On the Evolution of Quality Conceptualization Techniques

Vladimir A. Shekhovtsov

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

Abstract. We investigate the notion of software product quality from the point of view of its integration into the modeling activities on the same level of abstraction as traditional functional models (a conceptualization of quality). We pay special attention to the evolution of the approaches for obtaining this conceptualization through the history of conceptual modeling, propose their classification according to common attributes and outline their distinguishing features. Based on the proposed classification, we outline a way of establishing an evaluation framework for quality conceptualizations aiming at supporting the choice of a conceptualization solution best suited for the problem at hand.

Keywords: software product quality, conceptual modeling, quality conceptualization, quality model, quality ontology, quality evaluation.

1 Introduction

Conceptual modeling is often viewed as an activity related to capturing the knowledge about the desired system functionality (as quoted from [89], “the conceptual schema of an information system is the specification of its functional requirements.”) This view, however, can lead to *the problem of quality-unaware conceptual modeling*:

– looking at the problem domain to obtain its conceptual model only from the point of view of the functionality of the system-to-be restricts the analyst and can be the source of mistakes due to the fact that quality has to be introduced into the existing system model later on the development lifecycle;
– these mistakes are often difficult to find until the later stages of the software process, because many quality-related issues become only evident when the system is put into use; on the other hand, being parts of early decisions they can be difficult and costly to fix [25, 49, 56].

To overcome the above problem, it seems natural to represent the quality concepts in a way compatible with conceptual modeling notions (define a *conceptualization of quality*) and make this representation connected to the conceptual schema of the desired system’s functionality (i.e. reflecting the software quality ***in*** conceptual models). Researchers proposed different techniques to conceptualize the product quality, in particular, *quality modeling* [20, 28, 32] and *metamodeling* [17, 18, 38], *quality ontological engineering* [13, 31, 57] etc.

While this abundance of methods allows the analyst to enjoy an excellent flexibility when choosing a conceptualization technique, it also has its downside, because this choice can be confusing. The problem is that the current research literature lacks a unified classification of the quality conceptualization techniques (in particular, we are not aware of any comparative treatment of quality modeling and quality ontology engineering). As a result, *it is difficult for analyst to decide which quality conceptualization solution is better suited for the problem at hand*. It is unfortunate to observe this situation in contrast to extensive attention paid to e.g. the classification and unification of the approaches to the problem of handling the quality ***of*** conceptual models [75, 84]. In this paper, we aim at overcoming this problem by elaborating a detailed classification of the existing quality conceptualization techniques together with a view of their evolution along the proposed classification dimensions. This classification can serve as a foundation for the evaluation framework for quality conceptualizations as these dimensions could underlie the quality criteria for evaluation.

Section 2 continues by introducing the concept of quality to be used in this paper. In Section 3, we present a classification of quality conceptualization techniques based on the dimensions of their evolution. In Section 4, we outline the way of establishing the evaluation framework based on this classification. Section 5 describes the related work; in Section 6, we make conclusions, and outline future research directions.

2 A Concept of Quality

Before elaborating the classification of quality conceptualization techniques (QCT), it is necessary to define the concept of quality to be used in this paper as we plan to build this classification, in part, upon the properties of this concept. We need to find a common ground here, as different researchers often rely on definitions of quality referring to quite different things [67]: adherence to requirements, fitness for a particular purpose, user satisfaction etc. We believe that it can be done based on the recent efforts of establishing the notion of software quality and its usage in terms of the formal ontology [78]. In this paper, we rely on this body of work, represented by Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [79] and, more specifically, by Core Ontology for Requirements Engineering (CORE) [57].

Quality Concept in DOLCE and CORE. We follow the approach of the authors of CORE in adopting the definition of quality from DOLCE to the case of software product quality. In doing so we agree that *Quality* (as a whole) is a concept embracing the set of particular *qualities* [79, pp. 16-18] i.e. perceivable and measurable entities that characterize particular individuals (such as functional units of the system-to-be) and are related to other qualities (reflecting e.g. a hierarchical order or quality interdependencies). Every quality is accompanied with a conceptual space [46] which is a collection of quality dimensions defining how the individuals are positioned (measured) according to this quality; it is called *quality space* in DOLCE. Qualities (such as *the response time for the particular operation*) are distinguished from their values (such as *1 sec*); these values are called *quales* and describe positions of individuals in a particular quality space.

To clarify the properties of qualities as perceivable entities we turn to CORE that categorizes ways of perceiving quality by stakeholders as a part of communicated information. This categorization is based on the type of the speech act used by stakeholders [57, p. 181]. Directive acts (ordering something to be done) define *goals*:

- precise (usually quantitative) *quality constraints* if the accompanying quality space is shared between all stakeholders so everybody agree how to measure this quality e.g. "*the response time of X must be below 0,5 sec*";
- vague qualitative *softgoals* in a NFR framework sense [26] if there is no such agreement so the perception of this quality is stakeholder-specific e.g. "*the performance of X must be good*";

Other kinds of speech acts define, in particular, (soft or hard) domain assumptions (descriptions of something existing in a domain e.g. "*the average response time of X is 0,4 sec*") and evaluations (comparative statements about something in a domain e.g. "*the response time of X in case of Y is 30% better than in case of Z*").

Our Definition of Quality. For now, we can distinguish the following properties of the particular quality *qc* in a DOLCE sense (i.e. the quality characteristic in a sense of ISO/IEC 25010 standard [55]):

- measurability (availability of the quality space for *qc*);
- characteristics of the quality space of *qc* (such as its structure or degree of sharing);
- characteristics of the relationships between *qc* and other qualities;
- characteristics of the relationships between *qc* and the functional units of the system-to-be characterized by this quality;
- characteristics of the relationships between *qc* and stakeholders referring to this quality in their description of the system-to-be.

To formalize this view in context of all available qualities, we describe the quality of the software system as

$$Q = \langle QC, QR, QU, QF \rangle \tag{1}$$

where

- *QC* is a set of qualities (quality characteristics) in a sense of DOLCE where every $qc \in QC$ is accompanied by a shared or private quality space with particular characteristics;
- *QR* is a set of relationships among particular qualities (such as those defining a hierarchy of qualities or interdependencies between particular qualities such as reliability and performance): $qr(QC' \subseteq QC) \in QR$, where *qr* is a particular relationship of this kind, *QC'* is a set of qualities involved into this relationship;
- *QU* is a set of relationships among qualities and stakeholders of the system-to-be such as those defining stakeholder speech acts involving the particular quality (directive acts to define goals etc) and connecting private quality spaces to stakeholders: $qu(QC' \subseteq QC, U' \subseteq U) \in QU$ where *qu* is a particular relationship of this kind, *QC'* is a set of involved qualities, *U'* is a set of involved stakeholders, *U* is a set of all available stakeholders;

– *QF* is a set of relationships among qualities and functional entities defined by a particular architecture of the system-to-be characterized by these qualities: $qf(QC' \subseteq QC, F' \subseteq F) \in QF$ where *qf* is a particular relationship of this kind, *QC'* is a set of involved qualities, *F'* is a set of involved functional entities, *F* is a set of all available functional entities.

