

# Software Quality Models: Purposes, Usage Scenarios and Requirements

Florian Deissenboeck, Elmar Juergens, Klaus Lochmann, and Stefan Wagner  
Fakultät für Informatik  
Technische Universität München  
Boltzmannstr. 3, 85748 Garching b. München, Germany  
{deissenb,juergens,lochmann,wagnerst}@in.tum.de

## Abstract

*Software quality models are a well-accepted means to support quality management of software systems. Over the last 30 years, a multitude of quality models have been proposed and applied with varying degrees of success. Despite successes and standardization efforts, quality models are still being criticized, as their application in practice exhibits various problems. To some extent, this criticism is caused by an unclear definition of what quality models are and which purposes they serve. Beyond this, there is a lack of explicitly stated requirements for quality models with respect to their intended mode of application. To remedy this, this paper describes purposes and usage scenarios of quality models and, based on the literature and experiences from the authors, collects critique of existing models. From this, general requirements for quality models are derived. The requirements can be used to support the evaluation of existing quality models for a given context or to guide further quality model development.*

## 1. Introduction

Research on software quality is as old as software research itself. As in other engineering and sciences disciplines, one approach to understand and control an issue is the use of models. Therefore, quality models have become a well-accepted means to describe and manage software quality. Beginning with hierarchical models proposed by Boehm et al. [2], over the last 30 years, a variety of quality models has been proposed, some of which have been standardized. Many of these models are used, for example to aid the specification of quality requirements, to assess existing systems or to predict the quality of a system in the field.

**Research Problem** Despite the progress made with software quality models, there are still issues that prevent them

from becoming widely adopted in practice. Practitioners, in particular, have been disappointed because quality models do not live up to expectations. It often remains unclear how quality models can be operationalized in practice to define, assess and predict quality. An underlying problem is the overwhelming number of diverse models that have been proposed by different research communities for different purposes without using a uniform terminology.

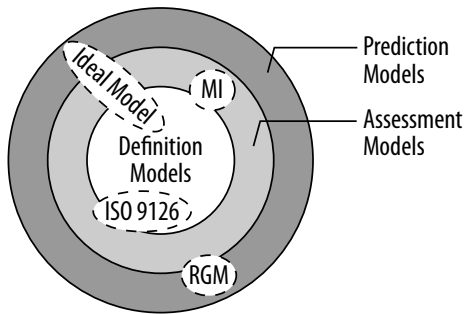
**Contribution** To clarify the situation regarding the plethora of existing quality models, we propose a simple three-level scheme to classify quality models w.r.t. to their intended purpose. Following this scheme, we collect the common points of criticism for the available quality models and the usage scenarios of these models as they are found in the relevant literature. Based on these, as well as on our personal practical experiences, we derive a set of general requirements for quality models and their meta-models. These requirements can be used to judge the appropriateness of a model for a given situation as well as to improve existing quality models.

## 2. Software quality models

The last three decades in quality modeling generated a multitude of very diverse models commonly termed “quality models”. Examples on the spectrum of diverse models include taxonomic models like the ISO 9126 [12], metric-based models like the maintainability index (MI) [4] and stochastic models like reliability growth models (RGMs) [16]. On first sight, such models appear to have little relation to each other although all of them deal with software quality. We claim that this difference is caused by the different purposes the models pursue: The ISO 9126 is mainly used to *define* quality, metric-based approaches are used to *assess* the quality of a given system and reliability growth models are used to *predict* quality. To avoid comparing apples with oranges, we propose

to use these different purposes, namely *definition*, *assessment* and *prediction* of quality, to classify quality models. Consequently, we term the ISO 9126 as *definition model*, metric-based approaches as *assessment models* and RGMs as *prediction models*.

Although *definition*, *assessment* and *prediction* of quality are different purposes, they are obviously not independent of each other: It is hard to assess quality without knowing what it actually constitutes and equally hard to predict quality without knowing how to assess it. This relation between quality models is illustrated by the DAP classification shown in Fig. 1.



**Figure 1. DAP Classification for Q-Models**

The DAP classification views prediction models as the most advanced form of quality models as they can also be used for the definition of quality and for its assessment. However, this view only applies for ideal models. As Fig. 1 shows, existing quality models do not necessarily cover all aspects equally well. The ISO 9126, for example, defines quality but gives no hints for assessing it; the MI defines an assessment whose relation to a definition of quality is unclear. Similarly, RGMs perform predictions based on data that is not explicitly linked to a definition of quality.

