

On Conceptualization of Quality

Vladimir A. Shekhovtsov

Department of Computer-Aided Management Systems,
National Technical University “Kharkiv Polytechnical Institute”, Ukraine
shekvl@yahoo.com

Abstract. We investigate the notion of software product quality from the point of view of its integration into conceptual modeling activities (we call this integrated notion a conceptual view of quality). We pay special attention to evolution of the treatment of this view over the history of the conceptual modeling. After reviewing the body of available papers, we state that the abundance of similarly applicable techniques implementing this view (quality models, ontologies, UML profiles, and metamodels) can be confusing, especially as neither clear rationale nor justification of applicability can be found for many of them. We show some ways to resolve this confusion by explicitly separating the original of quality (a conceptualization source) and its conceptual view, and generalizing quality representation approaches as conceptualization rules.

Keywords: software product quality, conceptual modeling, conceptualization

1 Introduction

Conceptual modeling is traditionally viewed as an activity related to capturing the knowledge about the desired system functionality (as quoted from the book by Olive [58], “the conceptual schema of an information system is the specification of its functional requirements.”) Sometimes this view can lead to ignoring the quality requirements. This is unfortunate, as looking at the system only from the point of view of its functionality severely restricts the analyst, and it is currently believed that failure to get quality requirements right might lead to the failure of the whole project.

To overcome this problem, in our opinion, the following goals need to be achieved:

- representing the quality concepts (specifications of the quality requirements) in a way compatible with conceptual modeling notions (we call this representation a *conceptual view of quality*);
- making this representation connected to the conceptual schema (in functionality-related sense) in a way allowing to address the whole spectrum of quality-to-functionality relationships;
- making possible to use the conceptual view of quality (connected to the conceptual view of functionality) as a driving force for the software process.

In this paper, we restrict ourselves with the first two goals, touching the last one only briefly. Our goal is to look at these goals from the point of view of evolution of their treatment over the history of the conceptual modeling.

Early in this history, the notion of quality received relatively little attention, but the coverage of this issue has been extended over time. Researchers proposed different techniques to express the product quality, in particular, *quality models*, *quality ontologies*, *UML quality profiles*, and *metamodels of quality*. While this abundance of methods allows the analyst to enjoy an excellent flexibility when choosing a representation technique, it also has its downside, because this choice can be confusing, especially as many approaches lack justification of their place in the body of methods dealing with quality (self-classification) or explanation of applicability for particular kinds of problems. In addition, the borders between different categories of methods are not always clear. In this paper, we attempt to look at this issue in more detail.

We continue by describing the evolution and the current state of the approaches aimed at expressing the conceptual view of quality in Section 2. In Section 3, we show some examples of the problems caused by an unclear and unjustified self-classification of these approaches and propose the classification approach aimed at resolving these problems to some degree. Our focus in Section 4 is on the treatment of a conceptual view of the quality-to-functionality relationships. In Section 5, we briefly outline possible issues of using the conceptual view of quality to drive the software process.

2 A conceptual view of quality

The purpose of this section is to look at the classification and evolution of means to describe a conceptual view of quality (achieving the first goal stated in the paper). In this paper, we call these means *quality representation approaches*.

We start from looking at quality models. They are the earliest means of representing a software quality predating most of the conceptual modeling research. After that, we investigate the evolution of the idea of representing quality using the notions closer to traditional conceptual modeling approaches focusing on UML profiles, ontologies, and metamodels. In our treatment, we follow all proposed self-classifications without attempts to assess or modify them; this is the goal of the next section.

2.1 Quality models

Quality models as taxonomies. These models were first introduced in the mid-70s by McCall [55] and Boehm [15] as taxonomies of quality characteristics (according to a perception of a product quality as a taxonomy of quality aspects by e.g. Garvin [37]). In these taxonomies, top-level elements represent general quality characteristics (functionality, reliability etc.); bottom-level attributes (quality sub-characteristics) represent more concrete characteristics (e.g. reliability is decomposed into fault tolerance, recoverability, etc.). Initially, the models did not allow any modifications of the proposed taxonomies (a *fixed model* approach [33]) and did not include any meta-information describing their structure.

Since then, the treatment of the taxonomic models (populated with characteristics) has evolved with respect to several criteria [21]:

1. *measurability* criterion – the degree of quantification of the quality sub-characteristics via quality measures (indicators). Initially, the quality models did not allow this quantification; the evolution of their treatment resulted in supplementing the models with extensive sets of measures [13, 33, 34, 76] and abandoning the measureless solutions. Current quality model standards like ISO/IEC 9126 [42, 43] and ISO/IEC 250xx [72], recommendations related to their application [24, 73], and derived solutions [59, 69] also include measures.
2. *structural complexity* criterion – indicating the degree of extending simple taxonomies to form more complicated graph-like structures. The treatment of this issue has been evolved into taking into account overlapping quality characteristics [17, 26, 39] or interrelationships between these characteristics [23, 46, 61]. Older pure-taxonomy solutions are still widespread, however.
3. *stakeholder perception* criterion – indicating the degree of taking into account an observation that different categories of stakeholders perceive different subsets of quality characteristics (“quality is in the eye of the stakeholder”). Only few solutions now allow this per-stakeholder filtering [68].

