V.Shekhovtsov, H.C.Mayr, C.Kop: Harmonizing the Quality View of Stakeholders, Chapter 3. In: Mistrik, I., Bahsoon, R., Eeles, R., Roshandel, R., Stal, M. (eds.): Relating System Quality and Software Architecture. Morgan-Kaufmann (Elsevier imprint), 2014, pp. 41-73

CHAPTER

Harmonizing the Quality View of Stakeholders

3

Vladimir A. Shekhovtsov, Heinrich C. Mayr, and Christian Kop
Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

INTRODUCTION

A software development process typically has many stakeholders: the prospective end users, business actors, architecture and software designers, software developers and integrators, database engineers, test engineers, and others. All these should agree at an early stage on the same—or close enough—view of quality to make a software development process successful. Only then quality requirements can be determined reliably, implemented during software development, and thus met by the resulting system.

Failure to establish an appropriate common understanding of the prospective system's quality leads to the problems in the late stages of the development lifecycle that are very difficult and expensive to fix. In particular, the end users' opinions and expectations on software quality tend to be neglected or at least handled incompletely; as a result, the understanding of system quality becomes biased toward the view of the software developers—a problem known as "the inmates are running the asylum" (Cooper, 2004). This is supported by evidence of the practice of development: For example, a study of the current practice of the Quality Aware Software Engineering (QuASE) partner companies revealed that in the early stages of their software lifecycles, mainly usability is addressed, including handling performance and other issues is often postponed until system deployment. Clearly, that approach comes with negative consequences for all parties, because fixing the problems with unsatisfied stakeholder expectations on the late stages of the software process is difficult and expensive and may even lead to the failure of the whole project (Walia and Carver, 2009; Westland, 2002).

Thus it is necessary to harmonize the different stakeholder views of quality from the beginning throughout the complete development process, addressing requirements engineering via architectural design, specification to implementation, test, and rollout. This applies for both traditional structured and lightweight agile development processes.

For this purpose, an appropriate communication channel between the stakeholders has to be established, allowing them to discuss all relevant quality issues from their respective views.

This chapter presents a conceptualization of quality view harmonization and summarizes the related state of the art in research and development. In detail, we identify the dimensions of knowledge that are relevant for view harmonization, propose a set of concepts for each dimension, and align these where possible with concepts offered by other approaches. Our concepts will be grounded in the Unified Foundational Ontology (UFO) (Guizzardi, 2005) and materialized by applying them to the empirical data in a given software development venture. Thus, our conceptualization is intended to be a key

constituent of a common ontology of stakeholder perception and assessment (QuOntology). This ontology can be used to support the organization of the interaction process following the ontology-based software engineering paradigm.

The following dimensions of quality view harmonization will be addressed:

1. The quality of system under development (SUD) itself and the related aspects as well as their interrelations, in particular, the notion of quality assessment and the concepts utilized in the quality assessment activities.
2. The knowledge about the various stakeholders, the differences in their perception of quality, and the amount of knowledge about the software quality issues.
3. The knowledge about the activities of the harmonization process and the capabilities of the process participants needed for performing these activities.

The harmonization process will be dealt with on three levels:

1. Terminology harmonization: The stakeholders seek an agreement on the common quality-related terminology, the language constructs used for expressing expectations and opinions on quality. These can emerge in natural, domain-specific, or even formal language. The purpose of harmonization is to make them understandable to all stakeholders of the process.
2. View harmonization: The stakeholders seek an agreement on the sets of objects and the types of their qualities they are interested in, as well as the procedures of assessment.
3. Quality harmonization: The stakeholders seek an agreement on the evaluation schemes and the particular qualities they are interested in.

In this chapter, we concentrate on the view and quality harmonization.

The concepts presented here are consolidated by a study that is part of the ongoing QuASE project. We carried out detailed interviews with various members of the four industrial QuASE partners. The answers have been transcribed using the basic techniques of coding and conceptualization stages of the grounded theory (Adolph et al., 2011; Coleman and O'Connor, 2008). The obtained evidence then was combined with our own experience and with the body of knowledge inherent in the software quality standards such as ISO/IEC 25010 (ISO, 2011). A detailed description of this empirical study and an outline of the upcoming industrial application of the proposed conceptualization are presented in Section 3.4.

The chapter is structured as follows: In Section 3.1, we present the concepts we adopt from the UFO for grounding our approach. Section 3.2 introduces the conceptualization of quality assessments to be used as a foundation for the conceptualization of the harmonization process that is discussed in Section 3.3 on the three aforementioned levels supplemented with a running example of instantiating the proposed concepts. In Section 3.4, we describe the conducted empirical study and show how to apply the proposed concepts in practice. The chapter closes with a conclusion and directions for future research.

3.1 ADOPTED CONCEPTS OF THE UFO

We follow a multilevel approach to conceptualization where upper-level (more generic) concepts form the foundation for the concepts belonging to the lower levels. Such frameworks are called *foundational ontologies* in the literature. We start with making specific foundational choices, the most important

among them being the selection of the foundational framework providing upper-level generic concepts. Such ontologies define a set of basic concepts that reflect the particular world view. They can be extended and reused by the lower-level conceptual frameworks.

There are several foundational ontologies providing the concepts that could serve as a fundament for our venture: GFO (Herre, 2010), DOLCE (Masolo et al., 2003), and UFO (Guizzardi, 2005). For our purposes, we choose to adapt UFO; the reasons of that choice will be discussed at the end of this section.

Figure 3.1 depicts the subset of adapted UFO concepts that are useful for our conceptualization. The right-hand side represents the type level; according to extensional semantics, each of these concepts spans a set of instances that materialize the respectful type. The left-hand side represents the instance level and, as result, conceptualizes the materialization of the type level.

The root of the relevant UFO concept hierarchy is *Thing*, which generalizes the concepts *Individual*, *Type*, and *Abstract*. Individuals are Things that exist in the real world and possess a unique identity; Types can be seen as "templates of features," which may be materialized in a number of Individuals.

