

An Integrated Approach to Quality Modelling

Stefan Wagner and Florian Deissenboeck
Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
{wagnerst,deissenb}@in.tum.de

Abstract

Software quality is described by various views using different attributes and models. All these types of software quality have their own benefits and applications. However, the isolated solutions do not allow an integrated view on software quality. This renders a comprehensive analysis of software difficult and causes overlaps and inconsistencies in the models. Therefore, this paper proposes an integrated approach to quality modelling. The approach is based on the idea to use an explicit meta-model that provides the means to develop a base quality model. This base model contains the relevant relationships w.r.t. quality. Furthermore, so-called purpose models are derived by quantifying from this basis in order to aid specific tasks in quality management. The approach is illustrated with a proven example of a meta-model and derived quality model for maintainability.

1. Introduction

In his influential article on product quality in different domains, Garvin concluded that “Quality is a complex and multifaceted concept” [5]. Accordingly, the software engineering community produced a plethora of methods to deal with the various quality facets (examples are given in the next section). However, these contributions, though highly valuable, are very specialised for specific facets and thereby relatively *isolated*. We claim that the lack of a systematic concept to integrate the existing quality approaches renders a comprehensive analysis of software difficult and causes overlaps as well as inconsistencies in definitions of quality.

We are well aware that there are good reasons to tackle different quality issues with different methods and do not aim at depriving the individual approaches of their power by forcing them in a uniform framework. However, the quality measures applied in particular contexts should be coordinated by a systematic concept that enables us to rea-

son about the different quality aspects and their interdependencies. This is especially important for the ultimate goal of a truly economically justified practice of quality engineering. Only if the different quality measures adhere to a generic quality concept, we will be able to compare them and thereby to determine the economic costs and benefits of specific quality measures (or a whole quality program) on a quantitative scale. Such a concept is indispensable for discussing quality tradeoffs posed by conflicting quality goals as they are frequently encountered for attributes like performance and maintainability.

Problem. There is a rich set of models and techniques for software product quality that support engineers in dealing with a multitude of quality related issues. However, we still lack an integrated approach that enables us to treat all these equally important, but currently isolated quality aspects in a uniform way.

Contribution. We present a classification that describes the different dimensions of quality that are typically addressed by existing approaches (Sec. 3). Deriving from this, we introduce a new, three-layered concept to integrate the different approaches to quality in a uniform manner (Sec. 4). This concept is based on cost factors and a common *meta-model* that enables us to precisely describe generic quality characteristics in a so-called *base model*. This base model can be extended with situation- and application-specific *purpose models*. This approach allows us to capture quality characteristics that are common to multiple contexts while still being open to extensions needed in specific situations. Thereby the quality model can be customised to provide different views on quality, pursue varying quality goals and support multiple quality assurance techniques applied in the respective development phases. This makes it possible to treat several quality aspects in an uniform way and thereby enables us to identify inconsistencies and redundancies caused by the isolated methods

currently used. To illustrate our approach we present an exemplary quality meta-model that we successfully applied in different industrial and academical contexts (Sec. 5).

2. Quality views & models

This section gives an overview of the views on quality and models that were developed to describe different facets of quality.

2.1. Quality views

In [5] Garvin gave a definition of five different approaches (views) to product quality that also gained attention in the software engineering community [7]. He distinguishes

1. the *transcendent view* that defines quality as something that can be recognised but not defined,
2. the *product view* that defines quality by specifying the product characteristics that contribute to quality,
3. the *user view*, that regards quality as subjective attribute that is “in the eye of the beholder”,
4. the *manufacturing view*, that defines quality as conformance to specification, and
5. the *value-based view* that regards quality as inherently connected to the costs to achieve it.

Existing approaches to software product quality usually take one (sometimes more) of these views. For example, usability and performance aspects are typically addressed from a user-based view, metric-based approaches take a product-view and process-related methods like CMM use the manufacturing view. As these views are necessary to discuss quality in a differentiated manner, an integrated approach to software product quality must be able to reflect them adequately.

2.2. Quality models

A number of *quality models* has been developed to describe selected facets of quality in a systematic way. We are not able to present a comprehensive survey in this paper but highlight some examples to show the breadth of existing models.

