (MCDA) (Trendowicz et al. 2009).

Variation factors can be refined, quantified, evaluated, and aggregated in a similar way as the quality focus (e.g., the concept of developer experience can be refined into domain and programming experience and can be quantified by the respective years of professional experience).

In Sect. 6.3, we present the creation of an example quality model for assessing maintainability, which is a good illustration of the structural components discussed above (Fig. 6.2).

A quality model may be characterized according to the structural components supported by the model. This gives valuable information about what can actually be accomplished in practice when using the model. For example, if a model does not specify evaluation rules, it is possible to measure quality characteristics, but it is not possible to evaluate/access the product's quality (which can be perfectly alright depending on the goals related to the use of the model).

6.2.2 Quality Model Goal

Kläs et al. (2009) use the Goal-Question-Metric paradigm (GQM) (Basili et al. 1994a) as a schema for specifying the goals related to using a quality model. The GQM approach is a de facto standard for defining goal-oriented

measurement programs. It supports organizations in clearly specifying measurement goals and systematically deriving metrics from these goals. The GQM goal template distinguishes five parameters that can easily be reused for specifying quality modeling goals:

- **Object**: Specifies the object that is analyzed by the quality model, for example, one or several types of artifacts from the development process, such as requirements specifications, design documents, or code (or parts thereof)
- **Purpose**: Specifies the usage purpose for which the quality model is built, such as characterization, understanding, evaluation, prediction, or improvement
- **Quality focus**: Specifies which quality characteristics are analyzed, for instance, product quality in general or certain quality characteristics as defined by ISO 25010, such as maintainability, reliability, or usability
- **Viewpoint**: Specifies the stakeholders interested in the results obtained by applying the quality model, for example, a quality manager (for assuring product quality at certain milestones of the development process) or a developer (for identifying quality issues and improving the quality of the affected artifacts)
- **Context**: Specifies the organizational scope of the quality model and the application domain covered by the quality model, for example, the model might focus on software developed in a certain business unit or on artifacts created for a certain class of projects; the model may be constructed for the domain of embedded software systems, management and information systems, or web applications

Figure 6.4 illustrates potential usage purposes of quality models (based upon typical GQM purposes) in connection with the structural components of a quality model needed for supporting these purposes:

1. **Specify**: The model is only used for specifying the term "product quality" by refining it into subqualities. Such models specially help to structure quality requirements or issues and define a common understanding of quality. For example, ISO/IEC 25010 defines a breakdown structure of quality characteristics
2. **Measure**: The model is used for quantifying quality characteristics by using metrics. For example, an organization wants to determine a baseline for software product quality based on the measures defined in ISO/IEC 25022
3. **Monitor**: The model is used for identifying trends in quality-related figures and making sure that a certain level is maintained by monitoring measurement data (values of metrics) over time. For example, an organization wants to make sure that the interface complexity of software components does not increase
4. **Assess**: The model is used for assessing/evaluating the quality of dedicated artifacts by checking that quality requirements are fulfilled. For example, it is checked that the application's response time is less than 2 s
5. **Control**: The model is used for assessing product quality periodically or at certain points of the development process. For example, product complexity is checked against a defined threshold every week

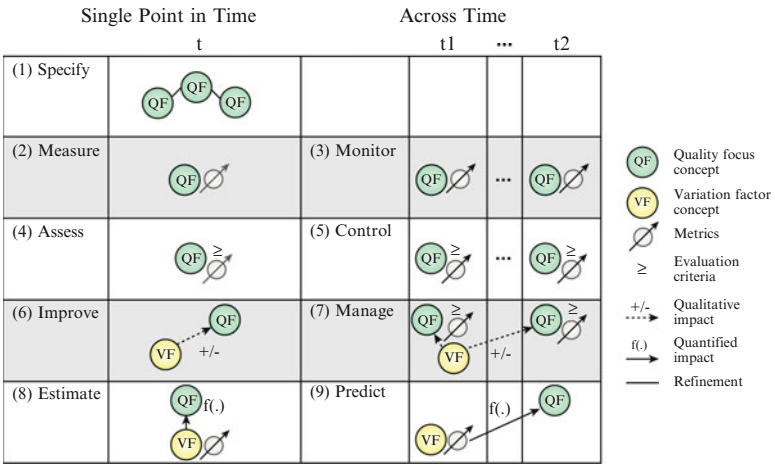

Fig. 6.4 Application purposes of quality models [cf Kläs et al. (2009)]

6. **Improve**: The model is used for improving quality characteristics by manipulating and controlling variation factors that have an impact on these characteristics. For example, coding guidelines (as a variation factor) are introduced to increase maintainability (as a quality characteristic)
7. **Manage**: The model is used for managing product quality over time by controlling product quality periodically or at certain points of the development process and making use of variation factors if quality requirements are not fulfilled. For example, involving more experienced developers (as a known variation factor) if the design of the system has some significant quality issues
8. **Estimate**: The model is used for estimating quality characteristics by making use of variation factors (e.g., because the quality characteristics cannot be measured directly). For example, Capture-Recapture models estimate the number of remaining defects by using the information about the number of joint defects found by different reviewers of the same artifact (Petersson et al. 2004)
9. **Predict**: The model is used to predict quality characteristics in the future, based on variation factors that can be measured earlier. For example, this is the principle of cost estimation models, but also of models that try to predict the number of defects in a delivered software product at early stages of the development process, such as Kläs et al. (2010b) (prediction). Where estimation models are used to determine the present but unknown state of a quantity such as the number of defects in the currently investigated artifact, prediction models make statements about the future, such as the prospective performance of the final product based on the analyzed design documentation

6.2.3 *Quality Model Landscapes*

Quality model landscapes make use of the classification scheme described at the beginning of this section to provide an overview of available quality models and to allow selecting those models that best fit the needs of an organization. An organization specifies its needs in terms of analyzing software product quality by filling in the classification scheme and thinking about the structural components needed, the goals related to using the quality model, and the other dimensions mentioned above. The landscapes presented in this chapter are based on the work described in Kläs et al. (2009) and combine the results of a literature review and a survey targeting practitioners and their experiences with quality models (Wagner et al. 2010a) (survey).

For example, the quality model overview diagram (Fig. 6.2), which was introduced in the beginning of this section, illustrates a simple quality model landscape visualizing three dimensions: (1) the creation date of the model (from the context information of field b of the classification scheme); (2) whether quality in general is addressed or a specific aspect (from the quality focus information of b); and (3) the dissemination of the models (from field e of the scheme).

Table 6.1 shows another quality model landscape illustrating the number of quality models identified in a systematic literature review and presented in Fig. 6.2 according to two other dimensions: (1) the usage purpose and (2) the main quality focus addressed as mentioned in the model descriptions. For example, if an organization wants to estimate the number of defects, seven potential models are of interest. As can be seen in the table, most of the models from the survey deal with product quality in general and support the specification, measurement, or assessment of product quality. Models considering the defect-proneness of products are mainly estimation or prediction models.

6.3 Building Custom-Tailored Quality Models

This section deals with how to build a custom-tailored quality model in practice. First, general strategies and the high-level construction process are shown. Afterward, the detailed steps for constructing a concrete model are illustrated.