To reflect the importance of the purpose, we propose a definition of quality models in this paper that explicitly takes the purpose of the quality model into account. Due to the diversity of different models, we deliberately do not restrict the type of model to a specific modeling technique or formalism:

**Definition 1 (Quality Model)** *A quality model is a model with the objective to describe, assess and/or predict quality.*

Independent of the modeling technique used to build a quality model, we consider the existence of a defined meta-model as crucial. Even though, in many cases, quality meta-models are not explicitly defined for existing quality models. A metamodel (or “structure model” [14]) is required to precisely define the model elements and their interactions. It not only defines legal model instances but also explains how models are to be interpreted. Accordingly, we define:

**Definition 2 (Quality Metamodel)** *A model of the constructs and rules needed to build specific quality models.*

Finally, it must be defined how the quality model can be used in the development and evolution of a software system. This typically concerns the process by which a quality model is created and maintained and the tools employed for its operationalization. We call this a *quality modeling framework*. Although not necessarily required for the enumeration and discussion of quality model requirements, we give a definition for the sake of clarity:

**Definition 3 (Quality Modelling Framework)** *A framework to define, evaluate and improve quality. This usually includes a quality metamodel as well as a methodology that describes how to instantiate the metamodel and use the model instances for defining, evaluating and improving quality.*

### 3. Critique

All the existing quality models have their strengths and weaknesses. Especially the latter have been discussed in various publications. We summarize and categorize these points of criticism. Please note that different quality models are designed for different intentions and therefore not all points are applicable to all models. However, every quality model has at least one of the following problems.

#### 3.1 General

One of the main shortcomings of the existing quality models is that they do not conform to an explicit meta-model. Hence the semantic of the model elements is not precisely defined and the interpretation is left to the reader.

Quality models should act as a central repository of information regarding quality and therefore the different tasks of quality engineering should rely on the same quality model. But today, quality models are not integrated into the various tasks connected to quality. For example, the specification of quality requirements and the assessment of software quality are not based on the same models.

Another problem is that today quality models do not address different views on quality. In the field of software engineering, the value-based view is typically considered of high importance [21]. This view is largely missing in current quality models [15].

The variety in software systems is extremely large, ranging from huge business information systems to tiny embedded controllers. These differences must be accounted for in quality models by defined means of customization possibilities. In current quality models, this is not considered [11, 13, 17].

### 3.2 Definition models

The existing quality models lack clearly defined decomposition criteria that determine how complex concepts of quality are to be decomposed. Most definition models depend on a taxonomic, hierarchical decomposition of quality attributes. This decomposition does not follow defined guidelines and can be arbitrary [3,5,14,15]. Hence, it is difficult to further refine commonly known quality attributes, such as availability. Furthermore, in large quality models, unclear decomposition makes location of elements difficult, since developers might have to search large parts of the model to assert that an element is not already contained. This can lead to redundancy due to multiple additions of the same or similar elements.

The ambiguous decomposition in many quality models is also the cause of overlaps between different quality attributes. Furthermore these overlaps are often not explicitly considered. For example, security is strongly influenced by availability (denial of service attack) which is also a part of reliability; code quality is an important factor for maintainability but is also seen as an indicator for security [1].

Most quality model frameworks do not provide ways for using the quality models for constructive quality assurance. It is left unclear how the quality models should be communicated to project participants. A common method of communicating such information are guidelines. In practice, guidelines that are meant to communicate the knowledge of a quality model experience various problems. Often these problems are directly related to corresponding problems of the quality models itself; e.g. the guidelines are often not sufficiently concrete and detailed or the document-structure of the guideline is not aligned according to an evident schema. Also the problem in giving rationales for the rules the guidelines impose is of that category. Another problem is that the quality models do not define tailoring methods to adapt the guidelines to the application area.

### 3.3 Assessment models

The already mentioned unclear decomposition of quality attributes is in particular a problem for analytical quality assurance. The given quality attributes are mostly too abstract to be straightforwardly checkable in a concrete software system [3,5]. Because the existing quality models neither define checkable attributes nor refinement methods to get checkable attributes, they are hard to use in measurement [9,15].