Quality models as meta-descriptions. Until now, we discussed the treatment of quality models defined as taxonomies of characteristics (even evolution along the structural complexity facet did not deviate from general characteristics of the models). On the other hand, another direction of evolution of the models has led to altering the very meaning of the quality model concept turning it into a meta-description. It was implemented via *custom model* and *mixed model* approaches [33]. These approaches allow a complete or partial modification of the model’s structure [13, 26, 41] and often do not include the taxonomies of characteristics into model description at all. As an example, IEEE proposed its standard (IEEE 1061-1998 [41]) based on custom model approach (only a structure of the model is defined; this structure is supposed to be customized for the particular applications). Another example is an attempt to perform UML-based formalization of the structure of the ISO/IEC 9126 quality model taxonomy [16, 21].

This representation has inherent problems; we discuss them in the Section 3.

2.2 UML quality profiles

UML quality profiles are the means of representing the quality in UML-based conceptual models. The criteria for their classification are as follows:

1. **Quality scope** criterion reflects the scope of quality taxonomy described via this profile. According to this criterion, there are two categories of quality profiles:
 - *partial profiles* describing particular quality characteristics (e.g. performance [7, 35, 60] or security [64]);
 - *embracing profiles* including the description of the complete quality taxonomy of the particular kind (for example, UML/QoS profile initially aimed at modeling quality of service and fault tolerance characteristics [1, 2], quality requirements

profiles [71, 79] connecting these requirements to conceptualizations of corresponding design decisions).

The evolution of UML profiles with respect to this criterion followed the path from partial profiles to more general embracing profiles, though both categories are widely used now.

2. **Representation** criterion reflects an approach to represent this profile. Some profiles extend the UML notation via supporting new kinds of diagrams etc.[7], other express everything via OCL [35] or stereotypes [1, 2] without extending the UML notation, mixed solutions are also available.

2.3 Quality ontologies

The earliest works describing ontological representation of quality in a problem space appeared in mid-90s, though some of them did not use the term “ontology” at all. In particular, Boehm and In [14] proposed the domain taxonomy which included the notion of quality aimed at helping to describe the conflicts in quality requirements. The way of representing this taxonomy was very close to ontological approach. Actually, the term “quality ontology” was first used in a paper [48] describing an ontology aimed at “a logical formalization of quality knowledge”; it decomposed the quality into several characteristics completely orthogonal to ones commonly used in traditional quality taxonomies and is not frequently referenced.

We can classify the ontological representation of quality in a problem space according to several criteria.

1. **Quality scope** criterion has mostly the same meaning as was described above for quality profiles. There are three categories of software quality ontologies according to this criterion:
 - examples of *partial ontologies* are specialized ontologies for some quality characteristics (e.g. performance [51], security and dependability [8, 47, 49], reliability [78]);
 - examples of *embracing ontologies* are measurement ontology [12], top-level QeGS (quality of e-Government services) ontology [52], ontology for software quality evaluation [66]. All of them incorporate complete quality taxonomies.
 - there is an additional category of *dedicated ontologies*. The sole purpose of such ontology is to represent a particular taxonomy of quality characteristics (examples are ISO/IEC 9126 quality ontology for ElicitO tool [4] and software product quality ontology used in ODE tool [31]).

An evolution of the ontological representation of quality with respect to this criterion was mostly similar to one for quality profiles (from partial to embracing ontologies). All these categories are widely used today.

2. **Generalization level** criterion reflects the level of generalization allowing by the particular quality ontology. It defines two categories of ontologies:
 - *concrete ontologies* include the complete taxonomies of quality characteristics and sub-characteristics. Among examples are an ontology by Boehm and In [14] and

ElicitO quality ontology [4] which includes the taxonomy of ISO/IEC 9126 quality characteristics;

- *descriptive ontologies* describe the structure of quality model taxonomies, often such ontologies do not include the elements of taxonomies (i.e. characteristics) themselves. Examples of such ontologies are numerous [12, 31, 52]; these ontologies include elements describing the type of quality characteristic, the type of quality measure etc.

Historically, the first known quality ontology [14] was of concrete kind, descriptive ontologies have been evolved later via generalization and are used more frequently now, though some concrete ontologies (e.g. [4]) are available as well.

3. **Application time** criterion reflects the applicability of the particular ontology. There are two categories of ontologies [65] with respect to their application time:
 - *development-time ontologies* that can be applied during the development time to drive the software process [12, 14, 66];
 - *run-time ontologies* that can be applied at run time to drive the particular tools (an example is ElicitO tool [4] which uses the ontology to help formulating questions for quality requirements elicitation interviews).
4. **Representation** criterion reflects the way of representing the particular ontology [65]. Some ontologies are represented informally (e.g. early ontology by Boehm and In [14]), approaches to formal description of quality ontologies include using UML diagrams to express their structure [31], using OWL [52] and other formal notations [12]. There are also ontologies available only via specific tools [4]. The categorization here is not strict, as ontologies can have several representations.