In the following section, we will use this definition to establish a classification framework for quality conceptualization techniques.

3 Quality Conceptualization Techniques: Dimensions of Evolution

We start from the definition of a conceptualization of quality to be used throughout this paper: ***a conceptualization of quality*** *is a representation of the system quality on the same level of abstraction as the conceptual model of the functionality of this system.* Currently such conceptualizations are obtained using various quality conceptualization techniques. The purpose of this section is to establish a classification of these techniques and to look at their evolution through the history of conceptual modeling. We find a set of dimensions or "axes" for the treatment of quality to evolve along and use these dimensions as a foundation for the proposed classification.

Categories for QCT Evolution Dimensions. Based on the definition of quality presented in Section 2, we distinguish five categories of QCT evolution dimensions (covering 14 dimensions in total).

First category reflects the properties of the conceptualization process itself:

1. *Conceptualization process dimensions* (2 in total) reflect the properties of the process of performing QCT (such as QCT ability to support model-based or ontological view on quality, and to represent quality on different levels of abstraction).

Next three categories group dimensions related to a QCT's ability to support software quality defined according to (1):

2. *Quality properties support dimensions* (3 in total) reflect a QCT's ability to support different properties of the *QC* set, including *QR* relationships (such as the way of organizing the set of qualities, the way of organizing quality spaces, the support for dependencies between qualities etc);
3. *Quality perception support dimensions* (2 in total) reflect a QCT's ability to support the properties of the *QU* set describing quality as a perceivable entity (such as the ability to reflect shared and separate quality spaces, the support for quality constraints and softgoals etc);
4. *Quality usage support dimensions* (3 in total) reflect a QCT's ability to support the properties of the *QF* set to model such software process artifacts as quality requirements or attributes (such as the support of *qf* relationships themselves, the availability of particular techniques for their description etc.).

Last category covers dimensions related to QCT applicability:

5. *Applicability dimensions* (4 in total) reflect a QCT's applicability for the particular classes of problems (such as the stage of a software lifecycle it can be used for, if it is applicable for the particular application domain or software category etc);

Next, we will look in detail at the particular dimensions belonging to these categories.

3.1 Conceptualization Process Dimensions

These dimensions reflect the capabilities of the QCT process. We start from two general dimensions reflecting model-ontology dichotomy and the supported abstraction level.

3.1.1 Conceptualization Space

Purpose: to reflect the problem-solution space dichotomy; following [7, 90] we define its two possible scale values (Table 1):

Table 1. *Conceptualization space* scale

Value	Meaning	Examples
quality modeling technique (QMT)	Means of expressing software quality in a *solution space* (prescriptively defining the system-to-be) under "*closed-world*" assumption (everything not explicitly described is assumed non-existent)	[15, 37, 55, 58], reviews are in [20, 32]
quality ontology engineering technique (QOT)	Means of expressing software quality in a *problem space* (describing the real-world problem domain addressed by the system under development) under "*open-world*" assumption (everything not explicitly described is assumed unknown)	[31, 40, 59], no reviews except [78] covering formal techniques not restricted to software quality

Notes on evolution. QMT are considerably older than QOT: first quality models were introduced in mid-70s [15, 80], whereas the earliest works describing ontological representation of quality appeared in mid-90s. To our knowledge, the term "quality ontology" was first used in a paper [66] describing an ontology aimed at "a logical formalization of quality knowledge"; it showed the quality as a set of concepts supporting predicting and assessing the quality for the software system. Later, [14] proposed the domain taxonomy (defined by means of ontological approach) aimed at helping to describe the conflicts in requirements; it also covered the quality concept.

For many years, QMT significantly outnumbered QOT, but recently the evolution trend started to reflect the growing need for deeper ontological foundations for quality conceptualization e.g. based on cognitive theory [46, 79, 87].

3.1.2 Abstraction Level

Purpose: to reflect that quality conceptualization solutions can be described on different levels of abstraction [112]; we consider abstraction levels for QMT and QOT separately as they treat such levels differently (despite some similarities).

Abstraction levels for quality models. We start from describing the abstraction level dimension of the QMT evolution according to the notion of a modeling meta-pyramid [7] where the solution belonging to a particular level defines the model for a language used to define the solutions on the level below (i.e. a metamodel). We restrict the pyramid to three levels (the scale values in Table 2 correspond to these levels):

- *M1 level* – for the **models** of some phenomenon; in our case a quality model directly describes the phenomenon of quality as used in a system-to-be;
- *M2 level* – for the **metamodels** (models of languages used for describing the M1-level models); in our case a quality metamodel is used to define quality models;
- *M3 level* – for the **meta-metamodels** (models of languages used for describing the metamodels); here a meta-metamodel can be used to define quality metamodels.

Table 2. *Abstraction level* scale (QMT)

Value	**Meaning**	**Examples**
concrete	QMT solution belongs to M1 level; e.g. it is a taxonomy of quality characteristics with more general ones on the top level and more concrete ones on the levels below, often not changeable (*fixed model* QMT [41]) and described by example without meta-information. Criticized due to the arbitrary choice of quality characteristics and the lack of connection to measurable quality attributes at the bottom level [32, 67]	early QMT [15, 80] standards [54, 55], etc.
meta-descriptive	QMT solution belongs to the M2 (meta-) level (a metamodel)	
	implicit quality metamodel: a metamodel not documenting this fact (introduced "bottom-up" e.g. via *custom* or *mixed model* QMT [41] allowing modification of the model structure and often omitting quality characteristics from the model descriptions)	[37, 53]
	explicit quality metamodel: introduced "top-down" with an explicit purpose of describing the set of possible quality models; industry examples are *UML quality profiles* adding support for new modeling concepts to UML via an extension of the UML metamodel	generic metamodels: [17, 18, 58, 106] UML profiles: [1, 6, 95, 109]
meta-meta-descriptive	QMT belongs to the M3 (meta-meta-) level; no solutions on this level explicitly addressing the representation of quality metamodels	generic: MOF-based [17, 18], formal notions [58]

Abstraction levels for ontologies. For QOT, this dimension reflects the level of abstraction of the concepts described with a quality ontology. It differs from the QMT abstraction level as it refers to the concepts being described – not the description approach (all ontologies of different levels can be defined using the same ontology language). It has the same values as for QMT but with different semantics (Table 3).