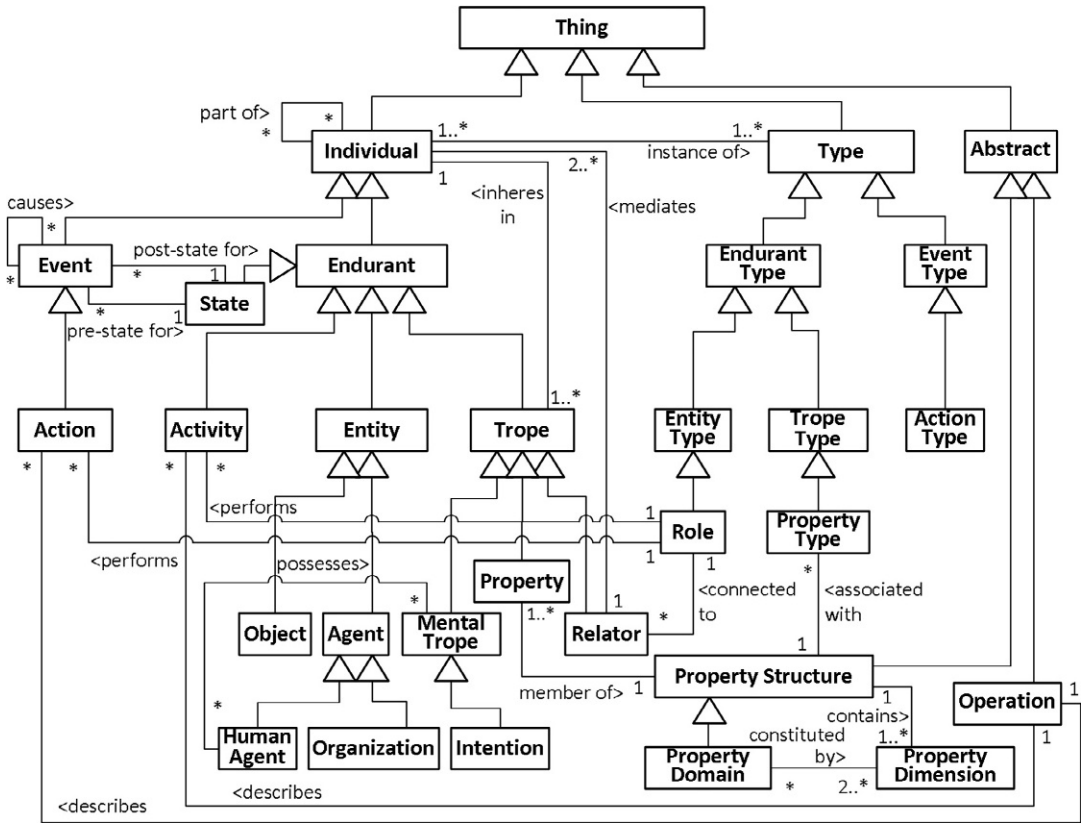

FIGURE 3.1

Fragment of the Unified Foundational Ontology.

Adapted from Guizzardi and Zamborlini (2013).

In other words, they are *instantiated* by Individuals; Abstracts are not materialized in the real world and therefore cannot be instantiated.

Individuals are specialized to *Endurants* or *Events*. Endurants are Individuals that exist, for a certain period, continuously in time. They are further categorized into *Entities* and *Tropes:* Entities are Individuals that may exist on their own. In contrast to this, Tropes can only exist in dependence with their "bearers," which are (instances of) Things. Entities are materializations (*instance_of*) Entity Types; Tropes are materializations of Trope Types. Following the object-oriented paradigm, Entities corresponds to objects, Tropes to attributes, and Entity Types to classes.

Relator is a special kind of Trope that defines a connection between two or more Individuals. As an example, think of a person assessing a particular quality. Following the object-oriented paradigm, Relators correspond to associations. One could question if the UFO explanation as a specialization of Trope is appropriate. In our opinion, it would be preferable to define Relators more generally as Entities; *inter alia*, this would allow for "objectified associations."

Roles are Entity Types associated to Entities where this association may change in time. In other words, the Individuals instantiating Roles are determined by their relationships (via Relators) to other Individuals.

Events are Individuals that happen at a particular point of time. Events are related to *States* that they initiate or terminate: Every Event is connected to its pre-State and post-State. *Actions* and *Activities* are performed by Individuals in a particular role by executing a (somehow abstractly defined) operation. As such, they represent reactions of a particular Role to a particular (triggering) Event. The difference between Actions and Activities is that the former are considered "timeless" (i.e., are performed in no or not interesting time), whereas the latter are supposed or expected to have a duration. Note the non-adapted UFO does not provide such strong orthogonal distinction between Event and State (no time vs. time), continued in the distinction between actions and Activities. However, for our purposes, we prefer such a clear separation of concepts.

The UFO conceptualization of quality is based on the theory of conceptual spaces by Gärdenfors (2000). This theory deals with the relationship between specific properties and the corresponding qualities as associated by human cognition. Within this context, UFO introduces the concept of *Property Type* (e.g., "color" or "response time") and associates it with a *Property Structure* consisting of *Property Dimensions* that describe the Individuals (*Properties*) of this Type (Property is a specialization of Trope inherent in Individual). Such structures may be simple; for example, the time dimension (associated with the Property Type "response time") defines a set of time values (i.e., "1 ms," "2 s," "1 min") because humans are accustomed to perceiving time in this way.

In other cases, it might be insufficient to use just such simple one-dimensional structures. For example, when dealing with a Property "color," we might want not only to use simple values like "red," "green" as perceived by humans. We rather might wish to represent colors by three values: hue (again possibly consisting of three RGB values), saturation, and brightness. As a result, for representing color, it is necessary to use a three-dimensional structure: a Property Domain represented by the set of defined triples (h, s, b). A collection of Property Domains forms, according to Gärdenfors, a conceptual space.

Property Structures are associated with Property Types; they define the concrete shape of the Individuals of these Property Types, otherwise known as the Properties. Property Types conceptualize quality characteristics and the relevant metrics as defined by such standards as ISO/IEC 9126 (ISO, 2001, 2003a,b, 2004) and ISO/IEC 25010 (ISO, 2011).

A specific subset of UFO (UFO-C) (Guizzardi et al., 2008) is devoted to conceptualizing the human social context. Among these concepts, we limited ourselves to adopting the concept of *Agent* describing active Entities that can respond to Events by performing Actions or Activities; passive Entities that cannot respond to Events are *Objects*. We further distinguish *Human Agents* and *Organizations*. UFO-C distinguishes *Mental Trope* as a subtype of Trope. Mental Tropes are related to Human Agents and reflect their capacity to respond to situations emerging in reality. In this chapter, we are interested in one category of such Mental Tropes, namely, *Intentions*. Intentions can be understood as "internal commitments" of the Agents to do something according to their will and, eventually, to start Activities or Actions.

3.1.1 Selection of the Foundational Ontology

We considered UFO, DOLCE, and GFO as the alternatives for the choice of the foundational ontology. Although all these ontologies provide the conceptual notions suitable for representing qualities of software artifacts and the process of their assessment by quality subjects, they apply different approaches to deal with these notions:

1. DOLCE treats all qualities as particulars (actually, this ontology positions itself as the "ontology of particulars"). There is no notion corresponding to the quality characteristic as understood by industry standards as a type (template) for the particular qualities, it also lacks concepts representing human social contexts.
2. GFO is an ontology that also lacks components related to human social context and the detailed representation of quality-related concepts.
3. UFO provides both particulars (Qualities) and generics (Quality Types); it also includes concepts related to quality and human social contexts.

As a result, we can justify the selection of UFO by the fact that it provides the most complete set of foundational concepts related to the representation of object and system qualities and the classification of such qualities.

As an alternative to using foundational ontology as the base for the presented set of concepts, it is possible to consider using meta-modeling approach such as SPEM (OMG, 2008) or ISO/IEC 24744 (ISO, 2007) targeting software development processes. These solutions, however, also do not provide concepts related to the representation of object and system qualities and the classification of such qualities.

3.2 ASSESSMENT AND RELATED CONCEPTS

Informally, quality assessment consists of associating Qualities (that conceptualize the assessment results) to Properties; this induces a ranking if the set of instances of the respective Quality Type is partially or fully ordered. Such ranking may be implicit if no other comparison object is explicitly mentioned.