Examples are the hierarchically structured models that were first used by Boehm et al. [2] and McCall et al. [4] and later adapted by the ISO standard 9126 [6]. These models define quality by breaking it down into the well-known quality criteria like *functionality*, *reliability*, *usability*, *efficiency*, *maintainability*, and *portability* which in turn are

broken down into more specific sub-criteria. Up to now these models have failed to establish a broadly accepted definition of quality because they mix criteria from different dimensions and fail to describe characteristics precisely enough to be actually assessable.

A completely different type of quality models are reliability growth models for software as described e. g. in [8]. Such models are successfully used to predict the future failure behaviour of a software based on test data. However, such models are applied in an isolated manner and are not integrated with other models that describe related quality characteristics.

For usability, there is also a variety of guidelines and models. Seffah et al. [9], for example, developed a consolidated model of usability that contains aspects of safety as well as conformance to user expectations. Safety, however, is sometimes considered as a quality attribute in its own right. Also the reliability of the software has an effect on the usability. The usability models known to us are not capable of making these interdependencies explicit.

Tian [11] acknowledges the abundances of quality models for the *correctness* quality aspect of software and presents a goal-driven method for selecting the appropriate model in a specific situation. However, his work focuses on correctness and he does not fully explain how different models can be integrated.

Basili, Donzelli, and Asgari [1] propose a unified model of dependability in order to elicit quality requirements. Their focus is on user-related requirements and how to collect, visualise and assess them. They do not derive any assessing or constructive model from that.

2.3. Integration

Though there usually is an emphasis on certain quality criteria, e. g. safety in the automotive domain, software development organisations do not focus on a single quality aspect but need to cover a broad quality spectrum. Today they do so by applying different quality models in isolation and thereby create a situation that makes it hard to recognise overlaps and inconsistencies in the various models. Moreover, this situation does not allow them to discuss quality tradeoffs in a systematic way. For example it is almost impossible today to estimate what impact a newly introduced guideline for *maintainability* will have on the *testability* or even the *performance* of a system. A similar example is the dependency between *usability*, *safety* and *reliability* presented above.

3. Dimensions of quality models

Before we propose an integrated approach to quality modelling, we need to identify along which dimensions the

large variety of models described in Sec. 2 differ. The models often have different intents and all concentrate on different quality views. We identify six different dimensions that form the basis of our integrated approach.

Purpose. We can distinguish three main types of purposes for quality models:

- *Constructive*: We see constructive models as explanations of relationships between constructive actions and some aspects of software quality. For example, the use of specific programming language constructs might influence the reliability of the system in a certain way. These relationships can be coarse-grained but help to understand and to choose from the possibilities during development.
- *Predictive*: The predictive models help to plan the future development of some quality aspects and hence are used to plan the quality assurance.
- *Assessing*: The assessing models allow to estimate the current state of the software to control the quality assurance.

View. We discussed in Sec. 2 different important views on quality. Different quality models support different views. For example, many models are product-based, i.e. they measure metrics of the software and use them to assess the quality. Other models are value-based and quantify the relationships using monetary units. In the following we use all of the views described in Sec. 2 with the exception of the *transcendental* approach because it is not suitable as a basis for a quality model in our sense. Also note that the main focus will be the value-based view.

Attribute. The quality attributes as defined by the ISO (cf. Sec. 2) such as reliability or maintainability currently constitute the dominant decomposition of quality. Commonly, each different quality attribute has its own set of quality models. Generally, this makes sense because the attributes have proven to be useful and intuitive aspects of software quality. However, the attributes are not independent of each other. For example, the reliability of a software system influences its usability or the portability can have an influence on the maintainability. Hence, there is an overlap that needs to be considered. Moreover, there can be contradictions because tradeoffs are involved. An exemplary tradeoff can be found in Microsoft Windows NT. From version 4.0 on the initially strict modularisation was given up and thereby portability and reliability were sacrificed for performance [10].

Phase. Several quality models concentrate on specific phases of the software life cycle where they can be used or they predict for. This can coincide with a quality attribute. For example, maintainability models are typically used to predict maintenance costs or efforts. Reliability growth models on the other hand are often in use to predict the test efforts as well as the failure rate in operation. Rather abstract models can be used in early phases of development where only coarse-grained estimates are available whereas other models depend on detailed results and specific development artefacts.

Technique. It is also often the case that a quality model focuses on the effects of a specific kind of defect-detection technique. For example, there are several models that are specific to inspections. Also most reliability growth models are only applicable to system tests that follow an operational profile.