Notes on evolution. An evolution of QCT along this dimension is not linear. For QMT, it mostly follows the path up the meta-pyramid from M1 to M2 levels [32, 57, p. 232], but particular M1-level solutions appear if no reuse of models is necessary. For QOT, the first known quality ontology [14] was of concrete kind, upper-level quality ontologies have been introduced later via generalization and are used more frequently now, though some concrete and mixed ontologies are available as well.

Table 3. *Abstraction level* scale (QOT)

Value	Meaning	Examples
concrete	QOT defines *quality ontology* directly describing real-world phenomenon of quality (via particulars; with concepts for performance, reliability etc)	[3, 14]
meta-descriptive	QOT defines *quality upper ontology* describing the concepts (universals) for concrete quality ontologies (e.g. by defining concepts for quality characteristic, quality metric, describing their relationships etc.)	[13, 40, 57, 76]
meta-meta-descriptive	QOT reflects the *ontology language*; we follow [7] in that it such language is a modeling language described on M3 level.	OWL [76], formal notations [57, 79]

3.2 Quality Properties Support Dimensions

These dimensions correspond to the degree of completeness of reflecting properties of the *QC* set of qualities (1) and the *QR* relationships among the elements of this set. We distinguish dimensions reflecting *QC structure*, *measurability* of its elements, and *dependencies* inside *QC* (omitting some others e.g. *prioritization* support).

For brevity, we list only few qualitative values per dimension; later we will show that to perform QCT evaluation based on this classification, it will be necessary to quantify these values to make the dimensions measurable.

3.2.1 Structural Complexity

Purpose: to reflect the organization of *QC* as defined by internal relationships $qr \in QR$ (Table 4).

Table 4. *Structural complexity* scale

Value	Meaning	Examples
single quality	QCT describes a single quality characteristic; often such techniques conceptualize the domain related to this characteristic	security [39, 106] etc. reliability [50, 116] etc. performance [74, 109]
set	QCT does not describe any structure for *QC*	[57, 66]
taxonomy	QCT describes a hierarchy of $qc \in QC$ with more general characteristics at higher level	Early QCT [15, 80], standards [53, 54] etc.
graph	QCT deviates from pure taxonomy in *QC* structure e.g. by taking into account overlapping characteristics	[16, 37]

Notes on evolution. The treatment of this issue has evolved into taking into account overlapping quality characteristics. This evolution has been driven by the increasing level of understanding of the real-world notion of quality, which goes beyond seeing it as a simple taxonomy. Older pure-taxonomy solutions are still widespread especially as they are parts of a standard e.g., ISO/IEC 9126.

3.2.2 Quality Measurability

Purpose: to reflect the degree of supplementing $qc \in QC$ with quality metrics i.e. the availability of the shared quality spaces [57] connected to these characteristics (Table 5).

Table 5. *Quality measurability* scale

Value	Meaning	Examples
none	Purely qualitative QCT; they are usually resulted from establishing top-level quality characteristics first	Early QCT [15, 80] etc.
basic	QCT includes only generic support for quality metrics (e.g. by establish the "metric" concept but not elaborating the concepts describing the way of calculating these metrics); usually this is true for the solutions that describe the quality but are not intended for its evaluation or prediction	[18, 40, 99]
complete	QCT includes more extensive support for quality metrics (e.g. conceptualizing the way of calculating their values)	[13, 81, 107]

Notes on evolution. Initially, QCTs were qualitative in nature; the evolution of their treatment resulted in supplementing the conceptualizations with extensive sets of quantitative metrics. Quality model standards [54, 55] also include metrics. Currently, there are some arguments in favor of the prohibition of using purely qualitative models at least for some problem areas, see the discussion in [32, 42, 56].

3.2.3 Quality Dependencies Support

Purpose: to reflect the degree of taking into account the interdependencies among quality characteristics defined by relationships $qr \in QR$ (Table 6).

Table 6. *Quality dependencies support* scale

Value	Meaning	Examples
none	No support for interdependencies	[15]
basic	QCT deals with interdependencies among quality characteristics (e.g. how they influence each other) in qualitative fashion (i.e. via such values as "supports" or "neutral")	[13, 26, 99]
complete	QCT includes quantitative and qualitative *QR* treatment	Jureta et al. [58]

Notes on evolution. The support for interdependencies inside QC is not widely available in QCT. Only recently a solution [58] appeared with their complete support.

3.3 Quality Perception Support Dimensions

These dimensions are related to the fact that different stakeholders perceive quality differently. Their values reflect the properties of the *QU* set (*QU-relationships*) (1).

3.3.1 Stakeholder Dependency

Purpose: to reflect a QCT's ability to support QU-relationships (Table 7).

Table 7. *Stakeholder dependency* scale

Value	Meaning	Examples
none	No QU-relationships: single-person view on quality (also implies "none" value for all other dimensions from this category)	[40, 54] etc.
basic	QU-relationships are supported but without private quality spaces (conceptualizations include the notion of stakeholder, but do not allow per-stakeholder metrics)	[48, 65]
complete	QU-relationships are accompanied by private quality spaces (allowing per-stakeholder measuring of quality)	[25, 26, 57]

Notes on evolution. The earliest QMT did not support QU-relationships at all. Later evolution led to including this support [48] and soon to accompanying it with private quality spaces via introducing the concept of softgoal in the NFR framework [26].

3.3.2 Speech Mode Support

Purpose: to reflect a QCT's ability to support different speech modes in QU-relationships; here we group values by the type of the artefact produced (Table 8).

Table 8. *Speech mode support* scale

Value	Meaning	Examples
partial	QCT supports a subset of the available speech modes:	
	goals: directive mode	[25, 61, 117] etc.
	assumptions/attributes: declarative/assertive mode	[62]
	evaluations: expressive mode	[24, 107] etc.
complete	QCT supports all speech modes	[57]

Notes on evolution. First supported speech mode was a directive one used to define goals (formulated as quality constraints [14, 71] or softgoals [26]). Later, e.g. after introducing evaluation-related QCT [107] (see Section 3.5.1) the support for other modes started to appear; a complete solution is now only available in CORE [57].

3.4 Quality Usage Support Dimensions

These dimensions are related to the usage of the quality concept to model quality requirements, quality attributes, or other concepts related to the quality of the system-to-be. They reflect a QCT's ability to support the properties of the *QF* set (1) i.e. the relationships between a conceptualization of quality and a conceptual model of the system functionality *(QF-relationships).*

The challenge here lies in a fact that currently QF-relationships are mostly treated implicitly, their descriptions are obscured by the descriptions of the particular requirement- or attribute-related modeling solutions; as a result, we know of no attempts for categorizing them or investigating their evolution.