2.4 Metamodels of quality

Metamodeling activities also use the representation of software product quality in a solution space, though the number of quality-aware metamodels is not as numerous as the number of available quality ontologies. The metamodel-based representation of quality in a solution space has evolved with respect to the following criteria (mostly similar to the criteria defined for UML profiles and quality ontologies):

1. **Quality scope** criterion defines three categories of metamodels:
 - examples of *partial metamodels* are specialized metamodels addressing security [45, 74] and reliability [40];
 - an example of *embracing metamodel* is a quality measurement metamodel built on the foundation of a similarly structured ontology [19];
 - examples of *dedicated metamodels* are specialized metamodels describing the structure of quality taxonomies such as SQMREA metamodel [27], MOF-based metamodel for ISO/IEC 9126 quality model [18]. Actually, all metamodels of quality models should be included in this category.
2. **Representation** criterion has the same meaning as the corresponding criterion for quality ontologies. Some metamodels are described using UML diagrams [19, 27], others use specialized notation such as MOF [18] or Dublin Core.

3 A conceptualization process

In this section, we state that current state-of-the art classification of the quality representation approaches can cause some confusion for practitioners. We show some examples of confusing approaches and discuss the sources of this confusion. After that, we discuss possible ways to resolve the issue.

3.1 Classification problems

There is some confusion in a software engineering community related to representing the quality in a software process. To illustrate this confusion, it is better to start with some examples.

Example 1. Suppose it is necessary to select the conceptual representation of quality among the following four alternatives:

1. An IEEE 1061-1998 quality model [41];
2. A part of UML/QoS profile [1, 2] dedicated to general quality representation;
3. A software quality ontology by Falbo et al. [31];
4. A metamodel for ISO/IEC 9126 quality model by Burgués et al. [18].

The structure for all four representations of quality is shown on Fig.1.

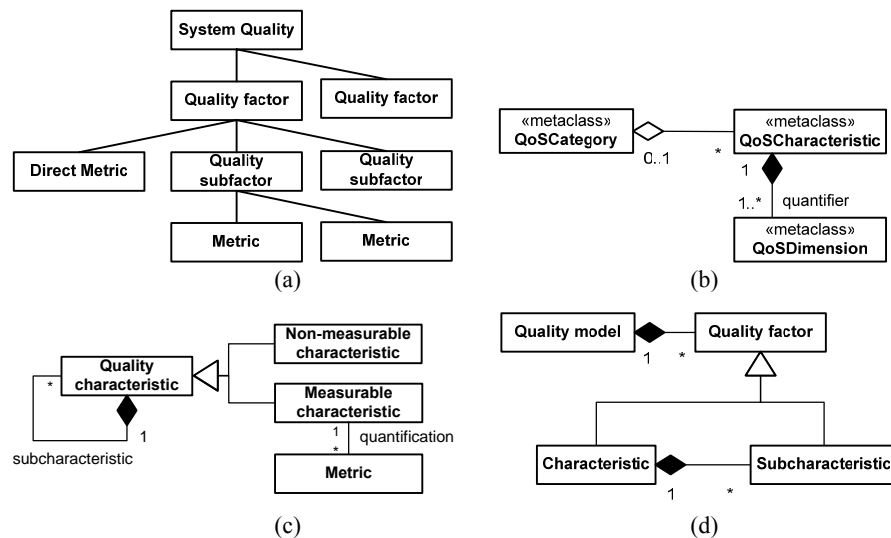


Fig. 1. Fragments of quality representations: IEEE 1061-1998 model [41] (a), UML/QoS profile [1, 2] (b), quality ontology by Falbo et al.[31] (c), ISO/IEC 9126 metamodel [18] (d)

It is obvious that all four approaches look very close to each other. All of them describe some taxonomy of quality characteristics and sub-characteristics (with some variations of naming them). In this situation, one can ask the questions like:

- Why do we need that many methods to choose from if they are so similar?
- What do they really have in common and is it possible to separate the common part and the differences?
- How do these techniques differ in their suitability for a particular purpose?

If we look at the descriptions of these approaches, we can see that even their authors often do not have obvious answers to these questions.

Example 2. Suppose it is required to select the quality ontology to describe the notion of quality for a particular problem. After looking at the available quality ontology approaches (see Section 2.3 above) it is possible to make the following list of alternatives:

1. A software quality ontology by Falbo et al. [31];
2. An ontology described as part of ElicitO tool [4];
3. A set of quality ontologies describing the necessary characteristics (e.g. reliability and performance [51, 78])

The second ontology as seen via Protégé tool and the third ontology in its original notation are shown on Fig.2 (the first one is depicted on Fig.1 above).

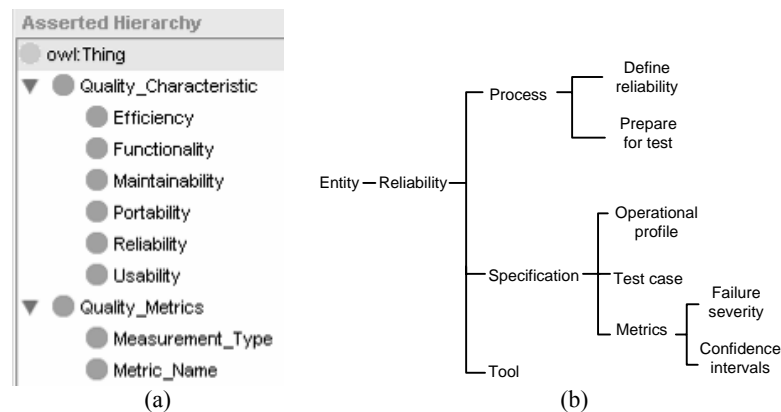


Fig. 2. Fragments of quality ontologies: ElicitO tool ontology (a), reliability ontology (b)

Here we can see another problem. This time, we have three solutions that position themselves as belonging to the same class but in fact describe the information of very different nature (and using rather different notations). It is also possible to observe similar problem while making a choice between several quality model solutions of “taxonomy” and “custom” kinds, e.g. between the model of ISO/IEC 9126 and the model of IEEE 1061-1998. In this case, one can suspect that some of the proposed solutions are actually misclassified.