There are two contrary strategies that can be followed: (1) An existing quality model may be selected (such as the one from ISO/IEC 25000) and applied or (2) a new custom-tailored model is built from scratch that exactly fits the needs of an organization. In this section, we will first discuss these two extremes and then explain why and how to combine these strategies to overcome their limitations.

The advantage of the first strategy is that a set of predefined quality characteristics and metrics is available that were elicited based on the knowledge and experience of experts from other organizations and institutions. The disadvantage is that existing quality models are often quite abstract and therefore hard to use out

Table 6.1 Quality models covering different usage purposes and quality focus [cf. Trendowicz et al. (2009)]

	General	Defects	Dependability	Functionality	Maintainability	Portability	Reliability	Safety	Usability
Specify	20		4		1	2		4	1
Measure	16	1		2	2				1
Monitor	1	2					1		
Assess	13				2			1	
Control		1			1				
Improve	7		2		2		1		
Manage	4	1							
Estimate	2	7							
Predict	4	5			1		3		

of the box. For example, the metrics proposed for the ISO/IEC 25000 series are very hard to implement in an organization. Another disadvantage is that the more detailed existing models were created for a certain context and cannot be transferred to a different context that easily. For example, a quality model that was designed for embedded systems cannot be directly transferred to and applied to information systems.

The advantage of the second strategy is that the model ideally fits the specific needs of an organization and can be designed in such a way that it integrates already available measurement data and data collection tools and fits into the organizational and development processes. The disadvantage is that guidance is needed for building up a meaningful quality model; therefore, expert knowledge and experience are needed. Moreover, this is a labor-intensive process and involves experts experienced in model building as well as experts of the corresponding organization the model is being built for. According to Wagner et al. (2010a) (survey), more than 70 % of 125 respondents of a survey employ company-specific models (either developed from scratch or tailored on the basis of company-specific requirements).

Kitchenham et al. (1997) propose a combination of these two strategies to overcome this gap by first choosing the most appropriate existing quality model and, then reusing this model or parts thereof for defining a custom-tailored model.

Figure 6.5 illustrates a process for developing a company-specific quality model based on the Quality Improvement Paradigm (QIP) (Basili et al. 1994b):

1. **Characterize**: Define the scope and application environment in which the organization wants to use the quality model
2. **Set goals**: Define the quality modeling goals (using the goal parameters introduced in the previous section), analyze the suitability of existing models with respect to these goals, select the most appropriate quality model if existent, and then adapt the model to the specific needs of the organization or build the model from scratch if no existing model could be found
3. **Choose process**: Define a measurement plan containing a list of metrics and additional data about who is responsible for collecting this data at which point in time of the development process. Define mechanisms/processes for data collection, processing, and visualization as well as corresponding tool support. The latter includes integrating the usage of the quality models at predefined stages of the development process
4. **Execute**: Apply the quality model to artifacts of the development process, collect measurement data, and assess/evaluate product quality based on evaluation guidelines
5. **Analyze**: Analyze and validate the assessment results and check the validity of the quality model
6. **Package**: Improve the quality model, if needed, and initiate actions for improving the software product quality, if needed

The following sections give a practical usage example of the process as described above based upon a real industrial application. For reasons of

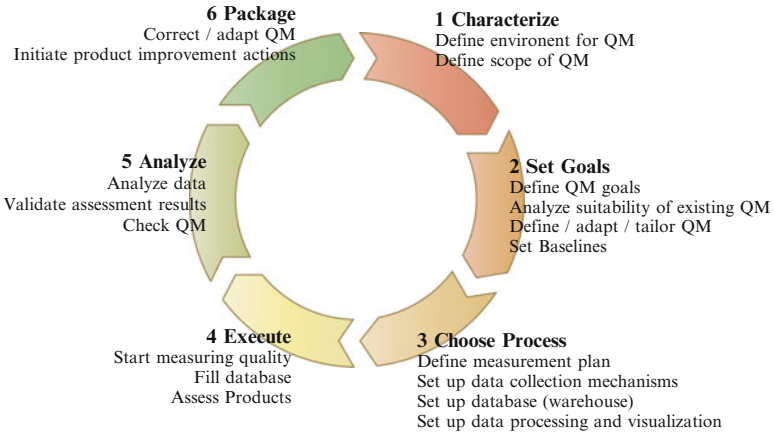

Fig. 6.5 Developing custom-tailored quality models

confidentiality, the original quality modeling goals, the quality model itself, as well as the analysis results have been carefully modified.

Additional industry-related lessons learned from building custom-tailored quality models following the process shown above can also be found in Lampasona et al. (2012).

6.3.1 Characterize: Define Environment and Scope

The first step is about defining the scope and application environment in which the organization wants to use the quality model. Our example organization develops safety-critical systems. In recent years, more and more functionality has been implemented as software. However, software is usually developed by external suppliers and delivered to the system manufacturer as an integrated unit containing the software as well as the hardware (processors, controllers, memory, etc.) the software runs on. The manufacturer has to integrate all these delivered units into an overall system and ensure that they can communicate with each other. The scope of the quality model has been defined as focusing on the software components developed by external suppliers and their integration into the overall system.

6.3.2 Set Goals: Define Goals and Build Quality Model

The second step is about defining the quality modeling goals, analyzing the suitability of existing models with respect to these goals, selecting the most appropriate quality model, and adapting the model to the specific needs of the

organization. The major organizational goal behind the usage of the quality model is the reduction of maintenance costs. This should be achieved by ensuring that the software components are of high quality in order to reduce problems during integration, testing, and rework. For this reason, the main goal related to software development is to improve the maintainability of a software component delivered by external suppliers. For assessing/evaluating the quality of the software delivered, a quality model is used.

The quality modeling goal is as follows:

- **Object**: Software components delivered (mainly code and interfaces)
- **Purpose**: Evaluation/assessment
- **Quality focus**: Maintainability
- **Viewpoint**: Quality manager and external supplier
- **Context**: Construction of safety-critical embedded systems

Based on this goal specification, ISO 25010 was used as a reference quality model. However, it was decided to establish custom-tailored metrics and corresponding evaluation rules for measuring and assessing quality characteristics of interest. The tailoring was performed by external quality modeling experts based on a series of workshops and interviews. First, structured interviews with domain experts from the organization were performed, and an initial quality model based on ISO 25010 was created. After that, the model was reviewed and discussed with a broader group of stakeholders (quality managers and suppliers) for achieving a basic consensus among the experts.

An overview of the model obtained is presented in Fig. 6.6. Basically, the quality model created consisted of two parts: one part containing quality characteristics directly related to maintainability and another part containing variation factors influencing maintainability. The main focus should be on analyzability and adaptability. As a consequence, only these two quality characteristics were refined following the classical GQM approach in order to determine metrics that fit these characteristics. For each characteristic, a few metrics were assigned, grouped into different technical topics. For example, analyzability is measured based on the coupling of internal software components. Coupling is measured using standard metrics such as Fan-In (ingoing relationships), Fan-Out (outgoing relationships), and Coupling between Objects (CBO) (Dörr et al. 2004). The variation factor part of the model contains, for example, metric groups related to the logical and physical partitioning/distribution of the software functionality that cannot be influenced by the pure software component, such as a high coupling with other units and the software components of those units. In our example, this cannot be influenced by the external software supplier and becomes part of the variation factor model.