3.4.1 Usage Support

Purpose: to reflect an objective of using a quality conceptualization via QF-relationships (Table 9).

Table 9. *Usage support* scale

Value	Meaning	Examples
none	No support for QF-relationships (also implies "none" value for all other dimensions from this category)	[15, 40, 54, 80] etc.
present	QCT reflects the relationships aimed at modeling the quality requirements for the system-to-be or the quality attributes for the implemented system	requirements [3, 25, 57,, 61, 82] etc. attributes [62, 85, 111] etc.

Notes on evolution. The earliest QMT did not support QF-relationships; they just described the quality phenomenon and were not concerned about its usage. Later, the general trend was to first include and then extend this support in both QMT and QOT.

3.4.2 Usage Explicitness

Purpose: to reflect a degree of explicitness in describing the usage of a conceptualization via $qf \in QF$ (Table 10).

Table 10. *Usage explicitness* scale

Value	Meaning	Examples
explicit	QCT reflects the relationships defined explicitly e.g. as associations between quality and functionally-related concepts	[3, 13, 42, ch. 18, 61] etc.
implicit	QCT reflects the relationships defined implicitly e.g. by specifying *the criteria for participation*	[29, 68, 83]

Implicit solutions employ aspect-oriented techniques [98] by defining criteria (*pointcuts* in aspect-oriented terminology) describing sets of functionality-related concepts (e.g. classes or methods) to be involved into relationships with particular quality-related concepts. In this case it is necessary to express such criteria at an appropriate level of abstraction [38, 104] e.g. using semantic notions [22, 102].

Notes on evolution. Initially, QF-relationships were of explicit kind, recently the techniques for expressing them implicitly started to appear, but the former are still in a wider use and there are doubts [61] if the introduced complexity is justifiable.

3.4.3 Usage Target

Purpose: to reflect the properties of the system-to-be functional units used in QF-relationships (Table 11).

Table 11. *Usage target* scale

Value	Meaning	Examples
static	QF-relationships depend on a static decomposition of the connected model (e.g. by targeting classes)	[42, ch. 18, 117]
dynamic	QF-relationships depend on a behavior of the system-to-be reflected in its model (e.g. by targeting activities, events or states)	[23, 33, 95]
all	QF-relationships target both static elements and behavior	[1, 83]

Notes on evolution. Currently we observe the move from static to dynamic relationships as more knowledge about the relations between quality and behaviour is gained.

3.5 Applicability Dimensions

These dimensions are related to the external view on the quality conceptualization by describing a QCT's applicability for the particular classes of problems.

3.5.1 Application Goal

Purpose: to reflect the goal of applying the solution produced by this QCT, we follow [32, 112] in defining its values (Table 12).

Table 12. *Application goal* scale [32, 112], ***"all"*** value is also available

Value	Meaning	Examples
definition	QCT defines the notion of quality	[15, 54, 79] etc.
assessment	QCT is used for quality assessment	review is in [107]
prediction	QCT is used for quality prediction	[94, 114] etc.

Notes on evolution. Early QCT were mostly of a "definition" kind. Now, techniques of all three kinds are being actively developed in parallel (with researchers often unaware of each other's work [32]) with definition models being the most widespread.

3.5.2 Process Stage Dependency

Purpose: to reflects the stage(s) of the software process the particular QCT to be applied for [112] (Table 13).

Table 13. *Process stage dependency* scale

Value	Meaning	Examples
none	QCT is generic enough to be applied at different software process stages	standards [54, 55], etc.
stage-specific	QCT is specific for a particular process stage(s)	
	requirements engineering	[36, 45, 57] etc.
	architectural design	[2, 29, 70, 108] etc.
	software implementation	[44]
integrated	QCT produces integrated quality conceptualizations to be used in MDA-like environment	[5, 68] etc.

Notes on evolution. Early QCT were mostly of a stage-independent kind; currently stage-specific solutions outnumber the former due to a wider field of application (with requirement engineering stage as the most frequent target). Very few QCT target software implementation. The trend is also toward integrated MDA-based techniques.

3.5.3 Application Category Dependency

Purpose: to reflect the classification of the software solutions QCT to be applied for (Table 14). *Notes on evolution:* early QCT were category-independent, over time, the average degree of category dependency tends to increase alongside the amount of gained knowledge about the quality of the particular software classes.

Table 14. *Application category dependency* scale

Value	Meaning	Examples
category-specific	QCT produces conceptualization suited for a particular software category(ies)	
	service-oriented systems	[45, 58, 65, 73, 100] etc.
	real-time systems	[6, 43, 109]
	component software	[4, 19, 20, 24] etc.
	web-based systems	[18, 51, 77, 86] etc.
none	QCT does not target any particular software category (*category-independent*)	standard-related QCT [17, 28, 54, 55, 105], etc.

3.5.4 Domain Dependency

Purpose: to reflect the degree of domain-dependency possessed by the QCT (Table 15). *Notes on evolution:* early QCT were domain-independent. Currently, solutions of this kind are still most widely available, domain-specific solutions are not often published in software engineering literature; recently, domain-customizable solutions started to appear instead.

Table 15. *Domain dependency* scale

Value	Meaning	Examples
domain-specific	QCT produces ad-hoc conceptualization specific for the particular application domain	e-government [76] e-commerce [12, 103] education [9, 47, 92]
domain-customizable	QCT allows to develop domain-dependent conceptualizations by customizing a generic "template"	[21, 40, 64, 101]
none	QCT produces conceptualization not specific to any particular domain (*domain-independent*)	all except above

4 Towards a QCT Evaluation Framework

In this section, we discuss the steps that could be taken to turn the proposed set of classification dimensions into the set of evaluation criteria to be used for selecting the best QCT for the problem at hand. Full coverage of this issue will be the target for subsequent publications.

4.1 Evaluating Quality of Quality Conceptualizations

To make the proposed set of QCT evolution dimensions a part of a QCT evaluation/comparison framework we need to take into account the fact that our proposed dimension *scale values* are just qualitative "tags" established for marking the evolution milestones. The problem with using such values to organize QCT comparison and evaluation is that quality spaces [46, 57] associated with our dimensions are rudimentary (the choice of values is arbitrary, there is no order or distance defined etc).

