

# Managing Model Quality in UML-based Software Development

Christian F.J. Lange, Michel R.V. Chaudron  
Department of Mathematics and Computer Science  
Technische Universiteit Eindhoven, P.O. Box 513  
5600 MB Eindhoven, The Netherlands  
{C.F.J.Lange,M.R.V.Chaudron}@tue.nl

## Abstract

*With the advent of UML and MDA, models play an increasingly important role in software development. Hence, the management of the quality of models is of key importance for completing projects successfully. However, existing approaches towards software quality focus on the implementation and execution of systems. These existing quality models cannot be straightforwardly mapped to the domain of UML models as source code and models differ in several essential ways (level of abstraction, precision, completeness and consistency). In this paper we present a quality model for managing UML-based software development. This model enables identifying the need for actions for quality improvement already in early stages of the life-cycle. Early actions for quality improvement are less resource intensive and, hence, less cost intensive than later actions. We discuss our experiences in applying the quality model to several industrial case studies. Finally we present a tool that visualizes our quality model. This tool helps in relating management level quality data to detailed data about specific quality subcharacteristics.*

## 1. Introduction

Models play a central role in modern software development. The Unified Modeling Language (UML) [18] has become the de facto standard for modeling software architectures and designs. For this reason we focus on UML as our main modeling language.

In the early stages of software development UML models are created in a sketchy manner. In this stage models serve as a means for understanding the problem domain and for communicating design decisions. The models are extended, refined and detailed throughout subsequent phases of the systems life-cycle. Ultimately, the UML models are used as a basis for implementation

and testing. If a project uses poor models, it risks encountering problems such as misunderstanding [13], building the wrong product, large testing effort and ultimately a low quality product.

To reduce the risk for these problems, models should be subject to continuous quality control - as is common for the case of source code. There are however, significant differences between source code and models. Source code closely corresponds to the executing systems as the latter can be automatically derived from the former. This correspondence is much weaker for models. Generally, models are more abstract than sources; i.e. details are left out. Furthermore, source code is subject to a formal syntax that is enforced by a compiler. UML modeling tools enforce syntactical constraints very weakly (generic drawing tools not at all) and semantic constraints are not enforced at all. Thus models may be incomplete and inconsistent. The inconsistency is aggravated by the fact that models employ multiple views. Inconsistencies between views are seldomly identified. Summarizing, UML and source code differ in abstraction level, precision, completeness, consistency and correspondence to the ultimate system (depicted in Table 1).

We based these observations on a number of industrial case studies into the use of the UML [12]. In these case studies we observed that the UML is used in many different ways, that models range from sketchy to detailed, but generally contain a large amount of defects.

To a degree these deficiencies seem inherent in UML models as it is a notation that aims to bridge the spectrum from informal problem statement to detailed design. However, these deficiencies potentially endanger the smooth execution of a project. Hence there is a need for objective means for establishing the quality of UML models.

These observations motivated us to develop a quality model for UML models that takes into account the aforementioned characteristics of models as well as the

manners and phases in which an organization aims to use the models. The quality model can be seen as complementary to existing quality model that focus on the quality of the system. This quality model for UML models aims to enable identifying the need for actions for quality improvement already in early stages of the life-cycle. Early actions for quality improvement are less resource intensive and, hence, less cost intensive than later actions [2]. Additionally new model-centric development techniques such as MDA [17] require high-quality models.

This paper is structured as follows. In Section 2 we summarize the most important approaches to software quality. In Section 3 we present our quality model for UML models. In Section 4 we present our experiences and ideas in applying the quality model. Section 5 concludes and discusses directions of ongoing and future work.

## 2. Software Quality

In this section we review the relevant literature on software quality and relate our approach for quality of UML models to it.

### 2.1. Perspective on Quality

The subject (software) product quality is widely discussed, and most organizations and stakeholders agree that quality is an important property of products. But there are many definitions of quality. To achieve the common target *quality*, one first has to agree upon the same definition. Garvin [8] defines five views of product quality:

**Transcendental view:** quality is seen as something than can be perceived but not defined.

**User view:** quality is seen as the fitness for the user's purpose.

**Manufacturing view:** quality is seen as conformance to the product's specification.

**Value-based view:** quality is seen in terms of a product's potential to generate economical turnover.

**Product view:** quality is decomposed into several characteristics that are inherent to the product.

The main application of our quality model is during development of the product. In this phase neither the user's perception nor the product's economical potential can be measured. During development only internal properties of the product and its model can be measured, hence, we apply the product view of quality since we decompose quality into characteristics that are inherent to the product.