The formulation of the problem. Based on the above examples, we argue that there are some facts that can cause the confusion mentioned at the start of this section.

1. There are too many approaches aimed at the purpose of representing the notion of quality and their descriptions often lack clear explanation of their purpose.
2. These approaches often look very close to each other and it is difficult to find the detailed explanation of their similarities and differences.
3. It can be not easy to see the place of the particular approach in the general taxonomy of available approaches
4. The rationale of choosing the particular approach for the problem at hand is often difficult to formulate because the applicability of this class of approaches to different classes of problems is not explained in detail;
5. There are usually no clear description of the quality concept underlying the particular quality representation and the relation of this representation to such concept;
6. It is difficult to figure out if actual properties of the method actually correspond to its classification proposed by the authors (so this self-classification is correct).

In the following section, we try to investigate these facts in more detail and look for the possible guidelines that can improve the situation with some of them.

3.2 The conceptualization of quality

We argue that it is possible to see from the above descriptions of the quality representation techniques that any conceptual view of quality in fact represents (*conceptualizes*) some underlying notion of quality. We call the process of transitioning from this underlying notion to a particular conceptual view *a conceptualization of quality*. Our goal is to describe this underlying notion (*conceptualization source*) together with the rules controlling the conceptualization process (*conceptualization rules*). Fig.3 illustrates the process of conceptualization.

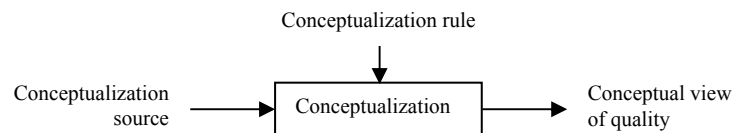


Fig. 3. A conceptualization of quality

The conceptualization source. To define this source we need to find something about the representation of quality that is independent of the notation or methodology. As mentioned above, we restrict ourselves with taxonomy view of product quality according to the works of Garvin [37]. We argue that in this case *the contents of such taxonomy are independent from any representation technique*. As a result, it is possible to see such pure taxonomy of concrete characteristics and sub-characteristics (or more complicated structure taking into account e.g. interdependencies between the characteristics) as a conceptualization source.

In a semiotic sense (according to e.g. [32]) the conceptualization source can be treated as a sign referent because this is the way the quality is perceived by stakeholder (it is kind of “real-world” quality). Actually, the only restriction we pose at this

source is that it must be concrete (so it can be some other structure of concept different from taxonomy).

We argue that *this concept is equal to the quality model in its simplest (fixed) sense* – as a set of characteristics, sub-characteristics and measures. We restrict quality models to the ones with specified characteristics because only such models reflect the real concept of quality as perceived by stakeholders. This way, for example, IEEE 1061-1998 quality model [41] does not comply to these requirements and we claim that *it is in fact a metamodel or ontology*. In general, this is true for all quality models built using “custom model” approach.

As a result, we can state the following guidelines relating current approaches reviewed in a Section 3 and conceptualization sources:

1. Taxonomy-based quality models (built following the fixed approach) are in fact conceptualization sources, so they should be excluded from the list of approaches to define conceptual views of quality;
2. Quality models built following custom approach are wrongly placed in the same class as taxonomy-based quality models; they are in fact metamodels or ontologies. Actual place of classification depends on the approach to their application: they are ontologies if they describe the quality for a problem space, metamodels – if they prescribe the notion of quality needed in a solution space.

The term *quality model* this way becomes a synonym for a *conceptualization source*. The models of a different structure (not taxonomies) can serve as such sources as well. An example can be the case of early enterprise quality ontology [48] which uses the notion of quality different from the standard taxonomy. We state that there is actually a special quality model underlying this ontology with a structure corresponding to its own notion of quality.

It is important to emphasize that we have actually limited the choice of conceptual view approaches to *quality profiles, ontologies, and metamodels*.

The conceptualization rules. After looking at the available quality representation approaches, we state that *the criteria of their classification define the relationships between the underlying quality model and the resulting view*. Actually, these criteria can be seen as attributes of the rules translating the underlying quality model into a conceptual view of quality (*conceptualization rules*).

For example, if we have an approach belonging to a category of *partial ontologies*, the process of conceptualization corresponding to this approach can be described via the conceptualization rule possessing an attribute of *Quality scope* with a value corresponding to *partial* category.

Some of the attributes are in fact independent from any representation approach, among them are *Quality scope* and *Generalization level* (the latter, while being initially proposed only for quality ontologies can be easily defined for quality profiles and metamodels), others are approach-dependent (such as *Representation*). In addition, to make these rules universally applicable we define an additional attribute of *View type* corresponding to the resulting view (ontology, metamodel, or quality profile).