We see several ways to overcome this limitation. First of all, it is possible to drop quality spaces altogether by establishing so-called "descriptive mode" comparison framework such as the one proposed by Babar et al. for software architecture evaluation methods [8]. Such framework defines only dimensions; for every QCT under evaluation, it is necessary to elaborate narrative descriptions of its positions related to these dimensions. For particular QCT, such descriptions can go into great detail (see the description of ATAM according to the Babar et al. framework [63]). In addition, for the specific classes of projects, it is possible to describe the most suitable QCT position w.r.t. every dimension. This way, one would document a set of useful (though informal) guidelines for selecting QCT suitable to the problem at hand.

Other approaches involve quantifying the dimensions turning them into *quality evaluation dimensions* to establish QCT quality model of the assessment type ("*quality of quality*" model - QM^2) and applying this model to the problem at hand. Several dimension quantification techniques can be used to aid in solving this problem.

1. Following bottom-up quality modeling approach [37] by defining measurable QCT quality-bearing properties and connecting our dimensions to these properties. To do so, we can adapt the metrics from e.g. general evaluation frameworks for conceptual models [75, 84, 91] and ontologies [30, 88]. For example, one can take the quality metamodel defined by the particular QMT and calculate the completeness of its coverage of the QU-relationships by modifying the technique from [35]. Next, we connect this property (and others) to the "*stakeholder dependency*" dimension by using the weighting approach of [10, 100] or by other means.
2. Quantifying the dimensions by means of multiple criteria decision making (MCDM) technique e.g. Analytic Hierarchy Process (AHP) [2, 12, 72]. This way, the positions of the particular QCT (decision alternatives) related to a particular dimension are compared using AHP pairwise comparison technique (*how better is QCT_A to QCT_B w.r.t. stakeholder dependency?*); as a result, the values for these relative positions are calculated for every dimension. This usually involves experts.
3. Applying fuzzy logic to the problem by converting dimensions into linguistic variables and establishing appropriate membership functions.

After the dimensions are quantified (we obtained the figures for the positions of QCT alternatives), we can again apply a MCDM technique (e.g. AHP) to take into account the relative importance of the dimensions (quality criteria) for the problem at hand (a generic procedure of this kind is described e.g. in [60]).

4.2 Engineering Quality Conceptualizations for Situational Methods

It seems feasible to integrate the customizable (or selectable) QCTs as method components into the situational method engineering (SME) framework [52] aimed at establishing the *situational conceptual modeling process* [11, 110] tailored to the problem at hand. Such components could be called *QCT method chunks* [93], it is also possible to turn them into *QCT method services* according to a method-as-a-service idea [34, 96]. QCT evaluation framework defined as outlined above could be integrated into this process as a means of selecting the appropriate chunks in a way similar to described in [69]; in method-as-a-service case it could be also feasible to treat QCT qualities as QoS and employ QoS-based service selection techniques (e.g. [115]).

The idea of introducing quality definitions into the software process by means of SME was first proposed by Saeki [97] who presented a SME framework aimed at embedding metrics into the software process activities to make produced development artifacts measurable. This approach automates assessing the quality ***of*** conceptual models (such as class diagrams complexity and readability) or quality characteristics of the system-to-be that could be measured at development time based on these artifacts (such as measuring modifiability based on attributes of the use case diagram). Our goal is to facilitate extending the existing methods with QCT method chunks/services aimed at dealing with quality ***in*** conceptual models such as making available pre-selected quality conceptualizations suitable for the problem at hand.

5 Related Work

Despite the fact that QCT received a lot of attention from the conceptual modeling research community, we know of no attempts to review all types of these techniques (including *both* quality models and quality ontologies) using a unified set of classification/evolution dimensions.

Available reviews of quality models are not numerous [20, 27, 32]. The closest to ours is the approach by Wagner at al [113] which proposed six classification dimensions to some degree similar to ours (purpose, view, attribute, phase, technique, and abstractness), they, however, did not explore this issue in detail as it was not the main purpose of that work, in particular, no evolution of these techniques was investigated.

There are also QCT reviews limited to particular evolution dimensions. For example, numerous reviews of quality requirement conceptualizations [25, 61, 108] are, from the point of view of this paper, detailed treatments of just a subset for a speech mode dimension possessing "goal" (directive) value. Another example is the review by Tian [107] dedicated to evaluation models.

Quality ontologies received even less attention. Actually, we know of no attempts to perform dedicated review of these techniques except some limited reviews in the "related work" sections of the papers dedicated to particular techniques [56, 57, 59]. We need, however, to mention a paper [78] reviewing different approaches of representing quality by means of formal ontology without explicit connections to software.

6 Conclusions and Future Work

In this paper, we proposed an extended set of evolution dimensions for quality conceptualization techniques which can also serve as criteria for their classification; such classification could help with a confusion related to the abundance of QCT and lacking clear definitions of the limits of their applicability. We based these dimensions on a QCT's ability to represent software quality defined according to its treatment in CORE [57] and DOLCE [79] ontologies, on the degree of support for stakeholder perception of quality and the connections to the functional components of the system-to-be, and on a QCT's applicability to the particular classes of problems. We outlined the ways of establishing a QCT evaluation framework based on the proposed classification criteria. This framework could help to resolve the problem of selecting QCT best suited for the project at hand.

We plan to apply our classification framework to all QCT known to us and make the results of this classification available to public; we expect that some adjustments to the set of dimensions will need to be made in the course of such application. On the other hand, we plan to elaborate the evaluation framework and QCT method chunks/services outlined in Section 4.

References

1. Aagedal, 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, Technical Report TR 2004-06-01. OMG (2004)
2. 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.) Proc. 27th International Conference on Software Engineering (ICSE 2005), pp. 244–253. ACM, New York (2006)
3. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: ElicitO: A Quality Ontology-Guided NFR Elicitation Tool. In: Sawyer, P., Heymans, P. (eds.) REFSQ 2007. LNCS, vol. 4542, pp. 306–319. Springer, Heidelberg (2007)
4. Alvaro, A., de Almeida, S.: A software component quality framework. ACM SIGSOFT Software Engineering Notes 35, 1–18 (2010)
5. Ameller, D., Gutierrez, F., Cabot, J.: Dealing with non-functional requirements in model-driven development. Report ESSI-TR-10-05. Universitat Politècnica de Catalunya (2010)
6. Apvrille, L., Courtiat, J.-P., Lohr, C., de Saqui-Sannes, P.: TURTLE: A Real-Time UML Profile Supported by Formal Validation Toolkit. IEEE Transactions on Software Engineering 30, 473–487 (2004)
7. Aßmann, U., Zschaler, S.: Ontologies, Meta-models, and the Model-Driven Paradigm. In: Calero, C., Ruiz, F., Piattini, M. (eds.) Ontologies for Software Engineering and Software Technology, pp. 255–279. Springer, Heidelberg (2006)
8. Babar, M.A., Zhu, L., Jeffery, R.: A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In: Proc. 15th Australian Software Engineering Conference (ASWEC 2004), pp. 309–318. IEEE Press, New York (2004)
9. Bajnaid, N., Cogan, B., Al-Nuaim, H.: Software quality ontology for teaching: a development methodology's issues. In: Proc. 5th International Innovations in Information Technology (IIT 2008), pp. 352–356 (2008)
10. Bansiya, J., Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering 28, 4–17 (2002)
11. Becker, J., Janiesch, C., Pfeiffer, D., Seidel, S.: Evolutionary method engineering: towards a method for the analysis and conception of management information systems. In: Proc. 12th Americas Conference on Information Systems (AMCIS 2006), pp. 3686–3697 (2006)
12. Behkamal, B., Kahani, M., Akbari, M.K.: Customizing ISO 9126 quality model for evaluation of B2B applications. Information and Software Technology 51, 599–609 (2009)
13. Bertoa, M., Vallecillo, A., Garc¡a, 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, Heidelberg (2006)
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, New York (1978)
16. Buglione, L., Kececi, N., Abran, A.: An Integrated Graphical Assessment for Managing Software Product Quality. In: Proc. 12th International Software Quality Conference (ICSQ 2002), Ottawa, Canada (2002)
17. Burgués, X., Franch, X., Ribó, J.M.: A MOF-Compliant Approach to Software Quality Modeling. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 176–191. Springer, Heidelberg (2005)