### 2.2. Existing Approaches

Early quality models that still serve as reference ones are the models proposed by Boehm [3] and by McCall [15]. The ISO 9126 [9] standard for software quality contains a quality model that is based on McCall's model. These models together with the model used by Rombach [20] also decompose quality in subcharacteristics that are inherent to the product.

A prominent way of defining subcharacteristics is by means of metrics. In literature there are many metrics [6] defined that are based on source code (e.g. Chidamber and Kemerer's suite [4] or McCabe's Cyclomatic Complexity Measure [14]). The quality of a system is assessed using metrics by relating metrics to quality attributes as described in a *quality model*.

The models by Boehm and McCall structure the subcharacteristics into three hierarchical levels, where each level represents a different class of characteristics. The three levels of McCall contain *Uses*, *Factors* and *Criteria*, respectively. Boehm uses different vocabulary but similar meanings. The levels in Rombach's model are only hierarchically ordered and do not denote different levels of characteristics. We also use the three-level approach with a specific class of characteristics for each level in our quality model.

The aforementioned quality models target a system during its entire life-cycle and also take the user view and the manufacturer view into account. These models have a rather general view on quality with an emphasis on the product in operation and in maintenance. Both, the Boehm and the McCall model define three *Uses* as in the highest level of the quality tree: *operation*, *maintenance* and *transition*. (Note that Boehm and McCall use different terminology for some concepts.)

The existing quality models lack the distinction between the system and its description, which is important for controlling quality of UML models. In the next section we introduce a new concept at the top level of the quality tree to include this distinction in our development-centric quality model.

Balla et al. [1] proposed the QMIM framework to classify existing quality approaches and to aid users of the models to choose one from (or a combination of) the different approaches using the framework's classification. The classification is two-dimensional with the *object* of interest in one dimension (process, product or project/resource) and in the other dimension the *quality specification* (metric, quality attribute or definition). Our proposed quality model is characterized to cover the product and the process on the object-dimension and all levels on the quality-specification-dimension.

	Source Code	UML Model
Abstraction	low	high - medium
Precision	high	medium - low
Completeness	high	low
Consistency	high	medium - low

**Table 1. Differences between Source Code and UML Models**

### 3. A Quality Model for UML

Here we describe the specific needs for a quality model that focusses on model-based development, the quality model and its concepts.

#### 3.1. Need for a model-centric quality model

The programming languages have a formal syntax and semantics and can automatically be transformed to machine readable instructions. Hence, the artefact source code (being the description) and the system (the product) coincide. Therefore the description describes the product unambiguously. The dominant quality models described in Section 2 do not distinguish between quality characteristics that are inherent to the system and quality characteristics that are inherent to its description.

Models describe systems at a higher abstraction level than source code. Additionally the UML introduces degrees of freedom and potential for inconsistency. For example the UML lacks a formal semantics and does not have strict guidelines about which of the 13 diagram types and their elements must be used to describe a specific purpose. This results in the fact that a model does not describe the system unambiguously. Unlike for source code, there is a gap between the artefact *model* and the system described by it.

Models are extensively used during development, but also during maintenance. Hence, there is a need to describe the quality characteristics of models. The quality model presented in this section distinguishes between quality characteristics of the model as a description and the system described by it.

#### 3.2. Concepts in this model and their definitions

In this section we describe the levels of our quality models and the concepts that belong to the levels.

**Level 1: Use.** The top level of our quality model describes the high-level use of the artefact. The well

known quality models described in Section 2 define three uses at this level:

- **Operation:** this use combines quality characteristics that concern the product when it is executing. This implies that the system is readily implemented. The characteristics concerning the system's operation relate to the user view of product quality. We focus on the product view, hence the operation use is out of the scope of our quality model.
- **Transition:** this use combines quality characteristics that concern the product when it is moved to another environment. Transition issues are implementation dependent and are not addressed by modelling in the design phase of a system. Therefore the transition use is out of the scope of our quality model.
- **Maintenance:** this use combines quality characteristics that concern the product when it is changed.