Abstractness. Finally, it is of importance to what level of detail the quality model is developed. Tian [11] distinguishes *general* and *product-specific* quality models depending on whether the model is valid for the whole (or parts of the) industry or it is tailored to a specific project or product. This is similar to upper(-level) ontologies vs. lower(-level) ontologies. The upper ontologies contain higher-level, more abstract concepts whereas the lower ontologies are related to a specific domain.

4. An integrated approach

The various dimension of Sec. 3 show the complexity of software quality and hence also the complexity of modelling it. We described in Sec. 2 the variety of existing quality models that cover various parts of those dimensions. Many of them are highly useful in those areas. However, for a comprehensive quality management in software development, we need to combine those isolated models to get a more complete picture of software quality. Furthermore, we need to reduce the elaborate work to develop specific quality models by maximising reuse. The following proposes a three-layer approach for quality modelling and a corresponding method to quality management. In essence, it consists of the idea to have a single model – the *base model* that contains all relevant relationships and derived *purpose models* that answer specific questions based on the base model. Both kinds of models conform to an explicit meta-model. For illustration purposes, we use the running example of building web shops in the following.

4.1. Cost factor components

We use the cost factor components as the primary means of decomposing the quality model. To use *cost factors* for

that has two reasons: (1) monetary value is the only metric everything can be converted to and (2) generating monetary value is the aim of all commercial software projects. To be able to judge the value, we look at the opposing side: the costs. The other parts of the quality model can be identified by asking what does influence the cost factors. Software costs are created by different stakeholders in different ways, e. g. development activities and development hardware create costs during development, user hardware and usage activities create cost during the usage of a product and its operation creates cost in the form of server hardware and support personnel.

Our aim in economical situations is to minimise these costs. Hence, we are interested in facts that are related to these cost factors. Moreover, we are able to find mappings of quality attributes, as discussed in Sec. 2, to activities. For example, usability is concerned with all the usage activities whereas maintainability is driven by the development activities. Hence, if we choose the quality attributes we are interested in, we will know which activities are involved and thereby are able to select the parts of the quality model that have an impact on those activities.

4.2. Three layers

Our proposed solution to integrated quality modelling involves the three layers (1) *meta-model*, (2) *base model* and (3) *purpose models* that are depicted in Fig. 1.

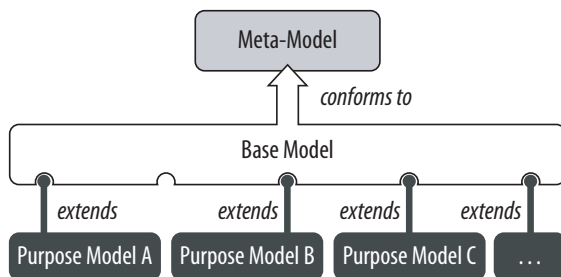


Figure 1. The integrated model

The *meta-model* constitutes the construction plan for all the other models. It defines the basic elements that can be used to build a quality model. We give in Sec. 5 an example how such a meta-model could look like but the general approach is independent of the concrete meta-model used. It is only important that the meta-model allows to express the important properties of the system, process, and environment together with their interrelations. In summary, we aim at modelling everything that has an influence on the quality in order to allow a comprehensive analysis.

The *base model* uses the framework defined by the meta-model to define the basic quality-related relationships of the system and process under consideration. Essentially, the

influences of various facts among each other are made explicit. In particular, we need the impacts of these facts on the cost factors that are important in our context. Cost factors are parts of the development or use that are the origin of incurring costs. In software development, these are mainly human activities. However, this also includes hardware or energy consumption. Because in principle, we are interested in evaluating the costs, we use cost factors as a major means for the decomposition of quality. The parts of our quality model that influence the cost factors form the *cost factor components* that are used to compose the base model. In our web shop example, we are probably very user-oriented. Hence, we might see the main cost-factor in the activities of the user. Therefore, we would model all influences on the usage activities.

Furthermore, the base model covers the dimensions *abstractness* of quality models identified in Sec. 3. It should at least contain an upper-level part that is general, i. e. it can be reused over different domains. However, it is often useful to detail the general parts to a product-specific level. This level contains all the refined relationships for a specific environment. In our example, we want the user to have control over the web shop and hence allow her/him to return to the home page with one click regardless of what subpage is currently active. In terms of the quality model, this means that having this possibility has an influence on the usage activity. This is a rather specific relationship for the web shop. However, the general concept is more universal: The user should always be able to get back to the starting point with one step. This could be included in the upper-level model and be instantiated in other domains.