Each metric, group of metrics, quality characteristic, and variation factor has a certain weight that defines how important the respective element is with regard to the parent element. Evaluation rules (employing simple mathematical functions) provide a mapping between measurement values and an evaluation scale. Afterward, a weighted sum is calculated for the groups, quality characteristics, and variation factors considered. The top-level grades make statements about the

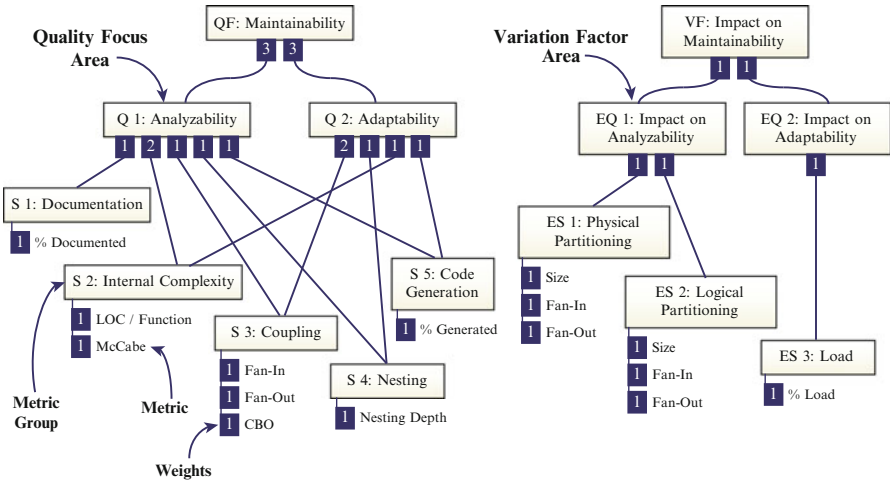

Fig. 6.6 Example quality model for maintainability

maintainability of the delivered software components and external factors that may explain variations between the quality characteristics.

There are several options regarding how to define evaluation rules, such as Trendowicz et al. (2009), and create a mapping function. The difficult part lies in identifying meaningful thresholds for distinguishing between good and bad values. An organization may retrieve values from literature. However, these values normally need to be adapted for the specific usage context. Therefore, most organizations building quality models create their own database of measurement values and perform some statistical analysis on these values in order to find realistic values and thresholds.

The assessment scale for quality characteristics was defined from 1 = very bad maintainability, 2 = bad, 3 = neither good nor bad, 4 = good, to 5 = very good maintainability. The assessment scale for variation factors was defined from 1 = very negative influences on maintainability, 2 = negative influences, 3 = no influences, 4 = positive influences, to 5 = very positive influences on maintainability. For example, if maintainability was rated with 4 and the variation factors were rated with 2, the software component provided good maintainability despite the fact that there was a negative influence.

6.3.3 Choose Process: Set Up Data Collection and Analysis

Step 3 is about defining a measurement plan as well as mechanisms/processes for data collection, processing, and visualization as well as corresponding tool support. Table 6.2 presents the measurement plan (Differding 2001) that was created for the quality model, gives a list of all metrics, and defines the range of measurement

Table 6.2 Measurement plan

ID	Range	Collection time	Data source	Resource
% Documented	%	Quality Gates	Static Code Analyzer	Supplier
LOC/Function	R	Quality Gates	Static Code Analyzer	Supplier
Average McCabe	R	Quality Gates	Static Code Analyzer	Supplier
Fan-In	N	Quality Gates	Static Code Analyzer	Supplier
Fan-Out	N	Quality Gates	Static Code Analyzer	Supplier
CBO	N	Quality Gates	Static Code Analyzer	Supplier
Nesting Depth	N	Quality Gates	Static Code Analyzer	Supplier
% Generated	%	Quality Gates	Static Code Analyzer	Supplier
Physical Size	N	Quality Gates	Interface Data	Manufacturer
Physical Fan-In	N	Quality Gates	Interface Data	Manufacturer
Physical Fan-Out	N	Quality Gates	Interface Data	Manufacturer
Logical Size	N	Quality Gates	Interface Data	Manufacturer
Logical Fan-In	N	Quality Gates	Interface Data	Manufacturer
Logical Fan-Out	N	Quality Gates	Interface Data	Manufacturer
% Load	%	Quality Gates	Questionnaire	Supplier

values, the point in time of the development process when the data should be collected (and analyzed), the data source, and who is responsible for data collection.

A questionnaire was created to get manual measurement data from external suppliers. Moreover, a static code analysis tool was used for extracting code metrics. Due to confidentiality reasons, the code was analyzed on the suppliers' side, and only the measurement values were exchanged. Moreover, interface data was extracted from the configuration database of the organization (containing an overview of all units provided by the suppliers and their interfaces). The organization decided to store all data in a relational database for further analysis and visualization. Figure 6.7 gives an overview of the general procedure for collecting and analyzing the data at certain quality gates of the organization's development process:

- Raw measurement data is collected from different artifacts of the development process (in our case, basically code and interface data) based upon the metrics defined in the quality model. This process is driven by the data collection resources mentioned in the measurement plan. The raw measurement data is stored in a database
- Afterward the measured artifacts are evaluated on the basis of the evaluation rules defined in the quality model. An assessment report is created containing the results of the evaluation. This process is driven by experts for the quality model (typically people from the quality management part of the organization and/or external measurement experts)
- The assessment results are discussed with the relevant stakeholders, in this example the suppliers and the responsible project manager of the organization. A list of countermeasures is created for coping with the weaknesses identified in the quality assessment. The succeeding quality assessment, conducted after the next quality gate of the development process, will check whether the defined

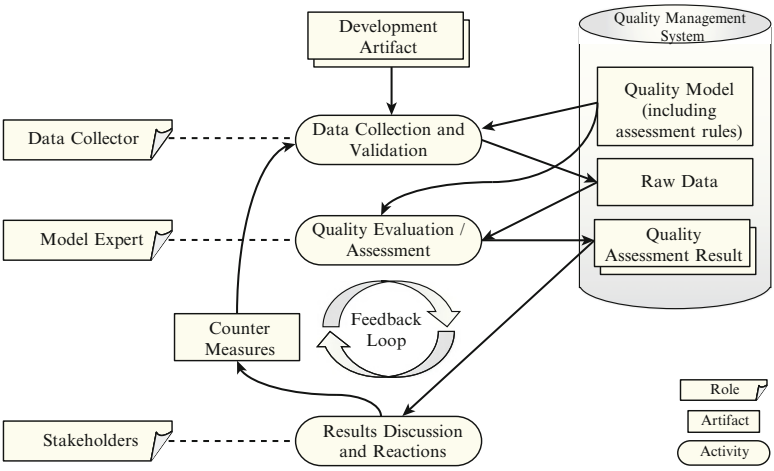

Fig. 6.7 Making use of models for quality assessments

countermeasures had the intended effect and probably make some adjustments, so that an overall feedback loop is created

6.3.4 Execute: Use the Quality Model for Evaluation/ Assessment

Step 4 is about applying the quality model to artifacts of the development process, collecting measurement data, and assessing/evaluating product quality based upon evaluation guidelines. The maintainability quality model was applied to 20 already delivered software components of different external suppliers of the organization. Measurement data was collected according to plan, the data was mapped to the evaluation scale described above, aggregated according to the defined weightings in the quality model, and finally, the quality of the delivered software components was analyzed by means of pairwise comparison. In order to avoid comparing apples and oranges, only meaningful comparisons were selected.