These uses do not address development of a model-based design of the system. Therefore we introduce a new use:

- **Development:** this use combines quality characteristics that concern the product and its artefacts in the phases before the product is finished.

UML models are used in the development and maintenance phases. Therefore our quality model takes these two uses into account. The decomposition of these uses is presented in the sequel.

**Level 2: Purposes of Modeling.** The second level of the quality model contains the *purposes* of the artefact. A purpose describes why the artefact is used. Usually an artefact is not used for all purposes at the same time (in fact, all purposes *cannot* be served at the same time; see Section 4.1).

The purposes for modeling are presented and described in Table 2.

**Level 3: Characteristics.** The third level of our quality model contains the inherent characteristics of the artefact. The characteristics described in Table 3. According to the distinction between characteristics of the model and characteristics of the system (see Section 3.1) we indicate for each characteristic whether it is a model characteristic (column *M*) or a system characteristic (column *S*). Some characteristics are defined for both model and system. The only exception is *correspondence*: this is a characteristic of a model-system pair. Correspondence between a UML model and the

Purpose	Description
Modification	The model and the system enable easy modification. Possible modifications are corrections, i.e. removal of errors, extensions of the system, or adaptive changes due to changed requirements.
Testing	The model is used to generate test cases.
Comprehension	The model and the system are easily comprehensible, i.e. the system elements and their functionality are modelled such that they can be understood correctly in a reasonable amount of time
Communication	The model enables efficient communication about the system's elements, their behavior and the design decisions. Communication includes communication during the development phase with different stake holders and documentation for understanding the system in later phases such as maintenance.
Analysis	The purpose of the model is to explore and analyse the problem domain including its key concepts and making some early design decisions.
Prediction	The model is used to make predictions about quality attributes of the eventual implemented system. These predictions are used to make early and therefore cheaper improvements of the architecture and design.
Implementation	The model is used as basis for (manual) implementation of the source code of the system.
CodeGeneration	The model is used to automatically generate the source code of the system. Code generation can be complete or partial (only a skeleton of the source code is generated).

**Table 2. Purposes of modelling**

source code of a system is needed to enable using the model to understand the system.

**Level 4: Metrics and Rules.** The characteristics concepts of the three aforementioned levels of the quality model cannot be measured directly from the artefact. The characteristics can be measured by (a combination) of metrics and rules. A metric is a mapping from the empirical domain to the numerical domain, such that its value reflects the level of some property of the artefact or an element of the artefact. Rules can be seen as special cases of metrics. They are mappings to a binary value: true or false. Rules are usually defined for elements of artefacts (e.g. 'Abstract class X must have a subclass'). Hence, a rule is a numeric metric for an entire artefact (i.e. the number of elements violating the rule).

A large number of metrics and rules are proposed in literature (e.g. [6, 10, 4]). For the sake of brevity we cannot present all metrics in this paper. We present a representative set of rules and metrics that are applicable in our quality model. The presented list contains some specific metrics and rules, but also families of related metrics and rules. The metrics and rules of models are presented and described in Table 4. The relation between metrics and rules and the characteristics of the quality model is explained in the sequel.

### 3.3. Relations between Concepts

Now we are ready to present the relations between the aforementioned concepts. The concepts together

with the relations between them form the actual quality model. Figure 1 shows the quality model. The arrows indicate relations between two concepts of different levels. An arrow indicates that the concept of the lower level influences the concept of the higher level. The arrows can also be interpreted as follows: a lower level concept *is part of* all higher level concepts to which it is related by an arrow, and a higher level concept *contains* the related lower level concepts.

The relations between metrics and rules are shown in Table 5

## 4. Applying the Quality Model

We have explained the quality model for UML models. This section describes how the quality model fits into different phases of the development. Additionally we present a tool prototype that we have implemented to support working with the quality model. We report findings from industrial case studies.