18. Cachero, C., Calero, C., Poels, G.: Metamodeling the Quality of the Web Development Process' Intermediate Artifacts. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 74–89. Springer, Heidelberg (2007)
19. Carvalho, F., Meira, S.R.L., Xavier, E., Eulino, J.: An Embedded Software Component Maturity Model. In: Choi, B. (ed.) Proc. 9th International Conference on Quality Software (QSIC 2009), pp. 426–431. IEEE Press, New York (2009)
20. Carvallo, J.P.: Systematic Construction of Quality Models for COTS-Based Systems. PhD Thesis. Universitat Politècnica de Catalunya, Barcelona (2005)
21. Carvallo, J.P., Franch, X., Quer, C.: Building and Using Quality Models for Complex Software Domains. Technical Report LSI-03-28-R. Universitat Politècnica de Catalunya, Barcelona (2007)
22. Cazzola, W., Jezequel, J.M., Rashid, A.: Semantic join point models: Motivations, notions and requirements. In: Proc. Software Engineering Properties of Languages and Aspect Technologies Workshop (SPLAT 2006) (2006)
23. Charfi, A., Müller, H., Mezini, M.: Aspect-Oriented Business Process Modeling with AO4BPMN. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 48–61. Springer, Heidelberg (2010)
24. Choi, Y., Lee, S., Song, H., Park, J., Kim, S.H.: Practical S/W Component Quality Evaluation Model. In: Proc. 10th IEEE International Conference on Advanced Communication Technology (ICACT 2008), pp. 259–264. IEEE Press, New York (2008)
25. Chung, L., do Prado Leite, J.: On Non-Functional Requirements in Software Engineering. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 363–379. Springer, Heidelberg (2009)
26. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston (1999)
27. Cote, M., Suryn, W., Georgiadou, E.: Software Quality Model Requirements for Software Quality Engineering. In: Proc. 14th International Conference on Software Quality Management (SQM 2006), pp. 31–50 (2006)
28. Cote, M.A., Suryn, W., Georgiadou, E.: In search for a widely applicable and accepted software quality model for software quality engineering. Software Quality Journal 15, 401–416 (2007)
29. Dai, L., Cooper, K.: Modeling and analysis of non-functional requirements as aspects in a UML based architecture design. In: Proc. 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2005), pp. 178–183. IEEE Press, New York (2005)
30. Davies, I., Green, P., Milton, S., Rosemann, M.: Analyzing and comparing ontologies with meta-models. In: Krogstie, J., Halpin, T., Siau, K. (eds.) Information Modeling Methods and Methodologies, pp. 1–16. Idea Group, Hershey (2005)
31. de Boer, R.C., van Vliet, H.: QuOnt: an ontology for the reuse of quality criteria. In: Proc. ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK 2009), pp. 57–64. IEEE Press, New York (2009)
32. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: Proc. 7th International Workshop on Software Quality (WoSQ 2009), pp. 9–14. IEEE Press, New York (2009)
33. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proc. IEEE International Conference on Software Maintenance (ICSM 2007), pp. 184–193. IEEE Press, New York (2007)
34. Deneckere, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In: Halpin, T., Krogstie, J., Proper, E. (eds.) Proc. 13th Intl Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD 2008). CEUR-WS, vol. 337, pp. 80–96 (2008)

35. Dieste, O., Genero, M., Juristo, N., Moreno, A.M.: A Proposal of A Measure Of Completeness For Conceptual Models. In: Piattini, M., Genero, M., Calero, C. (eds.) Metrics for Software Conceptual Models, pp. 19–57. Imperial College Press, London (2005)
36. Dinkel, M., Baumgarten, U.: Modeling nonfunctional requirements: a basis for dynamic systems management. ACM Software Engineering Notes 30, 1–8 (2005)
37. Dromey, R.G.: Cornering the Chimera. IEEE Software 13, 33–43 (1996)
38. Eichberg, M., Awasthi, P., Ostermann, K.: Pointcuts as functional queries. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 366–381. Springer, Heidelberg (2004)
39. Ekelhart, A., Fenz, S., Klemen, M., Weippl, E.: Security Ontology: Simulating Threats to Corporate Assets. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 249–259. Springer, Heidelberg (2006)
40. Falbo, R.A., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engineering. In: Proc. 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), pp. 351–358. ACM Press, New York (2002)
41. Fenton, N., Pfleeger, S.L.: Software metrics: a rigorous and practical approach. PWS Publishing, Boston (1997)
42. Firesmith, D.G., Capell, P., Hammons, C.B., Latimer, D.W., Merendino, T.: The Method Framework for Engineering System Architectures. Auerbach Pubs, Boca Raton (2008)
43. Flake, S., Müller, W.: A UML profile for real-time constraints with the OCL. In: Li, J., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 179–195. Springer, Heidelberg (2002)
44. Fuentes, L., Sanchez, P.: Towards executable aspect-oriented UML models. In: Proc. AOM Workshop at AOSD 2007, pp. 28–34. ACM, New York (2007)
45. Galster, M., Bucherer, E.: A taxonomy for identifying and specifying non-functional requirements in service-oriented development. In: Proc. IEEE Congress on Services (SERVICES 2008) - Part I, pp. 345–352. IEEE Press, New York (2008)
46. Gärdenfors, P.: Conceptual Spaces: A Geometry of Thought. MIT Press, Cambridge (2000)
47. Gasparini, I., Lichtnow, D., Pimenta, M.S., de Oliveira, J.P.M.: Quality Ontology for Recommendation in an Adaptive Educational System. In: Proc. International Conference on Intelligent Networking and Collaborative Systems (INCOS 2009), pp. 329–334. IEEE Press, New York (2009)
48. Gilb, T.: Principles of Software Engineering Management. Addison Wesley, Reading (1988)
49. Glinz, M.: On Non-Functional Requirements. In: Proc. 15th IEEE International Conference on Requirements Engineering (RE 2007), pp. 21–26. IEEE Press, New York (2007)
50. Grassi, V., Mirandola, R., Sabetta, A.: From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In: Proc. 5th International Workshop on Software and Performance (WOSP 2005), pp. 25–36. ACM Press, New York (2005)
51. Hakkarainen, S., Strasunskas, D., Hella, L., Tuxen, S.: Choosing appropriate method guidelines for web-ontology building. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 270–287. Springer, Heidelberg (2005)
52. Henderson-Sellers, B., Ralyte, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science 16, 424–478 (2010)
53. IEEE 1061-1998: IEEE Standard for Software Quality Metrics Methodology. IEEE Press, New York (1998)
54. ISO/IEC 9126-1, Software Engineering – Product Quality – Part 1:Quality model. International Organization for Standardization (2001)
55. ISO/IEC FCD 25010: Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Organization for Standardization (2010)