The main purpose of the first application of the quality model was to execute a proof of concept and calibrate the model for future usage. To define reliable baselines, more software components would have to be analyzed to allow sound statistical analyses. An example comparison of two software components can be found in Table 6.3. As can be seen from the table, SC1 was evaluated as having good maintainability (4.0 on a scale from 1 to 5) despite having a slightly negative influence on maintainability resulting from the external variation factors (2.5). In contrast, SC2 was evaluated as having poor maintainability (1.75) while having a slightly positive influence on maintainability (3.5). The same argument is generally true for both of the quality subcharacteristics analyzability and adaptability.

Table 6.3 Example results comparing software components SC1 and SC2

	SC1	SC2
Quality focus area		
QF: Maintainability	4.00	1.75
– Q1: Analyzability	3.50	2.00
– Q2: Adaptability	4.50	1.50
Variation factor area		
VF: impact on maintainability	2.50	3.50
– EQ1: impact on analyzability	2.00	4.00
– EQ2: impact on adaptability	3.00	3.00

6.3.5 Analyze: Analyze Evaluation/Assessment Results

Step 5 is about analyzing and validating the assessment results and checking the validity of the quality model. For this purpose, the assessment results were compared with expert statements from the organization regarding the actual maintainability of the software components in practice. The experts gave pairwise ratings about which software component is actually better in terms of maintainability. The quality model rated the software components in the same way as the experts in 90 % of all cases. For the 10 % of failed ratings, the organization observed a significant difference in the requirements implemented in the software component. Because of the limited number of data points, no general statement about the validity of the quality model could be made at that point in time.

6.3.6 Package: Define Improvement Actions

Step 6 is about improving the quality model if needed and initiating actions for improving software product quality. In our example case, the analysis results were discussed in a workshop, and feedback to the data collection resources was provided. The evaluation results of the software components were traced back to concrete metric results (e.g., high coupling, complexity) and corresponding improvement actions have been launched for refactoring the components. Furthermore, based on this discussion, the following improvement actions were defined for the future use of the quality model in the organization:

- Increase the number of analyzed software components and cluster them according to the different types of units to improve the interpretability of the results
- Analyze the distribution of measurement values in the database to establish more reliable baselines/thresholds for the different clusters identified
- Focus on automatic data collection to reduce data collection effort
- Set up means for dealing with missing data to increase the robustness of the quality model

6.4 Specification and Application of Quality Models

The previous sections illustrated how to select an appropriate quality model and how to adapt and make use of such a model in practice for systematically analyzing the quality of software products. However, there is little tool support available for specifying, customizing, and applying these models. This makes it hard and very effort-consuming to develop complete and consistent quality models. A comprehensive framework is needed for the cost-efficient specification, adaptation, and practical usage of quality models. This problem was addressed in the Quamoco research project, which aimed at creating a "Quality Standard for Software-Intensive Systems" (Wagner et al. 2012).

The major motivation was the lack of a mandatory, applicable, and tool-supported quality standard for software development comparable to standards in other industries. A large number of different quality models addressing different goals exist (as seen in Sect. 6.2). However, the existing models are hard to use because the abstraction level is often too high, and it is difficult to come up with reliable and collectable measures. Moreover, adapting/tailoring the models to the specific needs is an effort-consuming process. There is also a lack of reliable evaluation criteria and little support for the meaningful aggregation of quality assessment results, which inhibits meaningful comparisons and benchmarking.

The Quamoco approach implemented the idea of balanced quality models, which means that detailed but highly adaptable core models are provided together with a fine-grained customization process (Kläs and Münch 2008). In Quamoco, the core consists of a quality base model comprising quality characteristics, metrics, and evaluation rules that are essential for all kinds of software-intensive systems. Based on this base model, different domain-specific extensions were created together with appropriate domain experts from various industries (e.g., for information systems, embedded systems, custom software development, etc.). These domain-specific quality models can be customized to the specific needs of an organization (or a part of an organization) using the Quamoco tool suite.

The tool suite supports a user in selecting an appropriate domain-specific model and tailoring this model to the specific needs of the organization by removing or modifying existing parts or adding completely new entities to the model. Furthermore, all models have default connections to measurement instruments actually collecting and analyzing the data and feeding the analysis results back into the model. Figure 6.8 shows a screenshot of the Quamoco quality model editor (downloadable from http://www.quamoco.de). As general quality models tend to become complex, visualization support is essential to get an overview of the quality characteristics addressed and the relationships modeled. The figure shows a sunburst (Stasko 2013) diagram visualizing the model structure of the Quamoco base quality model (which is available at http://www.quamoco.de), together with the evaluation results for an analyzed system. The right part of the sunburst shows the hierarchy of quality characteristics.

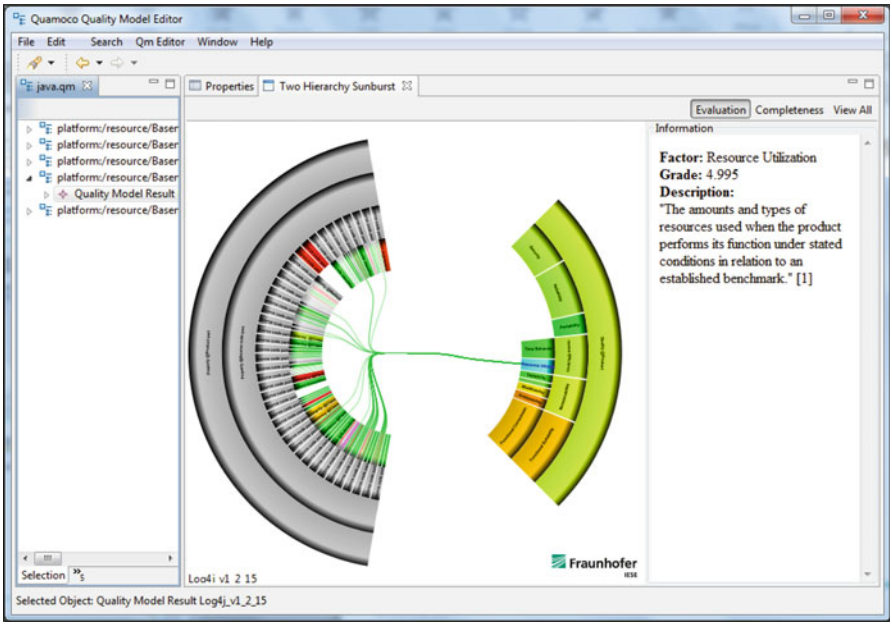

Fig. 6.8 Quamoco quality modeling tool suite

The left side comprises the measured product-related factors (variation factors) in the model. Each quality characteristic is impacted by different product-related factors; the lines visible in the figure indicate all product-related factors impacting the quality characteristic Resource Utilization (the right hand-side target of all visible lines). The information pane next to the sunburst shows the results of the selected quality characteristic for the system under evaluation (Log4J, a Java-based logging mechanism). The color encoding of the sunburst diagram indicates the evaluation results of quality characteristics and technical factors (from "green" equaling no quality issues to "red" equaling many quality issues). Furthermore, the tool supports tracing down the identified quality issues to concrete findings in the analyzed artifacts.