Quality assessment is done by the Quality Subjects (stakeholders) according to their particular viewpoints, appraisals, preferences, and priorities—and thus requires a subsequent harmonization. As a consequence, quality assessment can be seen and formalized from two levels: the *specification*

level (dealing with specification of assessments) and the *execution level* (dealing with the execution of assessments).,Conceptualizing the specification level means to describe the way of performing assessments and the capabilities of the subjects who perform the assessments; conceptualizing the execution level means to capture the relevant aspects of assessment execution along its specification, including the assessed values, the results of the assessments, the time needed and consumed for assessments. In this section, we will describe the relevant concepts for both levels.

3.2.1 Specification-level concepts

Figure 3.2 depicts the specification-level concepts; our extensions to UFO are shaded in gray.

We start by introducing the concepts that reflect the human way to assess Properties, namely to associate *Quality Types* (e.g., "color quality" or "response time quality") with Property Types and to assign instances ("*Qualities*," e.g., "good," "bad," "interesting") of the former to the latter. Note that in the case of Property Structures, such associations and assignments may occur on different levels and include aggregations.

Software Artifacts are Objects that are produced in a software process; their qualities are of interest to the involved *Quality Subjects*. *Quality Assessment Functions* conceptualize the association of Qualities (or combinations of those) to Software Artifacts. More specifically, Qualities are assigned to the related Properties of the given artifact. This leads to the following definition:

Definition 1: Let SA be a Software Artifact, PT_{SA} the set of Property Types, instances of which are inherent in (components of) SA. For $pt \in PT_{SA}$, let I_{pt} be the set of instances of pt and I_{qt} the set of instances of the Quality Type qt associated with pt. Then a Property-Level *Quality Assessment Function* is formally defined as f_{SApt}: $I_{pt} \rightarrow I_{qt}$.

Note that, depending on the related Quality Structure definition, Qualities may be scalar as well as structured. In a concrete application, the Quality Assessment function may be both

- *ad hoc* defined by a Quality Subject in such a manner that he/she directly assigns Qualities to the Properties at hand.

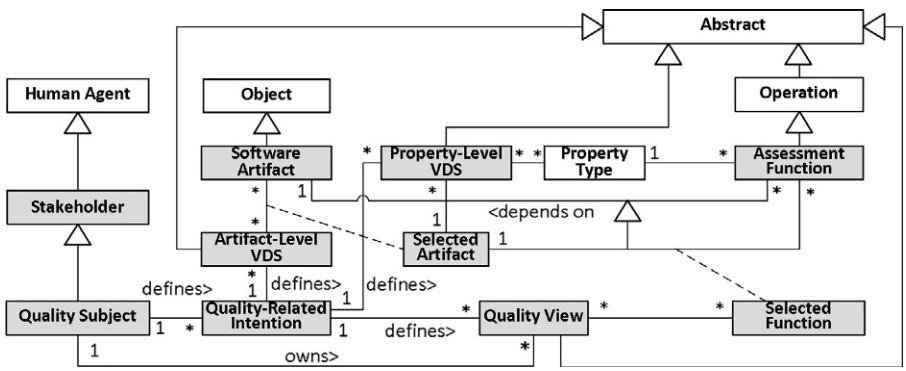

FIGURE 3.2

Specification-level concepts for Property-Level Assessments.

$f_{SA,response\ time}$	
$I_{response\_time}$	$I_{response\_time\_quality}$
<1 s	Good
[1s, 2s]	Satisfying
>2 s	Bad

FIGURE 3.3

Tabular definition of a Quality Assessment Function.

- predefined by means of a functional definition that can be used to calculate a concrete value assignment. As an example, consider Figure 3.3, which depicts a tabular definition of a Quality Assessment Function that associates the Qualities "Good," "Satisfying," and "Bad" with Quality Type "response_time_quality" depending on the current Individual of Property Type "response time."

Artifact-Level Quality Assessment Functions associate Qualities with Artifacts based on (sub)sets of their Properties. Such function can be defined as $f_{SA\ PT'_{SA}} : \{I_{pt} | pt \in PT'_{SA} \subseteq PT_{SA}\} \rightarrow I_{SA}$, where I_{SA} is the set of instances of the Quality Type associated with the Entity Type of the particular Software Artifact. Clearly, the SUD itself is such an artifact.

As depicted on Figure 3.4, it is possible to distinguish:

- *System-Level Qualities,* which belong to the whole system and cannot be derived from the Qualities of its parts (e.g., its total cost).

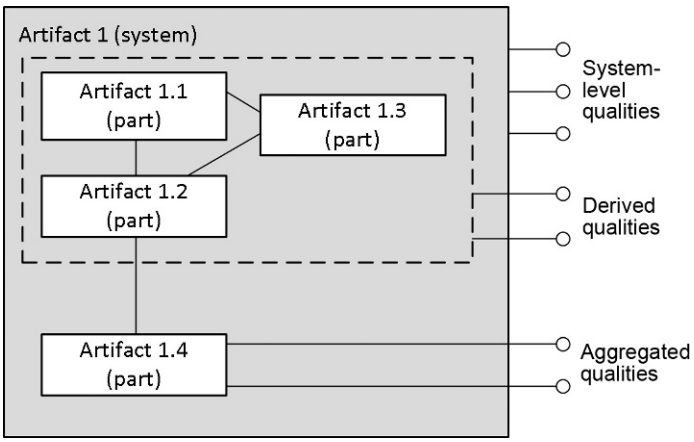

FIGURE 3.4

The relationships between the qualities of the system and the qualities of its parts.

- *Derived Qualities,* which depend on the Qualities of the parts of the system (e.g., the value of latency depends on the Quality Type related to the particular component such as hardware and software performance).
- *Aggregated Qualities* (e.g., the user-interface-related Qualities of the whole system could be considered as the aggregated Qualities of its user-interface-related components).

All these categories of Qualities need to be taken into account in the System-Level Assessment Function.

In general, because Software Artifacts are Individuals that can be part of other Individuals, we can reduce the System-Level and Artifact-Level Functions to the generic version of the Artifact-Level Function, which could perform aggregation over the parts of their corresponding Software Artifact.

Stakeholders are Human Agents participating in the software development process. Quality Subjects are Stakeholders capable of performing Quality Assessments and thus parties in the harmonization process. They apply Quality Assessment Functions driven by their Individual Intentions.

Not all Stakeholders are necessarily Quality Subjects. For example, if we consider the Stakeholder instances "hardware supplier," "developer," and "end user" and assume that hardware supplier has delivered the best of the market so no optimization of hardware is possible, in case of a performance issue in the SUD, the Quality Subjects who have to harmonize their views are "developer" and "end user," whereas "hardware supplier" can be filtered out in this situation. If dealing with quality issues related to the response time of data accesses, then the involved Quality Subjects will be "end user," "database backend developer," and "database supplier," whereas "UI developer" might be filtered out. On the other hand, while doing a usability check, "UI developer" and "end user" will more likely serve as Quality Subjects.