### 4.1. Tailoring the model for different phases

The quality model covers eight purposes of modelling. The need for a specific purpose is not the same for every situation. When using the quality model, the important purposes, and, hence the important characteristics, should be carefully selected. The importance of purposes depends on the project phase, the

Characteristic	Model	System	Description	Ref
Complexity	✓	✓	Complexity: measures the effort required to understand a model/system	[6]
Traceability	✓	✓	The extent that relations between design decisions are explicitly described.	
Modularity		✓	The extent that its parts are systematically structured and seperated such that they can be understood in isolation.	
Completeness	✓	✓	A system possesses completeness to the extent that it's functionality covers all requirements. A model possesses completeness to the extent that overlapping parts of different views contain the same elements and the system is completely described by the model.	[12]
Consistency	✓		The extent that no conflicting information is contained	[5]
Communicativeness		✓	The extent that it facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and useful.	[3]
Self-Descriptiveness	✓	✓	The extent that it contains enough information for a reader to determine its objectives, assumptions, constraints, inputs, outputs, components, and status.	[3]
Detailedness	✓		The extent that it describes relevant details of the system	
Balance	✓	✓	A model possesses balance to the extent that all parts of the system are described at an equal degree of all other model characteristics. A system possesses balance to the extent that all parts of it are equally with respect to all other system characteristics	
Conciseness	✓		The extent that the system is described to the point and not unnecessarily extensive.	[3]
Esthetics	✓		The extent that its graphical layout enables ease of understanding of the described system.	[19]
Correspondence	✓	✓	A pair of model and system possess correspondence to the actual system to the extent that system elements, their relations and design decisions are the same in the model and the system	

**Table 3. Characteristics**

development process and characteristics (and taste) of the development team. For example in a development team that is geographically distributed over different countries, comprehension and communication are important purposes. In a project that develops a very large system containing many risks, early prediction of the quality attributes of the system is important to enable cost-effective improvements.

In several industrial case studies (see Section 4.2) we have discussed importance of modelling purposes with project members. As a result, we have built a schema that relates purposes of modelling to project phases. Figure 2 shows the relations between project phases and modelling purposes. The shaded boxes show that the purpose is seen as important for the specific phase. Darker shading indicates higher importance. This schema can be seen as an indication, but different priorities for individual projects are of course possible.

## 4.2. Industrial Case Studies

In our research we conduct case studies with industrial partners for three reasons: to empirically validate the developed methods, to get direct feedback on their

practical usefulness and to get insight in problems that occur when using the UML in industrial practice. A typical case study is structured in three steps:

1. **Interview.** In an initial interview with a representative of the project team we assess the project context. We investigate the phase, the purpose of modelling, the size (in persons and man years), the development process, known problems and other characteristics of the project.
2. **Analysis.** We analyze the UML model(s) using our metrics and rule-checking tools. The raw results are related to the described quality model
3. **Feedback.** The analysis results and findings are presented to the project team and the team members have the opportunity to discuss and interpret the findings.

Here we present the characteristics of a selection of our case studies including the project phase and the purposes of modelling in the project. The case studies were conducted in five different companies with different application domains.

Table 6 shows the data from the case studies including model size in terms of number of classes, team