In the field of software quality, a great number of metrics for measurement have been proposed. But these metrics face problems that also arise from the lack of structure in quality models. One problem is that despite defining metrics, the quality models fail to give a detailed account

of the impact that specific metrics have on software quality [15]. Due to the lack of a clear semantic the aggregation of metric-values along the hierarchical levels is problematic. Another problem is that the provided metrics have no clear motivation and validation. Moreover, many existing approaches do not respect the most fundamental rules of measurement theory and, hence, are prone to generate dubious results [7].

Due to the problems in constructive and analytical quality assurance, also the possibility of certification on basis of quality models experiences elementary problems [9].

It has to be noted that measurement is vital for any control process. Therefore the measurement of the most important quality attributes is essential for an effective quality assurance processes and for a successful requirements engineering.

### 3.4 Prediction models

Predictive quality models often lack an underlying definition of the concepts they are based on. Most of them rely on regression using a set of software metrics. This regression then results in equations that are hard to interpret [8].

Furthermore, prediction models tend to be strongly context-dependent what also complicates their broad application in practice. Many factors influence the common prediction goals and especially which factors are the most important ones varies strongly. Usually these context conditions are not made explicit in prediction models.

## 4. Usage scenarios

In order to specify requirements for software quality models, we first need to know how the models shall be used. The usage purpose and context have a strong influence on the structure as well as the content of the model, independent of whether it is used for definition, assessment or prediction purposes.

*Definition* models are used in various phases of a software development process. During requirements engineering, they define quality attributes and requirements for planned software systems [15,23] and thus constitute a method to agree with the customer what quality means [15]. During implementation, quality models serve as basis of modeling and coding standards or guidelines [6]. They provide direct recommendations on system implementation and thus constitute constructive approaches to achieve high software quality. Furthermore, quality defects that are found during quality assurance are classified using the quality model [6]. Apart from their use during software development, definitional quality models can be used to communicate software quality knowledge during developer training or student education.

*Assessment* models often naturally extend definitional quality model use cases to control compliance. During requirement engineering, assessment models can be used to objectively specify and control stated quality requirements [15]. During implementation, the quality model is the basis for all quality measurements, i.e. for measuring the product, activities and the environment [6, 20, 22]. This includes the derivation of guidelines for manual reviews [5] and the systematic development and usage of static analysis tools [6, 19]. If applied in a continuous fashion, this use case demands high precision of results in order to be adopted by developers. During quality audits, assessment models serve as a basis of the performed audit procedure. In contrast to continuous assessments, audits can, due to their non-continuous nature, better cope with false positives and thus potentially require a different trade-off between completeness and precision. Thereby, internal measures that might influence external properties are monitored and controlled [15]. Apart from their use during software development, assessment models furthermore constitute the touchstone for quality certifications.

*Prediction* models are used during project management. More specifically, predictive models are used for release planning and in order to provide answers to the classical “when to stop testing” problem [18].

## 5. Requirements

Based on the collected points of criticism as well as the usage scenarios of existing quality models, we derive general requirements as well as requirements for definition models, assessment models and prediction models. In the following, we use the term *quality criterion* for an element of a quality model.

### 5.1. General requirements

Any quality model shall define how it can be integrated with the development tasks. These range from requirements engineering to quality assurance tasks. Using this information, the process engineer can define when and how the model has to be used in the development process.

### 5.2. Definition models

For the highest unambiguousness and structuredness, it is necessary for definition models to have an explicit meta-model that defines the constructs and rules used to build and extend the definition model [5, 14]. In order to clarify the further discussion, this allows to separate the requirements into model and meta-model requirements. Hence, a definition model shall be based on an explicit meta-model.

The meta-model shall define the structure of the definition model and an unambiguous decomposition mechanism for its content [5, 14]. This results in a structured content that allows a clear and unambiguous representation of quality criteria. Furthermore, the model can be extended and refined without introducing ambiguity.

The decomposition mechanism enables the requirement that the definition model shall support different levels of specificness [22, 23]. Very high-level quality criteria have a wide scope and hold for many different software systems but are only of limited use. In order to integrate quality criteria into the development process, more detail is necessary [5]. In the model, all these levels are needed.

Moreover, the meta-model shall ensure that all quality criteria in the model have a defined justification. It is essential for the use and acceptance of the definition model that for each quality criterion it shall be explained what is the reason for its existence. For example, it can be derived from a business goal or be part of a prescribed standard. Building on this, a definition model shall define a notion of completeness and be complete w.r.t. that notion. Using the examples again, the quality model is complete if it covers the corresponding business goals and the prescribed standards of the software product.