The major outcomes of the Quamoco project can be summarized as domain-independent as well as domain-specific quality models, an approach for tailoring these models to the specific needs of an organization, a method for assessing the quality of software products, and, finally, tool support for the specification and application of quality models.

The sections below provide additional insights into the metamodel used for the specification of quality models and the approach taken for evaluating/assessing software product quality.

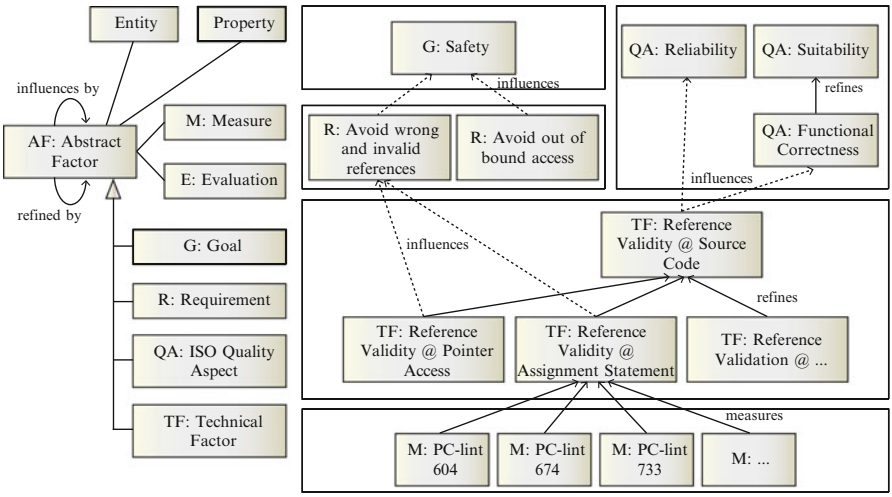

Fig. 6.9 The Quamoco metamodel for embedded systems, [cf. Mayr et al. (2012)]

6.4.1 The Quamoco Quality Metamodel

The underlying metamodel used for the specification of Quamoco quality models is quite generic, with the intent being sufficient expressiveness to model quality in diverse environments with different perceptions on quality (Kläs et al. 2010a) (meta-models). An overview exemplified by an excerpt of the domain-specific model for embedded systems (Mayr et al. 2012) can be seen in Fig. 6.9. The left side illustrates the structural components of the metamodel. The right side illustrates these structural components with excerpts from the quality model for embedded systems.

- **Abstract Factors**: An abstract factor can describe any concept (e.g., quality characteristic or variation factor) that is considered in a quality model. In the embedded systems model, four types of factors are distinguished:
  - Technical Factors define technical properties of entities that are independent of the programming language (such as having "valid pointer references").
  - ISO Quality Aspects describe ISO 25000 quality characteristics (such as "reliability" or "functional" correctness)
  - Requirements describe quality requirements impacted by technical factors (such as "avoiding wrong and invalid references")
  - Goals describe technical goals related to quality requirements (such as having "safe software systems")
- **Entity and Property**: A factor may further be described by an entity and a property it refers to. For example, the technical factor "reference validity of

assignment statements" addresses the entity "assignment statement" (as part of a source code) and the property "reference validity."

- **"Refined by" Relationship**: A factor can be refined into subfactors. However, a factor can only be refined by factors of the same type. For example, the quality aspect "suitability" is refined by the quality aspect "functional correctness." Refinement, abstract factor, entity, and property elements together implement the refinement concept (cf. Sect. 6.2.1).
- **"Influenced by" Relationship**: The relation states whether a factor positively or negatively influences another one and provides a textual justification for this relationship. However, impacts can only be modeled between factors of different types. For example, the technical factor "valid pointer references" influences the requirement "avoiding wrong and invalid references." The impact elements add the possibility to specify quality relationships as introduced in Sect. 6.2.1
- **Measure**: Factors can be quantified by measures. A measure is actually implemented by measurement instruments, which in turn are provided by measurement tools. For example, the technical factor "reference validity of assignment statements" is measured by the measure "PC-lint 604," which is provided by the "PC-lint" tool. In general, a measure may be implemented by different tools. For example, the popular measure "lines of source code" is implemented by all static code analysis tools. If subjective measurement data needs to be collected and no measurement tool is available, a manual measurement instrument (e.g., a certain questionnaire that needs to be filled in manually) can be assigned to the measure. Measure elements implement the quantification concept (cf. Sect. 6.2.1)
- **Evaluation**: Factors are evaluated by evaluation rules describing how to assess the factor on a certain evaluation scale. Factors that have no metrics assigned to them cannot be evaluated in the model. For example, the evaluation rule for the technical factor "reference validity of assignment statements" describes how to interpret and aggregate the measurement data provided by all measures assigned to the factor. Evaluation elements therefore provide implementation for both the general evaluation and the aggregation concepts as introduced in Sect. 6.2.1

The metamodel allows implementation of all conceptual structures needed to specify, measure/monitor, assess/control, and improve/manage quality (cf. Fig. 6.3): It also allows making a clear distinction between the quality characteristics of interest and all factors having an impact on them. By offering the opportunity to formulate arbitrary hierarchies of factors and relationships between different factors, the model allows the definition of different views/perspectives, such as combining a technical/developer view (requirements and technical goals) with a customer/manager (quality aspects) view in one comprehensive model. Furthermore, it allows for explicitly specifying the impact relationship between different factor hierarchies and expressing basic causal relationships.