Finally, the *purpose models* are derived from different levels of abstractness of the base model to serve different purposes, i. e. assessment or prediction. Those models have four additional dimensions that correspond to the dimensions described in Sec. 3: *attribute*, *phase*, *technique* and *view*.

Many of the models described in Sec. 2.2 can be seen as purpose models although they are not based on a base model and they are not explicitly defined in all the four dimensions. For example, a reliability growth model uses the attribute *reliability*, works in the phases *system test* and *operation*, and relies on the technique *operational testing*. The view, however, is not so clear. One could argue that it is a user-based view because they often only consider failures, i. e. defects that are visible to the user. For the web shop example, we could derive review guidelines based on a very detailed level by identifying the positive and negative influences of aspects of the code and architecture on the usage activity.

4.3. Method

Based on the three-layer approach explained above, we propose a method for quality modelling and management that allows an interplay of different quality models for different purposes. The main idea is that the base model contains several components on different levels of abstraction with the quality-related knowledge available. The purpose models quantify the relationships defined in the base model in order to aid planning and assessment.

Quality goals. Using goals is a common approach in requirements engineering, e. g. [12]. Goals are also used for quality requirements and hence define the desired quality of the system on a high level. The goal definitions give us the base for the further quality modelling. It shows which quality goals are more important than others and on what level of detail they need to be defined and analysed. This often uses the decomposition of quality attributes that have established corresponding metrics. For view that is already more quality-oriented, we can use the UMD method of Basili, Donzelli, and Asgari [1] to describe the dependability requirements. The web shop might have the goal that it does not take longer than 2 minutes for an average user to buy a specific product.

Base model building. Having analysed which quality areas are important for our specific needs and on what level of abstraction we analyse them, we can build cost factor components that form the base model. It is desirable to have an upper base model available in a company or domain that is then detailed for specific products. This way, many reuse opportunities can be exploited. The base model encodes all the domain knowledge available related to quality w.r.t. the product, the process, and the environment. The identified relationships form the base of the following steps. However, the base model itself has benefits. It is constructive, i. e. it gives (more or less concrete) instructions on how to develop a “high-quality” software system. Hence, it aids the continuous learning of all the developers involved.

The first activity in building the base model is to identify the cost factors – especially the activities – that are of interest. Activities are the main source of costs in software development. Hence, we want to model all the influences on these cost factors as precise as possible. Hence, the cost factors give us a natural decomposition of the quality model, the components. We choose which cost factors are considered and build the components accordingly. This avoids overlaps that would be developed when structuring using the quality attributes directly.

Derivation of purpose models. The base model describes all important relationships that we need to consider

to model quality. In order to support the planning and realisation of quality assurance we need assessing and predicting models. Hence, those models follow a specific purpose for which they are built. These purpose models are derived from the base model by quantifying the relations modelled in the base model. This involves the following steps:

1. Choose the necessary relationships on the necessary level of detail.
2. Choose the values on the four dimensions attribute, view, technique, and phase.
3. Quantify the relationships adequately in order to make assessments or predictions

The purpose models can then be used to analyse the quality goals. For example, a purpose model could be a model that analyses the reliability of the software in the field. For this, we choose a level of abstraction that seems reasonable, quantify the influences of the system and process properties that influence the activity “usage of the software”, and collect the data we need to measure the properties. In this way, we derived a reliability model with a well-founded basis. Coming back to the web shop example, we identified a quality goal above. Now, we need to derive a purpose model in order to analyse this goal. We have encoded in the model all the influences on the usage activity of the user. In the specific case we look at the specific activity “buy product”. These influences are now quantified either using empirical data, other experience or expert judgement. For example, the fact that the user is able to return to the home page with one click improves the duration of the “buy product” activity by 10%.

5. A quality meta-model

Based on our experience with modelling *maintainability* [3] we developed a novel two-dimensional quality meta-model. Stepwise, this model was adopted to describe quality attributes as different as usability [14] and dependability [13] and thereby became generic enough to serve as a formal specification of the base model and the purpose models presented above.

The model is based on the general idea of hierarchical models, i. e. breaking down fuzzy criteria like *maintainability* into sub-criteria that are tangible enough to be assessed directly. However, we found that most existing quality models promiscuously mix quality characteristics with activities performed on (or with the system). Examples are attributes like *modifiability* and *comprehensibility*. Though adjectives are used, these attributes actually describe the activities *modification* and *comprehension* which in turn are influenced by product characteristics like *modularisation* [3].