Let us look at a more complete example. Suppose we need to define a rule generalizing an ontology described as part of ElicitO tool [4]. We can see from the Section

2.3 that this ontology is dedicated, concrete, run-time and is only available via the tool. As a result, the necessary rule will have attributes shown in Table 1.

Attribute	Value
View type	ontology
Quality scope	dedicated
Generalization level	concrete
Application time	run-time
Representation	tool

Table 1. Attributes of a conceptualization rule

Such rules can help to generalize any quality representation approach (*so they are the means of the universal treatment of the conceptualization of quality*) and even suggest the directions of future research (via investigating previously unimplemented combinations of attributes).

By establishing the above generalization, we distinguished the underlying notion of quality from its representation, revealed some sources of the problems described in Section 3.1 (namely, wrong placement of “custom” quality models) and better structured the set of available approaches.

While we did not address all the issues mentioned in Section 3.1, this structuring could help to describe the rationale for every approach in more unified way (actually, it is now possible to document and justify all the attributes of the conceptualization rule once and have this documentation available for all current and future conceptual view approaches).

4 Quality-to-functionality relationships

After establishing the conceptual view of quality, the next step is to connect it to the conceptual model of the system functionality (i.e. traditional conceptual model) using *quality-to-functionality (QF-) relationships*. The result of this connection is a quality-aware conceptual model (we show this process on Fig.4). We look briefly at the conceptual view of QF-relationships in this section; our plans are to address them in more detail in subsequent publications.

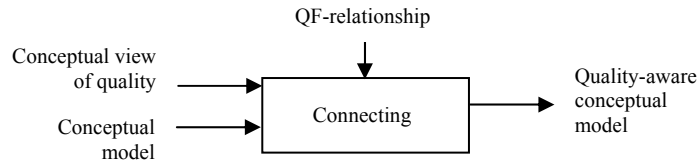


Fig. 4. Connecting the models using QF-relationships

QF-relationships can be classified according to several criteria:

1. **Structural** criterion defines two categories of relationships:
 - *conceptual relationships* defined via traditional relationship types of high cardinality [4, 12, 34];
 - *aspectual relationships* treating quality and functionality as crosscutting concerns [38]. As a result, they can be represented using Aspect-Oriented Modeling [29] notions. It allows utilizing extensive body of research carrying on in this field. Some examples are using special UML profiles for aspect-oriented concepts [6, 30], special kinds of diagrams [70], specialized modeling languages [57].

The evolution with respect to this criterion advanced from conceptual to aspectual relationships with proliferation of aspect-oriented techniques, but the former are still widespread (in industry [34] they are even more widely used than any alternatives).

2. **Involved notions** criterion reflects the connected notions (ontologies, metamodels, UML quality profiles). The evolution of the relationships with respect to this criterion reflects the evolution of the conceptual view of quality in general

3. **Behavioral** criterion also defines two categories of relationships:

- *static relationships* that depend on a structure of connected models. As an example, it is possible to express such relationships using crosscutting notions integrated into class diagrams [10, 22], conceptual relationships are mostly of static nature;
- *dynamic relationships* depending on a dynamic behavior of the connected systems. Examples are the relationships expressed using crosscutting extensions to the activity [9], sequence [25] or state [5, 77] UML diagrams (so the place to connect the quality to the functionality depends on a behavior of the program).

The evolution with respect to this criterion reflects the move from static to dynamic relationships. The former are still widely used.

We feel that it is possible to establish *QF-connection rules* to generalize QF-relationships similarly to establishing the conceptualization rules to generalize conceptual view approaches. An attributes of these rules would correspond to the criteria for QF-relationships. We will not discuss this issue in detail.

5 Software Process Perspective

In this section, we briefly look at the ways for the conceptual view of quality to influence the different stages of a software process (requirements engineering, software architecture design, and software implementation) or the process as a whole. As for QF-relationships, we plan to look at this issue in more detail in future.

When describing an influence of a conceptual view of quality on a software process, we need to investigate this influence for every quality representation approach. In fact, we need to establish the rules of applicability of every approach to every stage of a software process (the rules about using the quality ontologies to drive the requirements analysis and all other possible combinations “*approach driving stage*”). We should also establish similar rules for the influence of these approaches to software process in general.

As a first step towards these rules, to reflect the above-defined influence we can add a new criterion to the conceptualization rule (and, as a result, to the every view created via this rule), namely, a *Process scope* criterion. It reflects the scope (particular software process stage or several stages) of this relationship.

Below, we briefly outline some issues related to integrating the conceptual view of quality into different stages of a software process.

1. *Requirements Engineering*. Current research on non-functional requirements [23] (which can be in most cases considered equal to quality requirements [38]) utilizes conceptual view of quality in one way or other [4, 56]. There also some approaches aiming at elaborating support of the transition of the quality concepts from this stage into an architectural design [62]. The body of works on Early Aspects [63] allows utilizing aspect-oriented techniques to describe the relationships between quality and functionality at requirements engineering level. Also, the promising approach would be to combine a conceptual representation of requirements semantics (a predesign model [54]) with a conceptual view of quality [67].
2. *Architectural Design and Software Implementation*. The conceptual view of quality is used to drive the whole architectural design process [44], for selection of the best architecture based on a set of quality criteria [3, 75] and for other purposes. Aspect-oriented techniques can be used to represent crosscutting relationships between quality concepts and architectural components [50]. On software implementation step, the conceptual view of quality is utilized in MDA-based executable models [36] and for using aspect-oriented techniques at this stage.
3. *Integrated Approaches*. We can look at the existing attempts to elaborate integrated quality-driven approaches (covering the whole software lifecycle or several stages of it). The trend here is to move from more traditional view of the software process [28, 73] to MDA-based approaches [20, 53] in parallel to elaborating approaches specific for the particular quality characteristics [11]. We argue that it is also desirable if the view of quality used to drive the process would evolve as well following the trends outlined in Section 2.