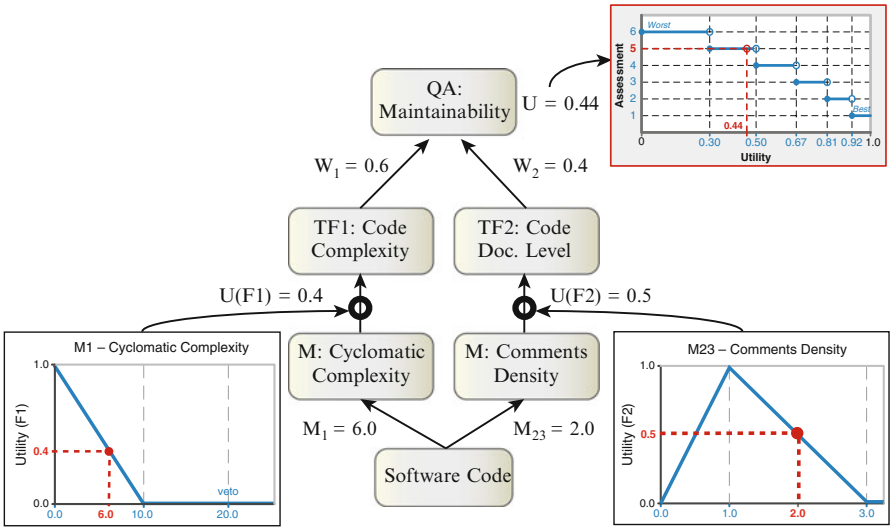

Fig. 6.10 Quality assessment example

6.4.2 *The Quamoco Quality Evaluation*

The main objective of a quantitative quality evaluation is to have an easy-to-understand, evidence-based (measurement-based), and reliable (repeatable) rating of the quality of software products. For that purpose, the measurement values need to be normalized and mapped to a uniform evaluation scale. Quamoco uses values between 0 and 1 as an internal evaluation scale, but supports different interpretation models, which can be defined based on the given context. The default interpretation model maps the evaluation results between 0 and 1 on a scale based upon the German school grade system. The mapping between evaluation results and grades was defined and checked for plausibility for Java-based systems using the evaluation results for more than 100 analyzed open source systems (Wagner et al. 2012).

The evaluation procedure employs multicriteria decision analysis (MCDA) techniques and is illustrated in Fig. 6.10. The procedure for determining evaluation rules (assuming that the remaining parts of a quality model are specified) is as follows (Trendowicz et al. 2009):

- **Weighting**: Quantify the importance of each factor relative to other factors of the same refinement and/or impact hierarchy
- **Scoring**: Collect measurement data for all leaf factors of the hierarchy
- **Evaluation**: Define so-called utility functions that map the measurement values to the evaluation scale with values between 0 and 1. Note that for most measures, some sort of normalization is required to allow defining that utility function
- **Aggregation**: Aggregate the evaluation results provided by the utility functions by computing a weighted sum (which is again a value between 0 and 1)

- **Interpretation**: Map the aggregated evaluation results to an interpretation scale (e.g., school grades)

The Quamoco models come with predefined evaluation rules calibrated on the basis of a survey on the most relevant quality characteristics, the experience of numerous software quality professionals involved in creating the model, and thresholds calculated on the basis of more than 100 software projects. Although the model results were confirmed by experts on five open source software products (Wagner et al. 2012), it is strongly recommended to perform additional calibration activities on organization-specific data before using the model.

6.5 Strategic Usage of Quality Models

The last section deals with how to support decision-making based on the outcomes from applying quality models and how this contributes to higher-level organizational goals and strategies. GQM$^+$Strategies®[2] (Basili et al. 2010) is a measurement planning and analysis approach that provides a framework and notation to help organizations develop/package their operational, measurable business goals, select strategies for implementing them, communicate these goals and strategies throughout the organization and translate these goals into lower-level goals and strategies down to the level of projects, assess the effectiveness of their strategies at all levels of the organization, and recognize the achievement of their business goals. The output of the GQM$^+$Strategies® approach is a detailed and comprehensive model that defines all the elements necessary for a measurement program. GQM$^+$Strategies® makes the business goals, strategies, and corresponding lower-level goals explicit.

In the past, a variety of approaches have been developed covering different aspects of linking activities related to IT services and software development to upper-level goals of an organization and demonstrating their business value, such as the Business Motivation Model (OMG 2010), Practical Software and Systems Measurement (USDoD 2003), Balanced Scorecards (Kaplan and Norton 1992), Information Technology Infrastructure Library (ITIL) (OGC 2002), Control Objectives for Information and Related Technology (COBIT)® (ISACA 2007), or the Sarbanes-Oxley Act (SOX 2002). The aim of GQM$^+$Strategies® is not to replace these approaches, but rather to close the existing gaps with respect to linking goals, their implementation, and the measurement data needed to evaluate goal attainment.

Figure 6.11 illustrates the basic concepts of the approach. The left side describes a hierarchy of organizational goals and strategies. Organizational goals define a target state the organization wants to achieve within a given time frame (e.g.,

[2] Registered trademark of the Fraunhofer Institute for Experimental Software Engineering, Germany and the Fraunhofer USA Center for Experimental Software Engineering, Maryland.

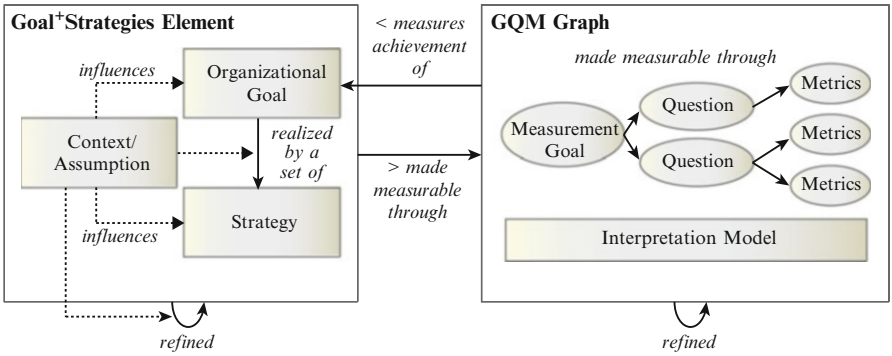

Fig. 6.11 The GQM+Strategies® grid metamodel

improved customer satisfaction or reduced rework costs). Strategies are possible approaches for achieving a goal within the environment of the organization. Context factors and assumptions provide the rationale for the refinement hierarchy. Context factors represent all kinds of factors the organization knows for sure, whereas assumptions are estimated unknowns, that is, what is believed to be true but needs to be reevaluated over time.

GQM graphs define how to measure whether a goal has been accomplished and whether a strategy has been successful. Following the classic GQM approach (Basili et al. 1994a), goals are broken down into concrete metrics. Interpretation models are used for objectively evaluating goals and strategies.

Figure 6.12 and Table 6.4 illustrate excerpts of an example GQM+Strategies® model focusing on the goals and strategies hierarchy and highlighting some measurement data that is collected for evaluating the achievement of organizational goals. On the lowest level, one strategy (DS-S) of the highlighted branch may actually be to build and introduce software product quality models for, e.g., software reliability. The use of these quality models at different quality gates of the software development process will in turn help to decrease the number of defects that slip to later stages of the process (DS-G) by finding potential reliability issues as early as possible. Less defect slippage is related to improving the quality assurance activities of organization X (PR-S), which is a strategy for improving the reliability of the IT products of organization X (PR-G). Improved IT products (NC-S1) will in turn attract more customers to use the IT-based services of company X (NC-G).