Quality-Related Intentions characterize the Intentions of Quality Subjects when performing Quality Assessment. Given that different Quality Subjects may have different attitudes toward performing the Assessment, we need to distinguish between several categories of *Quality-Related Intentions:*

- *Interest-Related Quality Intention*: The Quality Subject is interested in the Assessment of a particular Software Artifact or of some or all of its Properties.
- *Force-Related Quality Intention*: The Quality Subject is forced (e.g., by the organizational duties) to perform the Assessment but probably has no interest in doing this.
- *Knowledge-Related Quality Intention:* The Quality Subject has the knowledge needed for performing the Assessment; he/she thus could be consulted as an expert.
- *Viewpoint-Related Quality Intention:* The Quality Subject represents a specific user group (viewpoint). As an example, an IT person assesses the Software Artifacts to be used by IT persons.

Quality-Related Intentions are not mutually exclusive: There could be several connections between a Quality Subject and a Software Artifact, each characterized by a different Quality-Related Intention.

We now proceed to conceptualize the Quality Views of Quality Subjects by first introducing *View-Defining Sets* (VDS) on the Artifact and Property Type Level, respectively. This will lead us to those Properties to which the Quality Subjects might want to apply assessment functions. For that purpose assume the following:

- A_{UoD} is the set of all Software Artifacts related to the given UoD.
- $A_u \subseteq A_{\text{UoD}}$ is the set of Software Artifacts that are connected to the Quality Subject u by means of Quality Intentions (*Subject-u-related Artifacts*).

- PT_a is the set of all Property Types defined for the Software Artifact $a \in A_u$.
- $PT_{au} \subseteq PT_a$ is the set of Subject-u-related Property Types of $a \in A_u$.
- P_u is the set of all Quality-Related Intentions of u.
- $VDS_i \subseteq A_u$ is the Artifact-Level "View-Defining Set" = set of Software Artifacts related to Intention $i \in P_u$.
- $VDS_{ia} \subseteq PT_{au}$ is the Property-Level "View-Defining Set" = (sub)set of Property Types associated with Software Artifact and related to the Intention i.

Then the *Quality View* of a Quality Subject u for a particular Intention i may be defined as the total of Quality Assessment Functions that this Quality Subject applies to the elements of the VDS for Intention i:

$$QV_i = \{f_{at} | t \in VDS_{ia}, a \in VDS_i\} \cup \{f_{aVDS_{ia}} | a \in VDS_i\}$$

A Quality Subject may have different Quality Views at the same time connected to his/her different Quality-Related Intentions; for example, a developer could deal with different Business Stakeholders. In this situation, he/she can utilize a specific Quality View for the purpose of communication with every Stakeholder.

3.2.2 Execution-level concepts

After conceptualizing what and how it is assessed, we now have to define the concepts that will enable us to capture the process of performing the assessments. The schema illustrating these concepts is shown on Figure 3.5.

The key execution-level quality-related concept is *Quality Assessment*. Again we differentiate between Software Artifact and Property-Type Level. On the latter, Assessment is viewed as an Activity depending on the following Individuals:

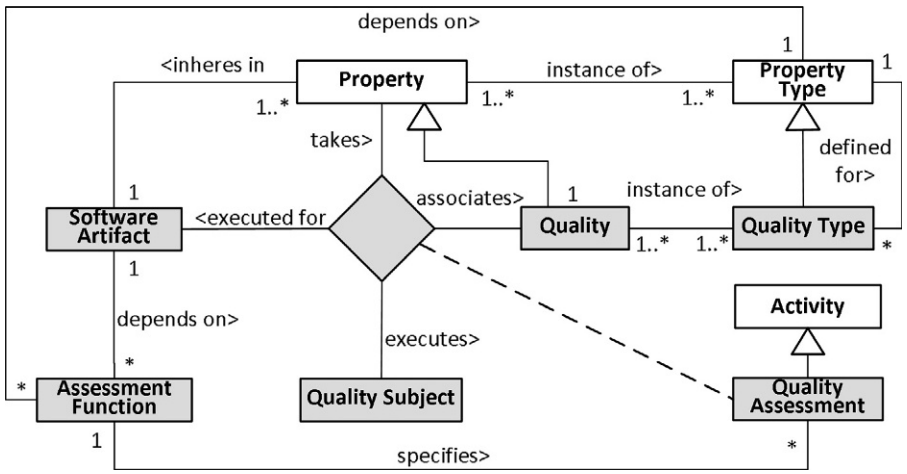

FIGURE 3.5

Execution-level concepts for Property-Type Level Assessment.

1. The Property of the particular Software Artifact to be considered for assessment.
2. The Quality Subject performing the assessment.
3. The Quality that is assigned to the particular Property as a result of assessment.

Artifact-Level Assessment is viewed as the Activity depending on the following Individuals:

1. The Software Artifact to be assessed.
2. The (sub)set of its Properties to be considered for assessment.
3. The Quality Subject performing the assessment.
4. The Quality that is assigned to the particular Artifact as a result of assessment.

To establish the connection between execution-level and specification-level concepts, Quality Assessment Functions are used to perform particular assessments. The Property-Level Quality Assessment operation takes a particular Property $p \in I_{\text{pt}}$, applies the corresponding Quality Assessment Function $f_{\text{SA pt}}$, thus associating the Quality $q \in I_{\text{qt}}$ to Property p according the specification of $f_{\text{SA pt}}$.

The Artifact-Level Quality Assessment operation takes a particular version of SA represented by a (sub)set of its Properties, $P_{\text{SA PT}'_{\text{SA}}} = \{p | p \in I_{\text{pt}}, \text{pt} \in \text{PT}'_{\text{SA}} \subseteq \text{PT}_{\text{SA}}\}$, and applies the corresponding Quality Assessment Function $f_{\text{SA PT}'_{\text{SA}}}$.

3.2.3 State of the Art: Addressing basic quality-related concepts

We published a detailed literature review of the *quality conceptualization techniques* in Shekhovtsov (2011). Among these techniques, following Aßmann and Zschaler (2006) and Pastor and Molina (2007), we distinguish:

There are survey papers devoted to *model-based approaches,* which conceptualize software quality in a solution space (prescriptively defining the system-to-be) under the "closed-world" assumption (everything not explicitly described is assumed nonexistent) (Carvallo, 2005; Deissenboeck et al., 2009).

Ontology-based approaches conceptualize software quality in a problem space (describing the real-world problem domain addressed by the SUD) under the "open-world" assumption (everything not explicitly described is assumed unknown).

In practice, the dividing line between these two categories of approaches is difficult to define, because the descriptions of the approaches do not clearly distinguish between the problem space and the solution space. As a result, the term "quality model" often refers to ontology-based solutions.

Following Shekhovtsov (2011) and Wagner and Deissenboeck (2007), we further classify these techniques according to their abstraction level; we consider abstraction levels for model-based and ontology-based approaches separately.

For quality models, we define the abstraction level according to the notion of a modeling meta-pyramid (Aßmann and Zschaler, 2006): The model on a particular level defines the conceptualization of the language that is used to define the models on the level below. In other words, it acts as a meta-model. Here, we restrict ourselves to only two levels:

1. *Model 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; examples are taxonomies of quality characteristics (*fixed model* techniques; Fenton and Pfleeger (1997)) that are described by example without meta-information (e.g., early techniques, Boehm et al., 1978; McCall et al., 1977, standards, ISO, 2001, 2011). Such approaches are criticized due to the arbitrary choice of quality characteristics (Deissenboeck et al., 2009; Kitchenham and Pfleeger, 1996).