56. Jureta, I.: Essays in Information Management: Contributions to the Modeling and Analysis of Quality in Information Systems Engineering, PhD Thesis. Department of Business Administration. University of Namur (2008)
57. Jureta, I., Mylopoulos, J., Faulkner, S.: A core ontology for requirements. Applied Ontology 4, 169–244 (2009)
58. Jureta, I.J., Herssens, C., Faulkner, S.: A Comprehensive Quality Model for Service-Oriented Systems. Software Quality Journal 17, 65–98 (2009)
59. Kabilan, V., Johannesson, P., Ruohomaa, S., Moen, P., Herrmann, A., Ehlfeldt, R.M., Weigand, H.: Introducing the Common Non-Functional Ontology. In: Gonçalves, R.J., Müller, J.P., Mertins, K., Zelm, M. (eds.) Enterprise Interoperability II, pp. 633–645. Springer, Heidelberg (2007)
60. Kaschek, R., Pavlov, R., Shekhovtsov, V., Zlatkin, S.: Characterization and tool supported selection of business process modeling methodologies. In: Abramowicz, W., Mayr, H.C. (eds.) Technologies for Business Information Systems, pp. 25–37. Springer, Heidelberg (2007)
61. Kassab, M., Ormandjieva, O., Daneva, M.: An Ontology Based approach to Non-Functional Requirements Conceptualization. In: Proc. 4th International Conference on Software Engineering Advances (SEA 2009), pp. 299–308. IEEE Press, New York (2009)
62. Kayed, A., Hirzalla, N., Samhan, A.A., Alfayoumi, M.: Towards an Ontology for Software Product Quality Attributes. In: Proc. 4th International Conference on Internet and Web Applications and Services (ICIW 2009), pp. 200–204. IEEE Press, New York (2009)
63. Kazman, R., Bass, L., Klein, M., Lattance, T., Northrop, L.: A Basis for Analyzing Software Architecture Analysis Methods. Software Quality Journal 13, 329–355 (2005)
64. Khomh, F., Gueheneuc, Y.G.: DEQUALITE: Building Design-based Software Quality Models. In: Proc. SPAQU 2008 (2008)
65. Kim, E., Lee, Y.: Quality Model for Web Services 2.0. OASIS (2005)
66. Kim, H., Fox, M.S., Gruninger, M.: An Ontology of Quality for Enterprise Modelling. In: Proc. ETICE 1995, pp. 105–116. IEEE Press, New York (1995)
67. Kitchenham, B., Pfleeger, S.L.: Software quality: the elusive target. IEEE Software 13, 12–21 (1996)
68. Koellmann, C., Kutvonen, L., Linington, P., Solberg, A.: An aspect-oriented approach to manage QoS dependability dimensions in model driven development. In: Proc. 3rd International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2007), pp. 85–94. INSTICC Press (2007)
69. Kornyshova, E., Deneckere, R., Salinesi, C.: Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. In: Ralyte, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, pp. 64–78. Springer, Heidelberg (2007)
70. Krechetov, I., Tekinerdogan, B., Garcia, A., Chavez, C., Kulesza, U.: Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. In: Proc. AOM Workshop at AOSD 2006 (2006)
71. Krogstie, J.: Integrating the understanding of quality in requirements specification and conceptual modeling. ACM SIGSOFT Software Engineering Notes 23, 86–91 (1998)
72. Kumar, A., Grover, P.S., Kumar, R.: A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO). ACM SIGSOFT Software Engineering Notes 34, 1–9 (2009)
73. Lee, J.Y., Lee, J.W., Du Wan Cheun, S.D.K.: A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. In: Proc. SERMA 2009, pp. 261–266. IEEE, New York (2009)
74. Lera, I., Sancho, P.P., Juiz, C., Puigjaner, R., Zottl, J., Haring, G.: Performance assessment of intelligent distributed systems through software performance ontology engineering (SPOE). Software Quality Journal 15, 53–67 (2007)