The entire model provides an organization with a mechanism for not only defining measurement consistent with larger, upper-level organizational concerns, but also for interpreting and rolling up the resulting measurement data at each level. Having this chain of arguments also supports an organization in demonstrating the values of software-related improvement initiatives, such as the systematic usage of software product quality models. The impact of these models can be evaluated directly in terms of an organization's higher-level goals and make the benefits measurable for it. More industry-related lessons learned from making strategic use

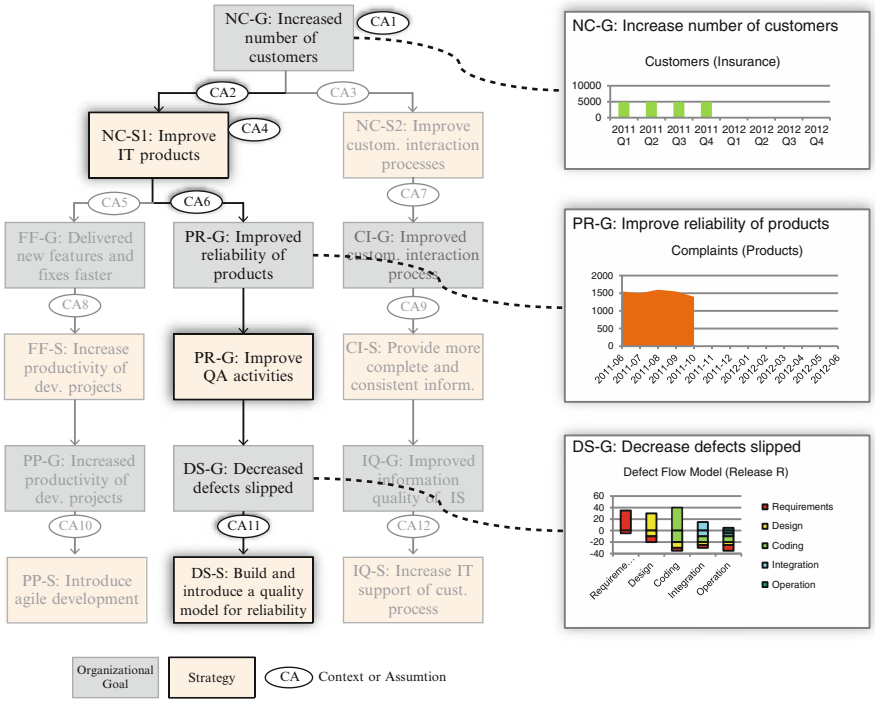

Fig. 6.12 Example GQM⁺Strategies® grid

Table 6.4 Overview of context and assumption

ID	Type	Description
CA1	Context	Company X provides banking and insurance services. X has a lot of customers in the banking area, but only few in the insurance area
CA2	Assumption	The quality of the IT products has to be improved
CA3	Assumption	The quality of the customer interaction processes has to be improved
CA4	Context	The services of X are built upon an Enterprise information system (IS) that is composed of different software components
CA5	Context	Customers complain that it takes too long to deliver new features and to fix existing bugs
CA6	Context	Customers complain that the IT products are not reliable
CA7	Context	Customers complain about issues related to customer interaction
CA8	Assumption	The delay of existing projects is mainly responsible for the inability to deliver new features and bug fixes faster
CA9	Context	Customers complain about inconsistent and incomplete information during their interaction with company X
CA10	Context	According to the experience from the recently run pilot project, agile development principles will be able to speed up software development
CA11	Context	According to the analysis of the defect data, too many defects appear in the design and coding stage
CA12	Context	Not all services of X are completely IT supported; some have to be provided manually, which decreases information quality

of quality models employing the GQM+Strategies® approach can be found in Basili et al. (2013).

6.6 Conclusions and Future Work

This chapter gave an overview of the challenges and potential solutions for systematically managing the quality of software products. Controlling the quality of all artifacts created during the software development process is one crucial task of professional project management (PMI 2008). Quality models support an organization in general and project managers in particular in objectively evaluating and assessing software product quality through the use of measurement. Knowledge and experience regarding critical quality characteristics and indicators for measuring and evaluating these characteristics are captured in these models.

The chapter highlighted four challenges when dealing with quality models in practice and proposed solutions developed in recent years. First, there is no universal quality model that can be applied everywhere. A variety of quality models exists, and mechanisms such as the proposed classification scheme and quality model landscapes are needed to identify the "right" model based on a clear picture of the goals that should be obtained from using the model.

Second, it is essential to tailor quality models to company specifics. Existing standards are often too generic and hard to fully implement in an organization. A structured process, such as the six-step process proposed, is needed to develop custom-tailored quality models. This also allows for collecting the measurement data needed and to focus data analysis and interpretation on the quality characteristics of interest.

Third, in practice, it is an effort-consuming process to specify and apply quality models because no proven standard techniques, methods, and tools are available. The Quamoco approach described above provides a well-defined metamodel for the specification of quality models and comes with tool-supported, domain-specific models, which can be customized to the specific needs of an organization.

Fourth, in order to create quality models that are sustainably implemented in an organization, the link and contribution to organizational goals need to be clarified. As illustrated by the GQM+Strategies® approach, the data provided from applying software product quality models can be used directly for guiding improvement actions and decision-making.

In the future, software projects will be faced with new challenges that need to be mastered from a practitioner's point of view if an organization wants to provide products with the right level of quality in order to defend and further expand its position on the market. In order to manage future projects successfully, processes and quality assurance mechanisms must handle ever shorter business and technology life cycles and must permit flexible adaptation. Introducing agile development principles is one potential approach allowing for more flexibility (as discussed in Chap. 11). Software products and systems are increasingly being developed in a

distributed manner in heterogeneous environments. Chapters 9, 10, and 12 highlight some further challenges and solution approaches for managing global software and IT projects. This is particularly true for cyberphysical systems, where organizations from different domains work together on an integrated solution, each with its own special requirements regarding the integration of different processes and quality management mechanisms. As a consequence, software product quality models must be easy to adapt to new quality requirements on the one side. On the other side, they must be able to address very heterogeneous quality requirements from different domains, which probably use different development processes, and to integrate all these aspects into a comprehensive model. From a researcher's point of view, one major challenge lies in providing empirically proven quality models that can be successfully applied in dedicated domains with known effects. Future work will focus on coping with these aspects.

Acknowledgments The research leading to these results was partially supported by the ARTEMIS Joint Undertaking under grant agreement no. 269335, the research project Quamoco (grant 01IS08023), and from the German Federal Ministry of Education and Research (BMBF).