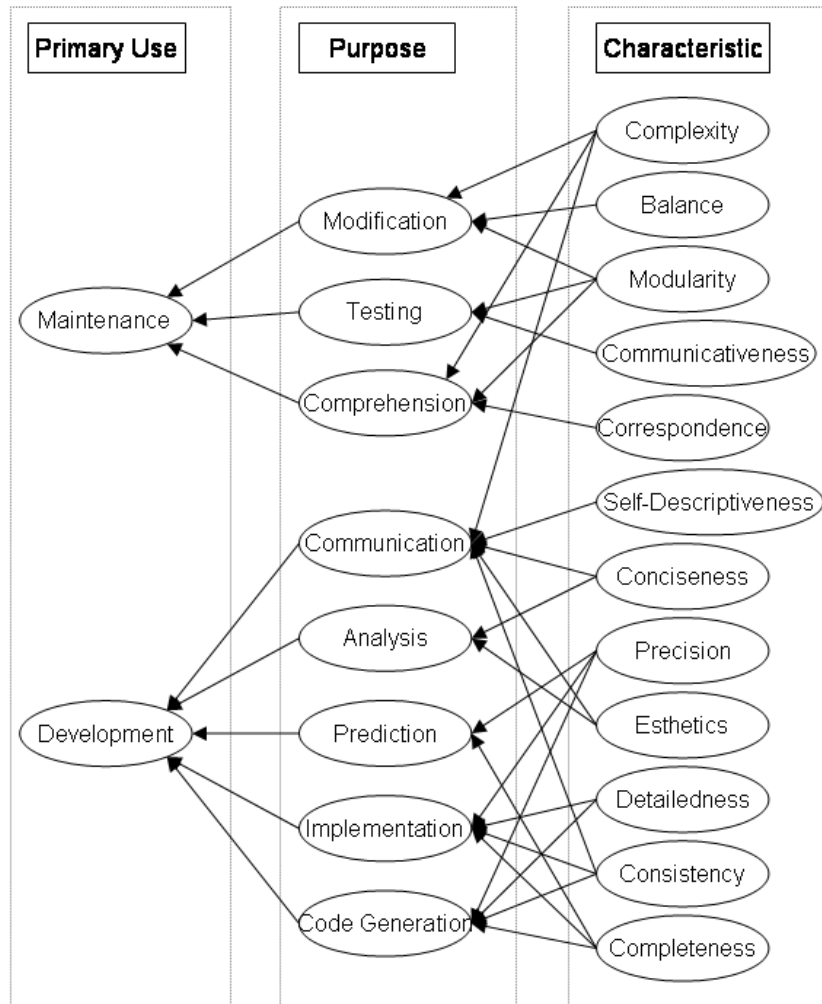
Name	Description	Ref
Dynamicity	Measures the complexity of a class' internal behavior. Dynamicity is based on the assumption that message calls correspond to state transitions. It takes information from the interaction diagrams into account.	[11]
Ratios	Ratios between number of elements (e.g. number of methods per class.)	
DIT	Depth of Inheritance Tree	[4]
Coupling	The number of other classes a class is related to.	[4]
Cohesion	measures the extent to which parts of a class are needed to perform a single task	[4]
Class Complexity	the effort required to understand a class	[16]
NCU	Number of classes per use case.	[16]
NUC	Number of use cases per class.	[16]
Fan-In	The number of incoming association relations of a class. Measures the extent to which other classes use the class' provided services.	
Fan-Out	The number of outgoing association relations of a class. Measures the extent to which the class uses services provided by other classes.	
Naming Conventions	Adherence to naming conventions	
Design Patterns	Adherence to design patterns	[7]
NCL	Number of crossing lines in a diagram.	[19]
Multi defs.	Multiple definitions of an element (e.g. class) under the same name.	
ID Coverage	Interaction diagram coverage. These metrics measure the extent to which the interaction diagrams cover the elements described in the structural diagrams. Examples are percentage of classes instantiated in interaction diagrams and percentage of use cases described by interaction diagrams.	[11]
Message needs	This consistency rule states that each message in an interaction diagram must correspond to a method defined in the class diagram.	
Method		
Code Matching	This family of metrics measures the extent to which the code matches the model. Example: percentage of model classes that occur in the code.	
Comment	Measures the extent to which the model contains comment. Example: lines of comment per class.	

**Table 4. Metrics and Rules**

size, phase and purposes of modelling indicated by the project team (marked by a '✓'). Each project team chose an individual set of modelling purposes. The analysis was tailored according to the indicated purposes.

The analysis of the case studies discovered the following concrete problems in the UML models:

- *Dead model parts.* In source code 'dead code' means regions of the source code that are outdated, but are still part of the artefact. In UML models we found similar problems. During our analysis we found complete packages and sometimes parts of packages of model that were not used any more and that contained outdated information. In this case, outdated regions of the model were neither removed nor explicitly marked as outdated. This can cause misunderstanding of the model and unnecessary effort for understanding the model. Poor values for the characteristics *conciseness* and *consistency* indicate this problem. This problem was found in case studies Case A and Case E.
- *Wrong model checked in.* In Case D the characteristics *completeness* and *consistency* showed very low values. In particular the ID coverage metric and rules for consistency between sequence diagram and class diagram showed that there were issues with the sequence diagrams. Due to these results the responsible developer discovered, that he had uploaded a premature version of the model (with sketches of the sequence diagrams) to the configuration management system. The correct version was only on his local machine and not accessible by the other team members.
- *Large number of inconsistencies.* All case studies showed large numbers of consistency defects as reported in [12]. For some case studies (Case G and Case H) we have analyzed subsequent versions of the model. We observed that the number of inconsistencies decreased when the developers started using the proposed methods for monitoring the quality of their systems.
- *Disbalance between developers.* In Case B the value for the characteristic *disbalance* was low. The de-