6 Conclusions and future work

In this paper, we looked at evolution of quality representation approaches and proposed the set of criteria for their classification. After performing this review, we revealed that there are some problems with a current body of available approaches. We formulated these problems and illustrated them with examples. After that, we generalized these approaches using the concept of quality conceptualization rule. This generalization helped us to reveal some sources of the above problems and come out with some suggestions of improving the situation. We also discussed the notion of QF-relationships and showed some ways of their classification and application.

The ultimate goal of our research is to establish a *quality-driven software process* (QDSP) based on conceptual view of software product quality. Every step of this process would have a specific conceptual view of quality completely defined and justified in a way that it would adequately reflect the need of this step. Our plans are to

establish some conceptual views suitable for every step of QDSP and to elaborate the QDSP steps in detail.

References

1. Agedal, J., de Miguel, M.A., Fafournoux, E., Lund, M.S., Stolen, K.: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. TR 2004-06-01, OMG (2004)
2. Agedal, J.Ø., Ecklund, E.F.: Modelling QoS: Towards a UML Profile. In: Proc.UML'2002. pp. 275–289 (2003)
3. Al-Naeem, T., Gorton, I., Babar, M.A., Rabhi, F.A., Benatallah, B.: A quality-driven systematic approach for architecting distributed software applications. In: Roman, G.-C., Griswold, W.G., Nuseibeh, B. (eds.): 27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA. pp. 244-253. ACM (2006)
4. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: ElicitO: A Quality Ontology-Guided NFR Elicitation Tool. In: Proc. REFSQ 2007. Vol. 4542, pp. 306-319. Springer (2007)
5. Aldawud, O., Bader, A., Elrad, T.: Weaving with Statecharts. In: 1st AOM Workshop at AOSD.
6. Aldawud, O., Elrad, T., Bader, A.: UML Profile for Aspect-Oriented Software Development. Proc. of Third International Workshop on Aspect-Oriented Modeling, (2003)
7. Apvrille, L., Courtiat, J.-P., Lohr, C., de Saqui-Sannes, P.: TURTLE: A Real-Time UML Profile Supported by Formal Validation Toolkit. IEEE Trans. Software Eng. 30, 473–487 (2004)
8. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1, 11-33 (2004)
9. Barros, J.P., Gomes, L.: Towards the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach. In: 4th AOM Workshop at UML.
10. Basch, M., Sanchez, A.: Incorporating Aspects into the UML. In: Proceedings of Third International Workshop on Aspect-Oriented Modeling, March. (2003)
11. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Transactions on Software Engineering and Methodology (TOSEM) 15, 39-91 (2006)
12. Bertoa, M., Vallecillo, A., Garcia, F.: An Ontology for Software Measurement. In: Calero, C., Ruiz, F., Piattini, M. (eds.): Ontologies for Software Engineering and Software Technology, pp. 175-196. Springer (2006)
13. Bøegh, J., Depanfilis, S., Kitchenham, B., Pasquini, A.: A Method for Software Quality Planning, Control, and Evaluation. IEEE Software 23, 69-77 (1999)
14. Boehm, B., In, H.: Identifying Quality-Requirements Conflicts. IEEE Software 13, 25-35 (1996)
15. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merritt, M.J.: Characteristics of Software Quality. North Holland Publishing Company (1978)
16. Botella, P., Burgues, X., Carvallo, J.P., Franch, X., Grau, G., Marco, J., Quer, C.: ISO/IEC 9126 in practice: what do we need to know? Procs. First Software Measurement European Forum (SMEF), Roma, January (2004)
17. Buglione, L., Kececi, N., Abran, A.: An Integrated Graphical Assessment for Managing Software Product Quality. In: Proc. ICSQ'02. (2002)
18. Burgués, X., Franch, X., Ribó, J.M.: A MOF-Compliant Approach to Software Quality Modeling. In: Conceptual Modeling - ER 2005. Vol. 3716, pp. 176-191. Springer (2005)