References

Avizienis A, Laprie JC, Randell B (2001) Fundamental concepts of dependability

Basili V, Caldiera G, Rombach D (1994a) Goal, question metric paradigm. Encyc Softw Eng 1:528–532 (John Wiley and Sons)

Basili V, Caldiera G, Rombach D (1994b) The experience factory. Encyc Softw Eng 1:469–476 (John Wiley and Sons)

Basili V, Heidrich J, Lindvall M, Münch J, Regardie M, Rombach D, Seaman C, Trendowicz A (2010) Linking software development and business strategy through measurement. IEEE Comput 43(4):57–65

Basili V, Lampasona C, Ocampo A (2013) Aligning corporate and IT goals and strategies in the oil and gas industry. In: Proceedings of the 14th international conference on product-focused software process improvement, lecture notes in computer science, vol 7983. Springer, New York, pp 184–198

Boehm BW (1978) Characteristics of software quality. North-Holland, Amsterdam

Cavano JP, McCall JA (1978) A framework for the measurement of software quality. In: Proceedings of the software quality assurance workshop on functional and performance issues. ACM, New York, pp 133–139

Differding C (2001) Reuse of measurement plans based on process and quality models. In: Proceeding of 3rd international workshop on advances in learning software organizations (LSO). Springer, pp 207–221

Dörr J, Trendowicz A, Kolb R, Punter T, Kerkow D, König T, Olsson T (2004) Quality models for non-functional requirements. Fraunhofer IESE Report No. 010-04/E

Dromey GR (1998) Software product quality: theory, model and practice. Griffith University, Brisbane, Australia

ECSS-Q-30A (1996) Space product assurance: dependability

IEC 61508-1 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems

IEEE 1061 (1998) Software quality metrics methodology

ISACA (2007) Control objectives for information and related technology (CoBIT®). Retrieved 04 12 2007, from www.isaca.org
ISO 8402 (1995) Quality management and quality assurance – vocabulary
ISO/IEC 14598-1 (1999) Information technology software product evaluation
ISO/IEC 15939 (2007) Systems and software engineering measurement process
ISO/IEC 25000-1 (2005) Software product quality requirements and evaluation (SQuaRE)Guide to SQuaRE
ISO/IEC 25010 (2011) SQuaRE system and software quality models
ISO/IEC 25021 (2012) SQuaRE quality measure elements
ISO/IEC 25040 (2011) SQuaRE evaluation process
ISO/IEC 9126-1 (2001) Software engineering product quality - part 1
Kaplan R, Norton D (1992) The balanced scorecard - measures that drive performance. Harv Bus Rev 71
Kitchenham BA, Linkman S, Pasquini A, Nanni V (1997) The SQUID approach to defining a quality model. Softw Qual Control 6(3):211–233
Kläs M, Münch J (2008) Balancing upfront definition and customization of quality models. In: Proceedings of the workshop on software quality modeling and assessment (SQMB 2008), Munich, Germany, pp 26–30
Kläs M, Heidrich J, Münch J, Trendowicz A (2009) CQML Scheme: a classification scheme for comprehensive quality model landscapes. In: Proceedings of the 35th EUROMICRO conference (SEAA 2009). IEEE Computer Society, pp 243–250
Kläs M, Lampasona C, Nunnenmacher S, Wagner S, Herrmannsdörfer M, Lochmann K (2010a) How to evaluate meta-models for software quality? In: Proceedings of the joint international conferences on software measurement. IWSM/MetriKon/Mensura, Shaker, pp 443–462
Kläs M, Elberzhager F, Münch J, Hartjes K, von Graevemeyer O (2010b) Transparent combination of expert and measurement data for defect prediction – an industrial case study. In: Proceedings of the 32nd international conference on software engineering (ICSE 2010), Cape Town, South Africa, pp 119–128
Lampasona C, Heidrich J, Basili V, Ocampo A (2012) Software quality modeling experiences at an oil company. In: Proceedings of the 6th international conference on empirical software engineering and measurement (ESEM), 20–21, pp 243–246
Mayr A, Plösch R, Kläs M, Lampasona C, Saft M (2012) A Comprehensive code-based quality model for embedded systems - systematic development and validation by industrial projects. In: Proceedings of the 23rd international symposium on software reliability engineering (ISSRE 2012), Dallas, TX
MISRA Report 5 (1995) Software metrics office of government commerce (2002). The IT Infrastructure Library (ITIL) Service Delivery, The Stationary Office London
Object Management Group (2010) The business motivation model (BMM) V. 1.1. Retrieved 06 08 2010, from www.omg.org
Office of Government Commerce (OGC) (2002) The IT infrastructure library (ITIL) service delivery. The Stationary Office, London
Petersson H, Thelin T, Runeson P, Wohlin C (2004) Capture–recapture in software inspections after 10 years research—theory, evaluation and application. J Syst Softw 72(2):249–264
Project Management Institute (2008) A guide to the project management body of knowledge (PMBOK® Guide), 4th edn. Project Management Institute
Sarbanes-Oxley Act (2002) Public Law No. 107-204, 116 Stat. 745, Codified in sections of 11, 15, 18, 28, and 29 in United States Code, July 30
Stasko J (2013) Sun burst. Retrieved 29 01 2013, from www.cc.gatech.edu/gvu/ii/sunburst
Trendowicz A, Heidrich J, Münch J, Ishigai Y, Yokoyama K, Kikuchi N (2006) Development of a hybrid cost estimation model in an iterative manner. In: Proceedings of the 28th international conference on software engineering (ICSE 2006), Shanghai, China, pp 331–340

Trendowicz A, Kläs M, Lampasona C, Münch J, Körner C, Saft M (2009) Model-based product quality evaluation with multi-criteria decision analysis. In: Proceedings of the joint international conferences on software measurement (IWSM/MetriKon/Mensura), Shaker, pp 3–20

United Kingdom Ministry of Defense (1997) Def Stan 00-55 requirements for safety related software in defense equipment

US Department of Defense and US Army (2003) Practical software and systems measurement: a foundation for objective project management, v. 4.0c, from www.psmsc.com

Wagner S, Lochmann K, Winter S, Göb A, Kläs M, Nunnenmacher S (2010a) Software quality in practice survey results. Retrieved 03 06 2014, from http://mediatum.ub.tum.de/doc/1110601/1110601.pdf

Wagner S, Broy M, Deißenböck F, Kläs M, Liggesmeyer P, Münch J, Streit J (2010b) Softwarequalitätsmodelle. Praxisempfehlungen und Forschungsagenda, Informatik Spektrum 33(1):37–44 (Springer)

Wagner S, Lochmann K, Heinemann L, Kläs M, Trendowicz A, Plösch R, Seidl A, Goeb A, Streit J (2012) The Quamoco product quality modeling and assessment approach. In: Proceedings of the 34th international conference on software engineering (ICSE 2012), Zurich, Switzerland, pp 1133–1142

Biography Jens Heidrich graduated from the University of Kaiserslautern, Germany, with a Diploma degree in Computer Science and received his doctoral degree from the same university in 2008. He is head of the Process Management division at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany, and a lecturer at the University of Kaiserslautern, Germany. He is a member of the German Informatics Society and part of the managing committee of the section "Software Measurement."

Dieter Rombach studied mathematics and computer science at the University of Karlsruhe, Germany, and obtained his Ph.D. in Computer Science from the University of Kaiserslautern, Germany in 1984. Since 1992, he has held the Software Engineering Chair in the Department of Computer Science at the University of Kaiserslautern. In addition, he is the founding and executive director of the Fraunhofer Institute for Experimental Software Engineering IESE in Kaiserslautern, Germany.

Michael Kläs graduated from the University of Kaiserslautern, Germany, with a German Diploma degree in Computer Science in 2005 and started working at the Fraunhofer Institute for Experimental Software Engineering IESE thereafter. He works on subjects concerning goal-oriented measurement, modeling and assessing quality, as well as defect prediction and cost estimation. His current research interests focus on early quality prediction and aligning large-scale technology evaluation endeavors.