Quality criteria often have more than one justification and can influence each other. Therefore, there can be overlaps between them. These overlaps shall be made explicit and especially conflicting justifications and criteria shall be detectable. For example, the addition of an authentication mechanism may secure the data in a system but may render its normal usage more difficult.

As was clearly shown by Garvin [10], there are several different views on quality, such as the user view, the manufacturing view or the value-based view. Clear views shall be defined and it shall be possible to take these different views in the definition model [15, 22, 23]. A comprehensive definition model shall cover both internal product characteristics and externally visible product characteristics [3, 5, 6].

Using the support for different levels of specificness of the meta-model, the model shall be structured into a general base model and several specific purpose models [23]. Experience has shown that there is a considerable amount of quality criteria that are independent of a specific context such as the programming language or the application domain. These can be reused in the base model. The specific purpose models then capture the variability.

One main usage scenario of definition models is to agree with the customer on a definition of quality and to use them for requirements elicitation. Hence, the content of the model shall be easy to interpret and usable in discussions with the customer. In many cases that means prose quality criteria are preferable to complex formalisms.

The model shall be usable for constructive quality assur-

ance. In detail that includes the quick access to information in the model for software engineers. For a fast overview, it shall support the (automatic) derivation of concise guidelines for modeling and implementation. It shall also support the derivation of more detailed guidelines including all the information of the model to give justifications for the proposed rules, to give assessable rules and to structure the guideline well. In principle, this is mainly a requirement for the tool support. However, also the model itself has to provide the necessary information and structure to derive such guidelines.

### 5.3. Assessment models

An assessment model also contains quality criteria but in a way suitable for quality assessments. Hence, all quality criteria in an assessment model shall be assessable. If the assessment model supports decomposition, the assessment shall at least be possible at its lowest levels.

In principle, the assessment of the quality criteria can be qualitative or quantitative. As far as possible, the quality criteria shall be described with measures. Criteria that are not directly measurable shall be described qualitatively. This ensures that it can be used for the objective specification of quality requirements. However, the model shall not be limited to automatically measurable metrics but may also contain measures that require manual evaluation. A clear distinction between these two types of metrics is necessary. As manual assessment of software systems is very elaborate, it should be well supported and minimized by tools. Hence, the measurement shall be as automatable as possible.

Nevertheless, the provided metrics must be clearly motivated – ideally supported by a description model – and validated. That means all measures shall be consistent with measurement theory [7]. Especially the aggregation of metric values should be taken into account.

Several quality criteria with associated metrics are usually combined in an assessment model. Hence, the model content is a combination of a set of metrics. For a sensible quality assessment, this combination needs to be understandable by the assessor. Hence, the content of an assessment model shall be easy to interpret by all project participants.

Today's software systems are large and incorporate a variety of different functionalities. Quality requirements are often not equal across the whole system. Therefore, an assessment model shall support the specification of different required quality profiles for different parts of the software product [14].

For the assessment, the model shall describe with what techniques each quality criterion can be assessed, i.e. dynamic tests, automated static analysis, and manual review.

A clear assessment description helps in generating checklists and test guidelines for the criteria.

The model shall be described in a way that it can be used as a touchstone for certification. In many contexts certification is necessary. Hence, the assessment of quality using a model should also serve as a basis for a certification of that quality.

### 5.4. Prediction models

For prediction models, all requirements to assessment models hold as well. Additionally, we see the following requirements as necessary.

Prediction models are often built using statistical regression of a set of metrics. Depending on the regression method, this can lead to rather complex and artificial equations. For a sensible use of prediction model in practice, however, the equations need to be comprehensible. Hence, the content of a prediction model shall be easy to interpret.

In order to support the most important use cases of quality prediction models, they shall support predictions to aid the planning of test phases and release cycles. It predicts, for example, the number and occurrences of defects and thereby the maturity of a software.

## 6. Related work

There is a huge amount of work on various forms of quality models. However, comprehensive overviews and classifications are scarce. A first, broad classification of what he called “quality evaluation models” was proposed by Tian [20]. He distinguishes between the specificness levels *generalized* and *product-specific*. These classes are further partitioned along unclear dimensions. For example, he distinguishes *segmented models* for different industry segments from *dynamic models* that provide quality trends. Two of the authors built on Tian's work and introduced further dimension. Wagner discussed in [22] the dimensions *purpose*, *quality view*, *specificness* and *measurement* where the purposes are *construction*, *assessment* and *prediction*. This was further extended by Wagner and Deissenboeck [23] with the dimensions *phase* and *technique*. A thorough discussion of critique, usage scenarios and requirements along these dimensions is not provided in any of these contributions.