**Figure 1. Quality Model**

velopers had large differences in their habits of modelling, with respect to their modelling style, level of detail of modelling, and types and amount of inconsistencies. This is a risk when the model is used for communication between different developers.

All of the mentioned problems were unrecognized by the developers before the case study. The developers acknowledged the detected problems.

#### 4.3. QualityView: Tool-support for quality modelling

Information about a product's quality is useful for different stake holders. Managers use the information for example to monitor progress, developers use the

information to identify flaws in the product and find opportunities for improvement. The most prominent sources of information about a the quality of a system under development are metrics tools such as SDMetrics [21]. These tools deliver fine-grained information at the class- or system-level. This information is at the lowest level of quality models like McCall, Boehm and our own model. It is generally undefined how this data is related to higher level concepts of the quality model like characteristics, purposes and uses.

We have developed a prototype tool to assist the user in relating metrics data to quality attributes. The QualityView tool visualizes the entire quality model as a tree. The user can also tailor the quality model such that only the purposes of importance are visualized or extend the quality model by adding concepts, levels and relations. Besides allowing the user to tailor the

Metrics and Rules	Modularity	Complexity	Completeness	Consistency	Communicativeness	Self-Descriptiveness	Detailedness	Balance	Conciseness	Esthetics	Correspondence
Dynamicity		✓	✓					✓			
Ratios	✓		✓				✓	✓	✓		
DIT	✓	✓			✓			✓	✓		
Coupling	✓							✓			
Cohesion	✓	✓						✓			
Class Complexity								✓			
NCU	✓	✓						✓	✓		
NUC	✓	✓						✓	✓		
Fan-In	✓							✓			
Fan-Out	✓							✓			
Naming Conventions						✓		✓			
Design Patterns	✓		✓			✓		✓	✓		
Layout-Guidelines								✓		✓	
Multi defs.				✓				✓			
ID Coverage			✓					✓			
Message needs Method			✓	✓				✓			
Code Matching			✓					✓			✓
Comment						✓		✓			

**Table 5. Relations between metrics and rules and characteristics**

quality model to his needs the tool provides two use cases in its current version:

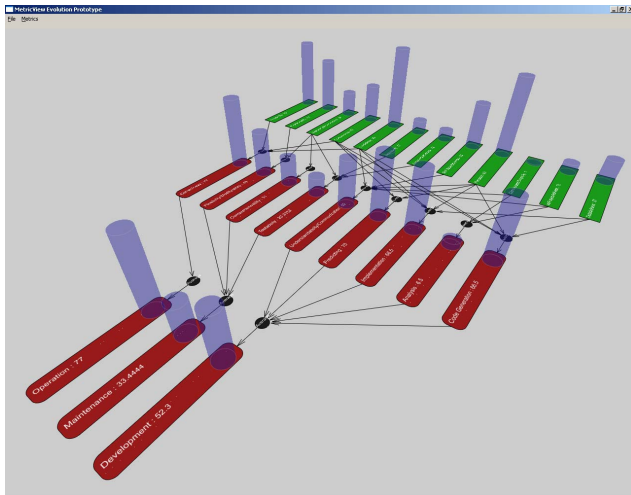
- *Assistance in navigating through the quality model.* The user can select a concept at any level of the quality model. All related concepts are highlighted to improve the understanding of how the concepts influence each other. A screenshot of this functionality is shown in Figure 3
- *Visualizing the values for the concepts.* QualityView is built on top of a metrics tool for UML models and takes the metrics as input. The value for each concept in the quality model is visualized in a three-dimensional view of the quality model: the height of the bar indicates the extent to which the analyzed model fulfills the concept. The value for concepts is composed of lower-level concepts. The composition function can be defined by the user to fit his specific needs. A screenshot of the three-dimensional visualization of values for concepts is shown in Figure 4.