While this distinction may look captious at first sight, we claim that it is of paramount importance for discussing quality because the performed activities ultimately define the development costs. Therefore, a quality meta-model is needed that explicitly describes this distinction and thereby enables us to reason about interdependencies of quality characteristics and cost factors. Consequently, quality models that confirm to our meta-model contain *two* tree-like structures, one for the system characteristics and one for the cost factors, and an explicit description of *impacts* that puts both trees into relation.

An example of an instance of the meta-model for the quality attribute *maintainability* is shown in Fig. 2. The matrix points out what activities are affected by which characteristics and allows to aggregate results from the atomic level onto higher levels in both trees. So, one can determine that concept location is affected by the names of identifiers and the presence of a debugger. Vice versa, cloned code has an impact on two maintenance activities.

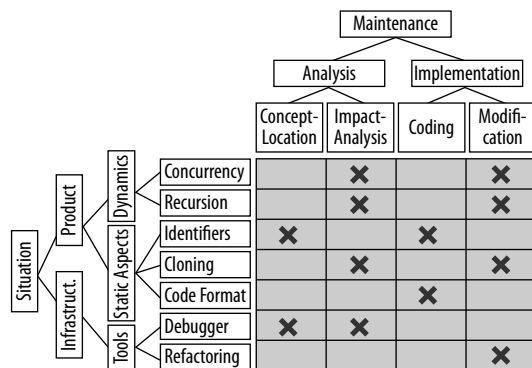


Figure 2. Maintainability matrix

6. Conclusions

The definitions and models of software quality are divided into several camps that do not fit straightforwardly. However, an integrated view and model of software quality is desirable because of the interrelations that exists between these different attributes. This is needed for a comprehensive analysis of software quality and avoids overlaps of the models. Therefore, we propose an integrated approach to quality modelling.

The approach is based on building a base quality model that relies on an explicit meta-model. This ensures uniform modelling of all the quality-related aspects. The base model contains in a simple form all the relevant relationships of the system, process, and environment. In particular, the relationship to cost factors is important because they are of most interest economically. Then purpose models are derived by quantifying the relationships. The purpose models serve as tools for supporting decisions during the project.

We also present a possible meta-model for quality that proved to be appropriate for modelling maintainability [3] and usability [14]. We are aware that there are many parts of this integrated approach that need to be worked out in more detail. However, we see such an approach as a viable possibility. From a different perspective, we could argue that this is the approach that is actually done today without making the meta-model and the base model explicit. Most models developed are directly purpose models. We propose to make those implicit models explicit and thereby gain benefits such as consistency and reuse.

References

- [1] V. Basili, P. Donzelli, and S. Asgari. A unified model of dependability: Capturing dependability in context. *IEEE Software*, 21(6):19–25, 2004.
- [2] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merritt. *Characteristics of Software Quality*. North-Holland, 1978.
- [3] M. Broy, F. Deissenboeck, and M. Pizka. Demystifying maintainability. In *Proc. 4th Workshop on Software Quality (4-WoSQ)*, pages 21–26. ACM Press, 2006.
- [4] J. P. Cavano and J. A. McCall. A framework for the measurement of software quality. In *Proc. Software Quality Assurance Workshop on Functional and Performance Issues*, pages 133–139, 1978.
- [5] D. A. Garvin. What does product quality really mean? *MIT Sloan Management Review*, 26(1):25–43, 1984.
- [6] ISO. Software engineering – product quality – part 1: Quality model, 2001.
- [7] B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. *IEEE Software*, 13(1):12–21, 1996.
- [8] J. D. Musa. *Software Reliability Engineering: More Reliable Software Faster and Cheaper*. AuthorHouse, 2004.
- [9] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Control*, 14(2):159–178, 2006.
- [10] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2nd edition, 2001.
- [11] J. Tian. Quality-evaluation models and measurements. *IEEE Software*, 21(3):84–91, 2004.
- [12] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. International Symposium on Requirements Engineering*. IEEE CS Press, 2001.
- [13] S. Wagner. A meta-model-based dependability model. In *Proc. Dagstuhl-Seminar Software Dependability Engineering*, 2007. To appear.
- [14] S. Winter, S. Wagner, and F. Deissenboeck. A comprehensive model of usability. In *Proc. Engineering Interactive Systems 2007 (EIS '07)*. Springer, 2007. To appear.