## 7. Conclusions

An impressive development of quality models has taken place over the last decades. These efforts have resulted in many achievements in research and practice. As an example, take a look at the field of software reliability engineering that performed a wide as well as deep investigation of

reliability growth models. In some contexts these models are applied successfully in practice. The developments in quality definition models even led to the standardization in ISO 9126 that is well known and serves as the basis for many quality management approaches.

However, the whole field of software quality models is diverse and fuzzy. There are large differences between many models that are called “quality models”. Moreover, despite the achievements made, there are still open problems, especially in the adoption in practice. Because of this, current quality models are subject to a variety of points of criticism that have to be acted on.

We provide a comprehensive definition of a *quality model* based on the purpose the model has. Using this tripartition in definition models, assessment models and prediction models (DAP), we summarized the existing critique and collected a unique collection of usage scenarios of quality models. From this, we derived a comprehensive set of requirements, again ordered in terms of the DAP classification, that can be used in two contexts: (1) evaluation of existing models in a specific context or (2) further developments and improvements of software quality models.

## Acknowledgements

This work has partially been supported by the German Federal Ministry of Education and Research (BMBF) in the project QuaMoCo (01 IS 08023B).

## 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. Merrit. *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] D. Coleman, B. Lowther, and P. Oman. The application of software maintainability models in industrial software systems. *J. Syst. Softw.*, 29(1):3–16, 1995.
- [5] F. Deibenböck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard. An activity-based quality model for maintainability. In *Proc. 23rd International Conference on Software Maintenance (ICSM '07)*. IEEE Computer Society Press, 2007.
- [6] R. G. Dromey. A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162, 1995.
- [7] N. Fenton. Software measurement: A necessary scientific basis. *IEEE Trans. Softw. Eng.*, 20(3):199–206, 1994.
- [8] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Trans. Softw. Eng.*, 25(5):675–689, 1999.
- [9] C. Frye. CMM founder: Focus on the product to improve quality, June 2008.
- [10] D. A. Garvin. What does “product quality” really mean? *MIT Sloan Management Review*, 26(1):25–43, 1984.
- [11] E. Georgiadou. GEQUAMO—a generic, multilayered, customisable, software quality model. *Software Quality Journal*, 11:313–323, 2003.
- [12] ISO. Software engineering – product quality – part 1: Quality model, 2001.
- [13] S. Khaddaj and G. Horgan. A proposed adaptable quality model for software quality assurance. *Journal of Computer Sciences*, 1(4):482–487, 2005.
- [14] B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni. The SQUID approach to defining a quality model. *Software Quality Journal*, 6:211–233, 1997.
- [15] B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. *IEEE Software*, 13(1):12–21, 1996.
- [16] M. R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.
- [17] J. Münch and M. Kläs. Balancing upfront definition and customization of quality models. In *Workshop-Band Software-Qualitätsmodellierung und -bewertung (SQMB 2008)*. Technische Universität München, 2008.
- [18] J. Musa and A. Ackerman. Quantifying software validation: when to stop testing? *IEEE Software*, 6(3):19–27, 1989.
- [19] R. Plösch, H. Gruber, A. Hentschel, C. Körner, G. Pomberger, S. Schiffer, M. Saft, and S. Storck. The EMISQ method and its tool support – expert based evaluation of internal software quality. *Journal of Innovation in Systems and Software Engineering*, 4(1), 2008.
- [20] J. Tian. Quality-evaluation models and measurements. *IEEE Software*, 21(3):84–91, 2004.
- [21] S. Wagner. Using economics as basis for modelling and evaluating software quality. In *Proc. First International Workshop on the Economics of Software and Computation (ESC-1)*, 2007.
- [22] S. Wagner. *Cost-Optimisation of Analytical Software Quality Assurance*. VDM Verlag Dr. Müller, 2008.
- [23] S. Wagner and F. Deissenboeck. An integrated approach to quality modelling. In *Proc. 5th Workshop on Software Quality (5-WoSQ)*. IEEE Computer Society Press, 2007.