75. Lindland, O.I., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. IEEE Software 11, 42–49 (1994)
76. Magoutas, B., Halaris, C., Mentzas, G.: An Ontology for the Multi-Perspective Evaluation of Quality in E-government Services. In: Wimmer, M.A., Scholl, J., Grönlund, Å. (eds.) EGOV. LNCS, vol. 4656, pp. 318–329. Springer, Heidelberg (2007)
77. Malak, G., Badri, L., Badri, M., Sahraoui, H.: Towards a multidimensional model for web-based applications quality assessment. In: Bauknecht, K., Bichler, M., Pröll, B. (eds.) EC-Web 2004. LNCS, vol. 3182, pp. 316–327. Springer, Heidelberg (2004)
78. Masolo, C., Borgo, S.: Qualities in formal ontology. In: Hitzler, P., Lutz, C., Stumme, G. (eds.) Proc. Workshop on Foundational Aspects of Ontologies (FOnt 2005), pp. 2–16 (2005)
79. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: The WonderWeb Library of Foundational Ontologies. WonderWeb Deliverable D18. Ontology Library (final). ISTC-CNR, Trento (2003)
80. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. NTIS (1977)
81. McQuillan, J.A., Power, J.F.: Towards the re-usability of software metric definitions at the meta level. In: ECOOP PhD Workshop (2006)
82. Mead, N.R., Hough, E.D., Stehney, T.R.: Security Quality Requirements Engineering (SQUARE) Methodology. Software Engineering Institute, Stanford (2005)
83. Meier, S., Reinhard, T., Seybold, C., Glinz, M.: Aspect-Oriented Modeling with Integrated Object Models. In: Modellierung 2006. LNI, vol. 82, pp. 129–144. GI, Bonn (2006)
84. Moody, D.L.: Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. Data & Knowledge Engineering 55, 243–276 (2005)
85. Morasca, S.: A probability-based approach for measuring external attributes of software artifacts. In: Proc. ESEM 2009, pp. 44–55. IEEE Press, New York (2009)
86. Nam, J.: Web portal quality. In: Proc. SOLI 2009, pp. 163–168. IEEE Press, New York (2009)
87. Neuhaus, F., Grenon, P., Smith, B.: A formal theory of substances, qualities, and universals. In: Varzi, A., Vieu, L. (eds.) FOIS 2004, pp. 49–59. IOS Press, Amsterdam (2004)
88. Obrst, L., Ceusters, W., Mani, I., Ray, S., Smith, B.: The Evaluation of Ontologies Toward Improved Semantic Interoperability. In: Baker, C.J.O., Cheung, K.-H. (eds.) Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, pp. 139–158. Springer, Heidelberg (2007)
89. Olive, A.: Conceptual Modeling of Information Systems. Springer, Heidelberg (2007)
90. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice. Springer, Heidelberg (2007)
91. Piattini, M., Genero, M., Poels, G., Nelson, J.: Towards a Framework for Conceptual Modelling Quality. In: Piattini, M., Genero, M., Calero, C. (eds.) Metrics for Software Conceptual Models, pp. 1–18. Imperial College Press, London (2005)
92. Plaza, I., Igual, R., Marcuello, J.J., Sanchez, S., Arcega, F.: Proposal of a Quality Model for Educational Software. In: Proc. EAEEIE 2009, pp. 1–6 (2009)
93. Ralyte, J., Deneckere, R., Rolland, C.: Towards a generic model for situational method engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 95–110. Springer, Heidelberg (2003)
94. Riaz, M., Mendes, E., Tempero, E.: A systematic review of software maintainability prediction and metrics. In: Proc. ESEM 2009, pp. 367–377. IEEE Press, New York (2009)
95. Rodríguez, A., Fernández-Medina, E., Piattini, M.: Capturing Security Requirements in Business Processes Through a UML 2.0 Activity Diagrams Profile. In: Roddick, J., Benjamins, V.R., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R.A., Grandi, F., Han, H., Hepp, M., Lytras, M.D., Mišić, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (eds.) ER Workshops 2006. LNCS, vol. 4231, pp. 32–42. Springer, Heidelberg (2006)

96. Rolland, C.: Method engineering: towards methods as services. In: Wang, Q., Pfahl, D., Raffo, D.M. (eds.) ICSP 2008. LNCS, vol. 5007, pp. 10–11. Springer, Heidelberg (2008)
97. Saeki, M.: Embedding metrics into information systems development methods: An application of method engineering technique. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 374–389. Springer, Heidelberg (2003)
98. Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M.: Towards a common reference architecture for aspect-oriented modeling. In: Proc. AOM Workshop at AOSD 2006. ACM, New York (2006)
99. Shekhovtsov, V.A., Kop, C., Mayr, H.C.: Capturing the Semantics of Quality Requirements into an Intermediate Predesign Model. In: Hesse, W., Oberweis, A. (eds.): SIGSAND-EUROPE 2008 Symposium. LNI, vol. P-129, pp. 25–37. GI, Bonn (2008)
100. Shim, B., Choue, S., Kim, S., Park, S.: A Design Quality Model for Service-Oriented Architecture. In: Proc. APSEC 2008, pp. 403–410 (2008)
101. Sibisi, M., van Waveren, C.C.: A process framework for customising software quality models. In: Proc. AFRICON 2007, pp. 1–8 (2007)
102. Soeldner, G., Kapitza, R., Schober, S.: AOCI: ontology-based pointcuts. In: Proc. 8th Workshop on Aspects, Components, and Patterns for Infrastructure Software, pp. 25–30. ACM, New York (2009)
103. Stefani, A., Xenos, M.: E-commerce system quality assessment using a model based on ISO 9126 and Belief Networks. Software Quality Journal 16, 107–129 (2008)
104. Stein, D., Hanenberg, S., Unland, R.: Query Models. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 98–112. Springer, Heidelberg (2004)
105. Suryn, W., Abran, A., Laporte, C.: An integrated life cycle quality model for general public market software products. In: Proc. BSI 2004, pp. 5–7 (2004)
106. Susi, A., Perini, A., Mylopoulos, J.: The Tropos Metamodel and its Use. Informatica 29, 401–408 (2005)
107. Tian, J.: Quality-Evaluation Models and Measurements. IEEE Software 21, 84–91 (2004)
108. Tsadimas, A., Nikolaidou, M., Anagnostopoulos, D.: Handling non-functional requirements in Information System Architecture Design. In: Proc. SEA 2009, pp. 59–64. IEEE Press, New York (2009)
109. UML Profile for Schedulability, Performance, and Time, version 1.0. OMG (2003)
110. van de Weerd, I., Brinkkemper, S.: Meta-modeling for situational analysis and design methods. In: Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, pp. 35–54. IGI Global, Hershey-New York (2009)
111. Wada, H., Suzuki, J., Oba, K.: Modeling Non-Functional Aspects in Service Oriented Architecture. In: Proc. SCC 2006, pp. 222–229. IEEE Press, New York (2006)
112. Wagner, S., Deissenboeck, F.: An integrated approach to quality modelling. In: Proc. WoSQ 2007. ACM, New York (2007)
113. Wagner, S., Lochmann, K., Winter, S., Goeb, A., Klaes, M.: Quality models in practice: A preliminary analysis. In: Proc. ESEM 2009, pp. 464–467. IEEE Press, New York (2009)
114. Wu, C., Lin, H.L.: Integrating fuzzy theory and hierarchy concepts to evaluate software quality. Software Quality Journal 16, 263–276 (2008)
115. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30, 311–327 (2004)
116. Zhou, J., Niemelä, E., Evesti, A.: Ontology-Based Software Reliability Modelling. In: Proc. SSVM 2007, pp. 17–31 (2007)
117. Zhu, L., Gorton, I.: UML Profiles for Design Decisions and Non-Functional Requirements. In: Proc. ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK 2007). IEEE Press, New York (2007)