## 5. Conclusions and Future Work

Models play an increasingly important role in software development. In order to support the management of quality from early phases of architecture and design, techniques are needed for assessing quality of models. Based on a number of industrial case studies, we have proposed a quality model for UML models. This model takes into account the different uses of models in a project as well as the phase in which a model is used. This model enables identifying the need for actions for quality improvement already in early stages of the life-cycle.

The main contribution of this paper is that we have developed a quality model that takes into account quality characteristics of the model as well as quality characteristics of the system. Our quality model is organized in a three-level decompositional structure. On the highest level we have introduced a new use: development. In the second level of our quality model we propose purposes for modeling. We relate these purposes to different phases in the life-cycle of the product. The set of important purposes differs per project phase,





**Figure 4. Three-dimensional visualization of the values for each concept in the quality tree using QualityView**

## References

- [1] K. Balla, T. Bemelmans, R. Kusters, and J. Trienekens. Quality through managed improvement and measurement (QMIM): Towards a phased development and implementation of a quality management system for a software company. *Software Quality Journal*, 9(3):177–193, November 2001.
- [2] B. Boehm. *Software Engineering Economics*. Prentice Hall, October 1981.
- [3] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, and M. J. Merrit. *Characteristics of Software Quality*, volume 1 of *TRW Series of Software Technology*. North-Holland Publishing Company, Amsterdam, 1978.
- [4] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.
- [5] S. Easterbrook and B. Nuseibeh. Using ViewPoints for inconsistency management. *BCS/IEE Software Engineering Journal*, pages 31–43, January 1996.
- [6] N. E. Fenton and S. L. Pfleeger. *Software Metrics, A Rigorous and Practical Approach*. Thomson Computer Press, second edition, 1996.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable Object-Oriented Software*. Addison Wesley, 1994.
- [8] D. Garvin. What does ‘product quality’ really mean? *Sloan Management Review*, 26(1):25–45, 1984.
- [9] ISO/IEC FCD 9126-1.2. *Information Technology - Software Product Quality*, part 1: quality model edition, 1998.
- [10] S. H. Kan. *Metrics and Models in Software Quality Engineering*. Addison Wesley Professional, 2nd edition, September 2002.
- [11] C. F. J. Lange. Empirical investigations in software architecture completeness. Master’s thesis, Technische Universiteit Eindhoven, September 2003. No. 969.
- [12] C. F. J. Lange and M. R. V. Chaudron. An empirical assessment of completeness in UML designs. In *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE’04)*, pages 111–121, 2004.
- [13] C. F. J. Lange and M. R. V. Chaudron. Effects of defects in UML models - an experimental investigation. In *Proceedings of the 28th International Conference on Software Engineering (ICSE’06)*. ACM, May 2006.
- [14] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.
- [15] J. A. McCall, P. K. Richards, and G. F. Walters. *Factors in Software Quality*, volume vol 1-3 of *AD/A-049-015/055*. Springfield, 1977.
- [16] J. Muskens, C. F. J. Lange, and M. R. V. Chaudron. Experiences in applying architecture and design metrics in multi-view models. In *Proceedings of EUROMICRO 2004, Rennes, France*, August 2004.
- [17] Object Management Group. *MDA Guide, Version 1.0.1*, omg/03-06-01 edition, June 2003.
- [18] Object Management Group. *Unified Modeling Language, Adopted Final Specification, Version 2.0*, ptc/03-09-15 edition, December 2003.
- [19] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton. UML class diagram syntax: an empirical study of comprehension. In *Australian symposium on Information visualisation*, volume 9, pages 113–120, September 2001.
- [20] H. D. Rombach. *Quantitative Bewertung von Software-Qualitäts-Merkmalen auf Basis struktureller Kenngrößen*. Phd thesis, Universität Kaiserslautern, June 1984.
- [21] J. Wüst. The software design metrics tool for the UML, version 1.3. <http://www.sdmetrics.com>.