19. Cachero, C., Calero, C., Poels, G.: Metamodeling the Quality of the Web Development Process' Intermediate Artifacts. In: Web Engineering. Proceedings of WISE'2007 Conference. Vol. 4607, pp. 74-89. Springer (2007)
20. Cachero, C., Poels, G., Calero, C.: Towards a Quality-Aware Web Engineering Process. In: Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'2007). pp. 7-16 (2007)
21. Carvallo, J.P.: Systematic Construction of Quality Models for COTS-Based Systems. PhD Thesis, LSI-UPC, Sept. 2005 (2005)
22. Chitchyan, R., Sommerville, I., Rashid, A.: An Analysis of Design Approaches for Crosscutting Concerns. In: AOD Workshop at AOSD.
23. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers (1999)
24. Cote, M., Suryan, W., Georgiadou, E.: Software Quality Model Requirements for Software Quality Engineering. Proc. SQM'06 (2006)
25. Deubler, M., Meisinger, M., Rittmann, S., Krüger, I.: Modeling Crosscutting Services with UML Sequence Diagrams. In: MoDELS. pp. 522-536 (2005)
26. Dromey, R.G.: Cornering the Chimera. IEEE Software 13, 33-43 (1996)
27. Dubielewicz, I., Hnatkowska, B., Huzar, Z., Tuzinkiewicz, L.: Software Quality Metamodel for Requirement, Evaluation and Assessment. In: Proc.ISIM'06. pp. 115-122 (2006)
28. Dubielewicz, I., Hnatkowska, B., Huzar, Z., Tuzinkiewicz, L.: An Approach to Software Quality Specification and Evaluation (SPoQE). In: Software Engineering Techniques: Design for Quality. Vol. 227, pp. 155-166. Springer (2007)
29. Elrad, T., Aldawud, O., Bader, A.: Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design. Generative Programming and Component Engineering: ACM SIGPLAN/SIGSOFT Conference, GPCE 2002, Pittsburgh, PA, USA, October 6-8, 2002: Proceedings (2002)
30. Evermann, J.: A meta-level specification and profile for AspectJ in UML. Proceedings of the 10th international workshop on Aspect-oriented modeling, pp. 21-27. ACM (2007)
31. Falbo, R.A., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engineering. In: Proceedings of the 14th international conference on Software engineering and knowledge engineering. pp. 351-358 (2002)
32. Falkenberg, E., Hesse, W., Lindgreen, P., Nilsson, B.E., Oei, J.L.H., Rolland, C., R.K.Stamper, Assche, F.J.M.V., Verrijn-Stuart, A.A., Voss, K.: FRISCO - A Framework of Information System Concepts - The FRISCO Report. FIP WG 8.1 Task Group FRISCO (1998)
33. Fenton, N., Pfleeger, S.L.: Software metrics: a rigorous and practical approach. PWS Publishing, Boston (1997)
34. Firesmith, D.: Using Quality Models to Engineer Quality Requirements. Journal of Object Technology 2, 67-75 (2003)
35. Flake, S., Mueller, W.: A UML profile for real-time constraints with the OCL. In: Proc. UML'2002. Vol. 2460, pp. 179-195. Springer (2002)
36. Fuentes, L., Sanchez, P.: Towards executable aspect-oriented UML models. Proceedings of the 10th international workshop on Aspect-oriented modeling 28-34 (2007)
37. Garvin, D.A.: Competing on the Eight Dimensions of Quality. Harvard Business Review 65, 101-109 (1987)
38. Glinz, M.: On Non-Functional Requirements. In: Proc. RE'07. pp. 21-26 (2007)
39. Grady, R.B., Caswell, D.L.: Software metrics: establishing a company-wide program. Prentice-Hall, Inc. Upper Saddle River, NJ, USA (1987)
40. Grassi, V., Mirandola, R., Sabetta, A.: From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In: Proceedings of the 5th international workshop on Software and performance. pp. 25-36 (2005)
41. IEEE 1061-1998: IEEE Standard for Software Quality Metrics Methodology. IEEE, Los Alamitos (1998)

42. ISO/IEC 9126-1, Software Engineering – Product Quality – Part 1:Quality model. ISO (2001)
43. ISO/IEC 14598-1, Software product evaluation – Part 1: General overview. In., (1999)
44. Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation. Carnegie Mellon University, Software Engineering Institute (2000)
45. Kern, A., Kuhlmann, M., Kuropka, R., Ruthert, A.: A meta model for authorisations in application security systems and their integration into RBAC administration. In: Proceedings of the ninth ACM symposium on Access control models and technologies. pp. 87-96 (2004)
46. Khaddaj, S., Horgan, G.: Factors in Software Quality for Advanced Computer Architectures. In: Proc.ESCOM'01. pp. 437-442. London (2001)
47. Kim, A., Luo, J., Kang, M.H.: Security Ontology for Annotating Resources. In: OTM Conferences 2005, Part II. Vol. 3761, pp. 1483-1499. Springer (2005)
48. Kim, H., Fox, M.S., Gruninger, M.: An Ontology of Quality for Enterprise Modeling. In: Proceedings of the 4th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. pp. 105-116. IEEE Computer Society Press (1995)
49. Klemen, M., Weippl, E., Ekelhart, A., Fenz, S.: Security Ontology: Simulating Threats to Corporate Assets. In: Proc.ICISS'2006. Springer (2006)
50. Krechetov, I., Tekinerdogan, B., Garcia, A., Chavez, C., Kulesza, U.: Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. 8th Workshop on Aspect-Oriented Modelling (AOM. 06), AOSD 6, (2006)
51. Lera, I., Sancho, P.P., Juiz, C., Puigjaner, R., Zottl, J., Haring, G., nter: Performance assessment of intelligent distributed systems through software performance ontology engineering (SPOE). *Software Quality Control* 15, 53-67 (2007)
52. Magoutas, B., Halaris, C., Mentzas, G.: An Ontology for the Multi-Perspective Evaluation of Quality in E-government Services. Sixth International EGOV Conference. Springer (2007)
53. Matinlassi, M.: Quality-driven software architecture model transformation. Proc. of Working IEEE/IFIP Conference on Software Architecture (2005)
54. Mayr, H.C., Kop, C.: Conceptual Predesign - Bridging the Gap between Requirements and Conceptual Design. In: 3rd International Conference on Requirements Engineering (ICRE '98), Putting Requirements Engineering to Practice, April 6-10, 1998, Colorado Springs, CO, USA, Proceedings. pp. 90. IEEE Computer Society (1998)
55. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. NTIS (1977)
56. Mead, N.R., Hough, E.D., II, T.R.S.: Security Quality Requirements Engineering (SQUARE) Methodology. Software Engineering Institute (2005)
57. Meier, S., Reinhard, T., Seybold, C., Glinz, M.: Aspect-Oriented Modeling with Integrated Object Models. In: Mayr, H.C., Breu, R. (eds.): Modellierung 2006, 22.-24. Marz 2006, Innsbruck, Tirol, Austria, Proceedings. Vol. 82, pp. 129-144. GI (2006)
58. Olive, A.: Conceptual Modeling of Information Systems. Springer Verlag (2007)
59. Olsina, L.: Web-site Quality Evaluation Method: a Case Study on Museums. In: Proc.2nd Workshop on Software Engineering over the Internet at ICSE'99. (1999)
60. OMG: UML Profile for Schedulability, Performance, and Time Specification. Version 1.0. In., (2003)
61. Perry, W.: Effective methods for EDI quality assurance. Prentice-Hall, Upper Saddle River, NJ (1987)
62. QuadREAD Project. Univ. of Twente, Twente, the Netherlands (2006)
63. Rashid, A., Sawyer, P., Moreira, A., Araujo, J.: Early aspects: a model for aspect-oriented requirements engineering. Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on 199-202 (2002)