2. *Meta-model level*: For the meta-models (models of languages used for describing the models); in our case a quality meta-model is used to define quality models; we further distinguish *implicit* and *explicit quality meta-models*. The former are introduced "bottom-up," via *custom* or *mixed model* QMT (Fenton and Pfleeger, 1997) and allow modification of the model structure; they often omit quality characteristics from the model descriptions (Dromey, 1996; IEEE, 1998). The latter are introduced "top-down" with the explicit purpose of describing the set of possible quality models (Burgués et al., 2005; Cachero et al., 2007; Jureta et al., 2009b; Susi et al., 2005). Industry examples are *UML quality profiles* adding support for new modeling concepts to UML via an extension of the UML meta-model (Aagedal et al., 2004; Apvrille et al., 2004; OMG, 2003; Rodríguez et al., 2006).

For *ontological approaches*, the abstraction level refers to the concepts being described, not the description approach (all ontologies of different levels can be defined using the same ontology language).

Before elaborating the description of these two levels, it is necessary to mention that basic concepts for representing qualities could be found in most foundational ontologies (we have already discussed this issue in Section 3.2). Different approaches to the definition of concepts for representing properties, qualities, and their perception in the human mind are surveyed in Masolo and Borgo(2005).

The two ontological levels based on foundational ontologies are as follows:

1. The *quality upper ontologies* level covers the techniques describing the concepts (universals) for concrete quality ontologies (e.g., by defining concepts for quality characteristic, quality metric, describing their relationships) (Bertoa et al., 2006; Falbo et al., 2002; Jureta et al., 2009a; Magoutas et al., 2007). Some of these techniques are based on the so-called foundational ontologies; in particular, UFO is applied for the ontologies of software requirement management (de Almeida Falbo and Nardi, 2008), software measurement (Barcellos et al., 2010), and agent-oriented software development (Guizzardi and Wagner, 2005);
2. The *quality ontologies* level covers the techniques directly describing the phenomenon of quality (via particulars; with concepts for performance, reliability, etc.) (Al Balushi et al., 2007; Boehm and In, 1996).

3.3 THE HARMONIZATION PROCESS

3.3.1 Quality Subjects' positions in the harmonization process

Prior to elaborating the conceptualization of the harmonization process, we introduce the following concepts describing the positions of Quality Subjects within the harmonization process (Figure 3.6):

1. The *Organizational Side* is an Organization participating in the software process.
2. The *Process Side* is a Role possessed by a Quality Subject with reference to his/her affiliation with the Organizational Side.

We define two concretizations of the Organizational Side:

1. The *Business Side* is the Organizational Side ordering the software product (e.g., the customer company).

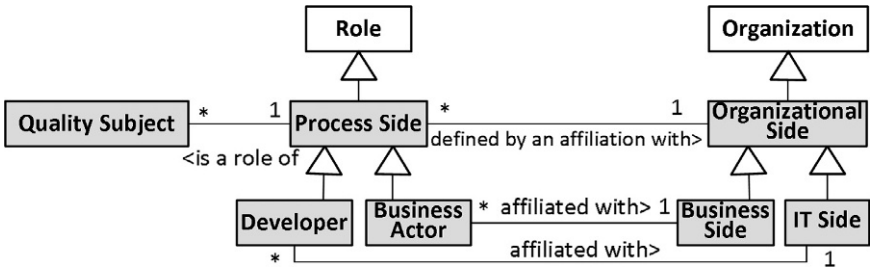

FIGURE 3.6

Process Side role and its related concepts.

2. The *IT Side* is the Organizational Side responsible for developing this product (e.g., the developing company).

The concretizations of the Organizational Side define the concretizations of the Process Side role:

1. The *Business Stakeholder* role is a Process Side that is taken by a Quality Subject (e.g., a person) affiliated with the responsible Business Side.
2. The *Developer* role is a Process Side taken by a Quality Subject (e.g., a person) affiliated with the IT Side.

In some projects, there could be additional roles extending the Process Side.

3.3.2 Process definition and harmonization levels

We define a *Harmonization Process* as *the entirety of interactions between involved Process Sides targeting at a set of Terminology Means, Quality Views, and Qualities that satisfies all these Process Sides.*

Such processes affect the following *harmonization levels* (which also could be seen as process stages):

1. *Terminology Harmonization*: The sides agree on the terminology used to describe quality-related concepts.
2. *View Harmonization*: The sides align their Quality Views.
3. *Quality Harmonization*: The sides agree on the Qualities for the particular version of the system or its component.

These activities are supposed to be enacted with different frequencies. In particular, it could be sufficient to harmonize terminology at the beginning of a particular software development project, whereas Quality harmonization, for example, should be enacted on every change in the system implementation that affects the current SUD state.

As mentioned in the section "Introduction," we restrict ourselves to the detailed description of the View and Quality harmonization activities; Terminology Harmonization is left to a separate publication.

3.3.3 Running example

For a better understandability of our description, we introduce first a running example for the harmonization process to be conceptualized that involves, for reasons of simplicity, only two persons and relates to one particular software artifact:

1. David (the Developer) and Sean (the Business Stakeholder) negotiated the quality of the software component called Financial Report Generator (FRG).
2. Initially, David considered also the Database Subsystem (DBS) and the Hardware Subsystem (HWS) as relevant artifacts, because the quality of FRG depended on their characteristics. Sean, however, only dealt with FRG because he did not see the other components of the system. As a result of negotiation, David agreed not to discuss DBS and HWS with Sean.
3. David was going to deal with the metrics for FRG internal and external quality characteristics (according to ISO/IEC 9126-2, ISO, 2003a, and ISO/IEC 9126-3, ISO, 2003b), in particular I/O utilization, memory utilization, and response time. Sean restricted himself to quality in use metrics (according to ISO/IEC 9126-4; ISO, 2004), namely FRG task efficiency. After negotiation, David agreed not to discuss I/O utilization and memory utilization; whereas Sean agreed to discuss response time instead of task efficiency because this external quality metric was understandable to him. As a result, the parties agreed to consider only FRG response time.
4. Before discussing FRG response time, the sides agreed that they will assess it using the following scale: {very bad, bad, satisfactory, good, very good} ordered from "very bad" to "very good." Also, Sean felt that very good response time would be immediately satisfactory to him so that he would accept FRG immediately; on the other hand, very bad would lead him to stop dealing with David; all other assessment results were negotiable to him.
5. Initially, David produced three FRG prototypes (starting from 1 ms response time) until he felt that the response time of 0.25 ms was good. Then he showed the prototype to Sean, who did not like it and felt this response time was bad.
6. As a result, David reconsidered his judgment and changed it from good to satisfactory. Then he continued producing prototypes until he obtained a response time of 0.1 ms, which he considered good even taking new insights into account. Then he again showed the prototype to Sean; this time Sean also felt the response time was good.
7. As a result, the parties agreed on FRG response time of 0.1 ms.

3.3.4 View harmonization process

At this level, the parties agree on their views on quality; we refer to the process of such agreement as View Harmonization Process.

In the treatment of this and the subsequent process, we assume that the parties share a common Quality-Related Intention of communicating with each other (omitting this Intention from the notational representations.)

3.3.4.1 Stage 1: Harmonizing artifacts

The sides agree on the set of software artifacts to be included in VDS^{H}, forming the harmonized Artifact-Level VDS. Suppose that the external quality-related Properties of the Software Artifact included in VDS^{BS} (i.e., the Developer's view) depend on internal quality-related Properties of other Software Artifacts included in VDS^{D}. Then, in a process of negotiation, the Developer could

1. Agree to discuss only the Software Artifact included in VDS^{BS}, or not to include other Software Artifacts from VDS^{D} into VDS^{H}.
2. Propose that the other party to discuss external quality-related Property Types related to Software Artifacts from VDS^{D}, including these Artifacts in VDS^{H}.

3.3.4.1.1 Artifact harmonization example

The part of the running example related to artifact harmonization is conceptualized as follows:

1. David in a process of dealing with FRG considers $VDS^{David} = \{FRG, HWS, DBS\}$.
2. At the same time, Sean only deals with FRG: $VDS^{Sean} = \{FRG\}$.
3. As a result of negotiation, David agrees not to include DBS and HWS into VDS^{H}.
4. Subsequently, the parties agree on $VDS^{H} = \{FRG\}$.

3.3.4.2 Stage 2: Harmonizing property types

For every Software Artifact a in VDS^{H} as agreed on the previous stage, the sides agree on the set of its Property Types to be included into VDS^{H}_{a}, the harmonized Property-Level VDS.

It is important to emphasize that the harmonized Property Types conceptualize quality characteristics and the relevant metrics belonging to different categories, in particular, Business Stakeholders usually include Property Types conceptualizing quality in use metrics (ISO, 2004) into their VDS, whereas Developers usually include Property Types conceptualizing external (ISO, 2003a) and internal (ISO, 2003b) metrics.

Suppose the Business Stakeholder considers the set of external quality-related Property Types whereas the Developer additionally considers the set of internal quality-related Property Types for the particular artifact. In a process of negotiation, the Developer could

1. Agree to discuss only Property Types included in VDS^{BS}_{a}, or not to include other Property Types from VDS^{D}_{a} in VDS^{H}_{a}.
2. Propose that the Business Stakeholder discuss additional Property Types, or *external quality*-related Types derived from *internal quality*-related VDS^{D}_{a} Property Types; these new Property Types have to be included into VDS^{H}_{a}, possibly instead of some Property Types from VDS^{BS}_{a}.

3.3.4.2.1 Example of harmonizing property types

The part of the running example related to property type harmonization is conceptualized as follows:

1. Having agreed on FRG as the artifact under discussion, the sides have different views concerning the relevant Property Types:
   a. David is going to deal with the Property Types related to internal and external quality: $VDS^{David}_{FRG} = \{\text{I/O utilization}, \text{memory utilization}, \text{response time}\}$
   b. Sean restricts himself to Property Types related to the quality in use: $VDS^{Sean}_{FRG} = \{\text{task efficiency}\}$.
2. As a result of negotiation:
   a. David agrees not to include {I/O utilization, memory utilization} into VDS^{H}_{FRG}
   b. Sean agrees to include response time instead of task efficiency into VDS^{H}_{FRG}.
3. As a result the parties agree on $VDS^{H}_{FRG} = \{\text{response time}\}$.

3.3.4.3 Stage 3: Aligning quality views

The sides now align their Quality Views to these View Definition Sets by selecting the appropriate Quality Assessment Functions. For simplicity, we assume here that the sides align Quality Views containing only Artifact-Level Functions; in this case, these views could be represented as follows:

$$\mathrm{QV}^{\mathrm{DH}} = \left\{ f^{\mathrm{D}}_{a\mathrm{VDS}^{\mathrm{H}}_a} \middle| a \in \mathrm{VDS}^{\mathrm{H}} \right\},\ \ \mathrm{QV}^{\mathrm{BSH}} = \left\{ f^{\mathrm{BS}}_{a\mathrm{VDS}^{\mathrm{H}}_a} \middle| a \in \mathrm{VDS}^{\mathrm{H}} \right\}$$

To align Quality Views, the following activities are performed:

1. The Developer adjusts Quality Assessment Functions belonging to QV^{D} to deal with the Properties of types belonging to $\mathrm{VDS}^{\mathrm{H}}$; in particular the following activities for every $a \in \mathrm{VDS}^{\mathrm{H}}$ are performed:
   a. If $a \notin \mathrm{VDS}^{\mathrm{D\,Old}}$ (new artifact has been taken into account as a result of harmonization) a new $f^{\mathrm{D}}_{a\mathrm{VDS}^{\mathrm{H}}_a}$ is formed and added to the Quality View $\mathrm{QV}^{\mathrm{DH}}$ (here $\mathrm{VDS}^{\mathrm{D\,Old}}$ is the Developer's original non-harmonized VDS).
   b. If $a \in \mathrm{VDS}^{\mathrm{D\,Old}}$, but the corresponding $\mathrm{VDS}^{\mathrm{D\,Old}}_a \neq \mathrm{VDS}^{\mathrm{H}}_a$ (it has been changed as a result of harmonization, e.g., by omitting internal quality-related Property Types not understood by Business Stakeholder), the corresponding $f^{\mathrm{D}}_{a\mathrm{VDS}^{\mathrm{H}}_a}$ is adjusted to deal with modified $\mathrm{VDS}^{\mathrm{H}}_a$.
2. The same activities are performed by Business Stakeholder.
3. Both sides explicitly agree on Quality Types for corresponding Quality Assessment Functions. For $f^{\mathrm{D}}_{a\mathrm{VDS}^{\mathrm{H}}_a}$ and $f^{\mathrm{BS}}_{a\mathrm{VDS}^{\mathrm{H}}_a}$, $a \in \mathrm{VDS}^{\mathrm{H}}$, this means that these functions assign Qualities of the same Quality Types: $I^{\mathrm{DH}}_{\mathrm{aqt}} = I^{\mathrm{BSH}}_{\mathrm{aqt}}$. This can be also performed based on the available historical information about the Quality Types involved in the past.

As a result, the sides agree on harmonized $\mathrm{VDS}^{\mathrm{H}}$ and $\mathrm{VDS}^{\mathrm{H}}_a$, $a \in \mathrm{VDS}^{\mathrm{H}}$ and the aligned $\mathrm{QV}^{\mathrm{DH}}$ and $\mathrm{QV}^{\mathrm{BSH}}$.