64. Rodríguez, A., Fernández-Medina, E., Piattini, M.: Capturing Security Requirements in Business Processes Through a UML 2.0 Activity Diagrams Profile. In: *Advances in Conceptual Modeling - Theory and Practice*, pp. 32-42. Springer (2006)
65. Ruiz, F., Hilera, J.: Using Ontologies in Software Engineering and Technology. In: Calero, C., Ruiz, F., Piattini, M. (eds.): *Ontologies for Software Engineering and Software Technology*, pp. 49-102. Springer (2006)
66. Sack, P., Bouneffa, M., Maweed, Y., Basson, H.: On Building an Integrated and Generic Platform for Software Quality Evaluation. *Proc.ICTTA'06, Vol. 2*, (2006)
67. Shekhovtsov, V.A., Kop, C., Mayr, H.C.: Capturing the Semantics of Quality Requirements into an Intermediate Predesign Model. In: *Proc.SIGSAND-EUROPE'2008 Symposium. GI* (2008)
68. Siakas, K., Berki, E., Georgiadou, E., Sadler, C.: The Complete Alphabet of Quality Software Systems: Conflicts and Compromises. *Proc. 7th World Congress for Total Quality Management*, pp. 603-618 (1997)
69. Simão, R.P.S., Belchior, A.D.: Quality Characteristics for Software Components: Hierarchy and Quality Guides. In: *Component-Based Software Quality: Methods and Techniques. LNCS 2693*.(2003). Vol. 2693. Springer, Heidelberg (2003)
70. Stein, D., Hanenberg, S., Unland, R.: Query Models. In: *Proc. UML'04*, pp. 98-112 (2004)
71. Supakkul, S., Chung, L.: A UML profile for goal-oriented and use case-driven representation of NFRs and FRs. In: *Proc. 3rd Intl. Conf. on Software Engineering Research, Management & Applications (SERA05)*, pp. 11-13 (2005)
72. Suryan, W., Abran, A., April, A.: ISO/IEC SQuaRE: The second generation of standards for software product quality. In: *Proc.of the 7th IASTED International Conference on Software Engineering and Applications*. (2003)
73. Suryan, W., Abran, A., Laporte, C.: An integrated life cycle quality model for general public market software products. *Proceedings of 12th International Software Quality Management & INSPIRE Conference (BSI) 5-7* (2004)
74. Susi, A., Perini, A., Mylopoulos, J.: The Tropos Metamodel and its Use. *Informatica* 29, 401-408 (2005)
75. Svahnberg, M., Wohlin, C., Lundberg, L., Mattsson, M.: A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates. *International Journal of Software Engineering and Knowledge Engineering* 13, 547-573 (2003)
76. Trendowicz, A., Punter, T.: Quality Modeling for Software Product Lines. In: *Proc.QA-OOSE'03 Workshop* (2003)
77. Whittle, J., Moreira, A., Araújo, J., Jayaraman, P., Elkhodary, A., Rabbi, R.: An Expressive Aspect Composition Language for UML State Diagrams. In: *MODELS. International Conference on Model Driven Engineering Languages and Systems, Nashville, TN*. (2007)
78. Zhou, J., Niemelä, E., Evesti, A.: Ontology-Based Software Reliability Modelling. In: *Proc. Software and Services Variability Management Workshop*, pp. 17 - 31 (2007)
79. Zhu, L., Gorton, I.: UML Profiles for Design Decisions and Non-Functional Requirements. In: *Proceedings of the Second Workshop on SHaring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*. (2007)