3.3.4.3.1 Quality view alignment example

The part of the running example related to quality view alignment is conceptualized as follows:

1. Prior to harmonization the parties had the Quality View configurations:
   a. $\mathrm{QV}^{\mathrm{David}} = \left\{ f^{\mathrm{David}}_{a\mathrm{VDS}^{\mathrm{David}}_a} | a \in \mathrm{VDS}^{\mathrm{David}} \right\}$; this view included the Quality Assessment Function: $\bar{f}^{\mathrm{David}}_{\mathrm{FRG},\mathrm{VDS}^{\mathrm{David}}_{\mathrm{FRG}}}$ where $\mathrm{VDS}^{\mathrm{David}}_{\mathrm{FRG}} = \{\text{network latency}, \text{cache capacity}, \text{response time}\}$ as defined above; this function reflected his ability to assess FRG based on three of its Properties. David's Quality View also included $\bar{f}^{\mathrm{David}}_{\mathrm{HWS},\mathrm{VDS}^{\mathrm{David}}_{\mathrm{HWS}}}$ and $\bar{f}^{\mathrm{David}}_{\mathrm{DBS},\mathrm{VDS}^{\mathrm{David}}_{\mathrm{DBS}}}$ but we omit their description here.
   b. $\mathrm{QV}^{\mathrm{Sean}} = \left\{ f^{\mathrm{Sean}}_{a\mathrm{VDS}^{\mathrm{Sean}}_a} | a \in \mathrm{VDS}^{\mathrm{Sean}} \right\}$; this view included only $f^{\mathrm{Sean}}_{\mathrm{FRG},\mathrm{VDS}^{\mathrm{Sean}}_{\mathrm{FRG}}}$ where $\mathrm{VDS}^{\mathrm{Sean}}_{\mathrm{FRG}} = \{\text{efficiency}\}$ as defined above; this function reflected his ability to assess FRG based on its perceived efficiency.
2. Now after agreeing on $\mathrm{VDS}^{\mathrm{H}} = \{\mathrm{FRG}\}$ and $\mathrm{VDS}^{\mathrm{H}}_{\mathrm{FRG}} = \{\text{response \_ time}\}$ both David and Sean adjust (implicitly, maybe unknowingly for them) their Quality Assessment Functions to reflect their abilities to assess the Software Artifacts belonging to these VDS. In particular,

   a. David's assessment functions for HWS and DBS are dropped and his function for FRG is changed to reflect his ability to assess it based on only one Property, as a result, his adjusted Quality View is as follows: $QV^{David\,H} = \left\{ f^{David}_{FRG, VDS^{H}_{FRG}} \right\}$. Now it reflects his ability to assess FRG based only on its response time.
   b. Sean's Assessment Function for FRG is changed to reflect his ability to rank FRG based on the different Property: $QV^{Sean\,H} = \left\{ f^{Sean}_{FRG, VDS^{H}_{FRG}} \right\}$; now it also reflects his ability to assess FRG based on the response time (and not on the perceived efficiency).
3. Both David and Sean agree that all Qualities to be associated by their Assessment Functions have to belong to the same Quality Type: $I^{David\,H}_{FRG\,qt} = I^{Sean\,H}_{FRG\,qt} = \{very\,bad, bad, satisfactory, good, very\,good\}$.
4. To sum it up, as a result of performing Quality View Alignment activities the sides formed $QV^{David\,H}$ and $QV^{Sean\,H}$ reflecting their common (harmonized) abilities to assess the quality of FRG by associating a common Quality Type to its response time.

It is important to note that the results of the assessments performed by the parties after completing all the steps of the View Harmonization Process can be different as the functions forming both views are different; aligning the assessment results (associated Qualities) is the subject of the subsequent quality harmonization activities.

3.3.5 Quality harmonization process

After completing the View Harmonization Process, the parties are able to assess the harmonized set of Software Artifacts; this is based (1) on using Properties that are instantiated from the harmonized sets of Property Types and (2) on using Qualities to represent assessment results that instantiate harmonized Quality Types. Thus is these parties can agree on the instances of Property Types (particular Properties) based on the instances of Quality Types (particular associated Qualities); we refer to the process of such agreement as *Quality Harmonization Process*.

3.3.5.1 Substitution artifacts

Quality harmonization activities are treated as being based on either the particular versions of the Software Artifacts or *Substitution Artifacts*. Substitution Artifacts are, for example, mockups, software prototypes, and simulation models; they allow for proposing particular Properties (stimuli) to the Stakeholders as arguments in negotiation *instead of* Properties of the original Software Artifact under harmonization and allow for collecting Stakeholders' reactions to these stimuli (associated Qualities) in place of the Qualities associated to the original Artifact. The Properties of the Substitution Artifact have to properly represent the Properties of the original Artifact; as a result, it should be possible to make decisions regarding the Qualities associated to the original Artifact based on those associated to the Substitution Artifact.

We conceptualize the Substitution Artifact as *the one that inherits the relevant Properties from the original Software Artifact and makes the original Artifact inherit its associated Qualities in turn*.

To simplify the subsequent treatment, we describe Quality Harmonization Process as operating with different *versions* of the Software Artifact $a \in VDS^{H}$, where each version is represented by the set of Properties $P_{a VDS^{H}_{a}} = \left\{ p | p \in I_{pt}, pt \in VDS^{H}_{a} \right\}$ possibly inhering in a Substitution Artifact for a;

we refer to such set as *a particular version of Quality Harmonization Object* referred to as $a^{\text{val}} = P_{a\text{VDS}_a^{\text{H}}}$. We also restrict ourselves to only Artifact-Level Assessments.

3.3.5.2 Rank-oriented and property-oriented harmonization. Expected property state

We conceptualize two possible approaches to the quality harmonization process:

1. *Rank-oriented harmonization:* Both sides perform assessments of the particular version of a^{val} (according to their own Quality Views) and perform negotiations based on the ranks over Qualities associated by these assessments. In particular,
   - **a.** The Developer assesses every produced version of a^{val} before submitting it to the Business Stakeholder associating Quality $q^{\text{D}}_{a^{\text{val}}}$.
   - **b.** Ranking over Qualities is introduced; according to this rank if $r\left(q^{\text{BS}}_{a^{\text{val}}}\right) < r\left(q^{\text{D}}_{a^{\text{val}}}\right)$ (the Business Stakeholder ranks the provided a^{val} less favorable than Developer) the sides try to eliminate the reasons of these rank differences.
   - **c.** In producing a^{val}_{i+1} the Developer is guided by both $r\left(q^{\text{BS}}_{a^{\text{val}}_i}\right)$ and $r\left(q^{\text{D}}_{a^{\text{val}}_i}\right)$.
2. *Property-oriented harmonization*: Both sides perform negotiations based on Properties and not the ranks over associated Qualities. In particular:
   - **a.** The Developer makes Business Stakeholder experience $a^{\text{val D}}_i$.
   - **b.** The Business Stakeholder executes Quality Assessment for $a^{\text{val D}}_i$ obtaining Quality $q^{\text{BS}}_{a^{\text{val D}}_i}$, and provides $q^{\text{BS}}_{a^{\text{val D}}_i}$ back to the Developer.
   - **c.** The Developer derives a^{val} implicitly desired for Business Stakeholder (his *Expected Property State*, $a^{\text{val BS}}_i$) from $a^{\text{val D}}_i$ experienced by Business Stakeholder and his/her associated Quality $q^{\text{BS}}_{a^{\text{val D}}_i}$.
   - **d.** In producing $a^{\text{val D}}_{i+1}$ the Developer takes into account revealed Expected Property State $a^{\text{val BS}}_i$.

The Expected Property State thus can be defined as *the set of* a^{val} *Properties that is conceived by the Individual Business Stakeholder as those he/she wants the final version of* a^{val} *to provide and that can be the subject of negotiation*. The Expected Property State can change throughout the Quality Harmonization Process as a result of the evolution of Business Stakeholder's quality vision.

In this subsection, we will describe Quality Harmonization Process only for the case of rank-oriented harmonization due to the space restrictions.

3.3.5.3 Stage 1: Producing an initial example property state

We define our conceptualization for the case when the interaction with Business Stakeholder is performed from the Developer side by *making the Business Stakeholder experience a version of* a^{val}: $a^{\text{val D}}_i$ *that could be the subject of subsequent negotiations*.

We call $a^{\text{val D}}_i$ the *Example Property State*. It reflects the current state of a^{val} development, although it should not necessarily be originated from the implemented version of a^{val}. The only requirement for $a^{\text{val D}}_i$ is that it should *represent* the set of Properties a^{val} is expected to have *if implemented exactly as defined at the given development stage* (e.g., according to the set of requirements, as specified by its accepted design specification, as perceived by the Developers). In practice, the Example Property State can be produced based on a Substitution Artifact.

At this stage, for every Production iteration j:

1. Developer produces the particular development (internal) version of the Example Property State: $a'^{\text{val D}}_j$.

2. Developer executes Quality Assessment based on his/her Quality Assessment Function for $a'^{\text{val D}}_j$ producing Quality $q^{\text{D}}_{a'^{\text{val D}}_j}$.
3. If $r\left(q^{\text{D}}_{a'^{\text{val D}}_j}\right) \geq \text{CPB}^{\text{D}}$ Developer provides $a'^{\text{val D}}_j$ to Business Stakeholder as its external version $a^{\text{val D}}_0$; he/she also takes into account the last $q^{\text{D}}_{a'^{\text{val D}}_j}$ as the Quality related to the version shown: $q^{\text{D}}_{a^{\text{val D}}_0} = q^{\text{D}}_{a'^{\text{val D}}_j}$. Here CPB^{D} is the *Conditional Production Boundary* reflecting the fact that some level of quality is good enough for the Developer to make him/her stop producing iterations and show the current $a^{\text{val D}}_j$ to the Business Stakeholder.

3.3.5.3.1 Example for producing the initial example property state

As stated in the previous section, as a result of completing View Harmonization process, David and Sean agreed on $\text{VDS}^{\text{H}} = \{\text{FRG}\}$ and $\text{VDS}^{\text{H}}_{\text{FRG}} = \{\text{response\_time}\}$ and adjusted their Quality Views. As a result, the Quality Harmonization Object is instantiated as $\text{FRG}^{\text{val}} = P_{\text{FRG},\text{VDS}^{\text{H}}_{\text{FRG}}} = \left\{p \in I_{\text{response\_time}}\right\}$. The relevant activities of the running example are conceptualized as follows:

1. David agrees with Sean the rank over {very bad, bad, satisfactory, good, very good} (i.e., they agree that $r(\text{very bad}) < r(\text{bad}) < r(\text{satisfactory}) \ldots$).
2. David defines for himself his conditional boundary for providing $\text{FRG}^{\text{val David}}$ version to Sean: $\text{CPB}^{\text{David}} = r(\text{good})$.
3. David produces the internal version of the Example Property State $\text{FRG}'^{\text{val David}}_j$. This state could look as follows: $\text{FRG}'^{\text{val David}}_j = \{1\,\text{ms}_{\text{response\_time}}\}$; here we denote by val_{pt} the property val of type pt. This process is repeated for several iterations until he ranks its associated Quality (resulting from the Assessment) as good or higher: $r\left(q^{\text{David}}_{\text{FRG}'^{\text{val David}}_j}\right) \geq r(\text{good})$; it happens on third iteration so it is satisfied with $\text{FRG}'^{\text{val David}}_3 = \{0.25\,\text{ms}_{\text{response\_time}}\}$.
4. David provides $\text{FRG}'^{\text{val David}}_3$ to Sean as $\text{FRG}^{\text{val David}}_0$ and takes into account $q^{\text{David}}_{\text{FRG}'^{\text{val David}}_3}$ as $q^{\text{David}}_{\text{FRG}^{\text{val David}}_0}$.

3.3.5.4 Stage 2: Executing initial assessment and deciding on a negotiation

The initial version of the Example Property State has to be assessed by Business Stakeholder by performing the following steps:

1. Business Stakeholder experiences the initial version of the Example Property State $a^{\text{val D}}_0$ provided by Developer.
2. Business Stakeholder assesses $a^{\text{val D}}_0$ (applying his/her Assessment Function) and produces the Quality $q^{\text{BS}}_{a^{\text{val D}}_0}$.
3. Prior to making the negotiation decision, two negotiation boundaries for Business Stakeholder (inside of his mind) have to be established: *Unconditional Acceptance Boundary* (UAB^{BS}) and *Unconditional Rejection Boundary* (URB^{BS}); the decision-making process then has the following form:
   a. If $r\left(q^{\text{BS}}_{a^{\text{val D}}_0}\right) > \text{UAB}^{\text{BS}}$, $a^{\text{val D}}_0$ is *unconditionally accepted* as completely satisfying the Business Stakeholder; no additional interactions are necessary in this case. Such experience can be of the

great service to the project because it establishes the feeling of trust between Business Stakeholders and Developers.

b. If $r\left(q^{\mathrm{BS}}_{a_0^{\mathrm{val\,D}}}\right) < \mathrm{URB}^{\mathrm{BS}}$, the harmonization process has to be aborted immediately without any attempt to perform negotiation and $a_0^{\mathrm{val\,D}}$ is *unconditionally rejected* as dissatisfying the Business Stakeholder; such experience can be damaging to the whole project.

c. If $r\left(q^{\mathrm{BS}}_{a_0^{\mathrm{val\,D}}}\right)$ falls between $\mathrm{URB}^{\mathrm{BS}}$ and $\mathrm{UAB}^{\mathrm{BS}}$, the negotiation process has to be initiated.

3.3.5.4.1 Example of a negotiation decision

The relevant activities of the running example are conceptualized as follows:

1. Sean defines $\mathrm{UAB}^{\mathrm{Sean}} = r(\mathrm{good})$ and $\mathrm{URB}^{\mathrm{Sean}} = r(\mathrm{bad})$.
2. Sean experiences $\mathrm{FRG}_0^{\mathrm{val\,David}} = \{0.25\,\mathrm{ms}_{\mathrm{response\_time}}\}$ provided by David.
3. Sean assesses $\mathrm{FRG}_0^{\mathrm{val\,David}}$ and produces $q^{\mathrm{Sean}}_{\mathrm{FRG}_0^{\mathrm{val\,David}}} = \mathrm{bad}$.
4. Sean checks the negotiation boundaries; as $\mathrm{URB}^{\mathrm{Sean}} \leq r(\mathrm{bad}) \leq \mathrm{UAB}^{\mathrm{Sean}}$ he decides to initiate the negotiation process.

3.3.5.5 Stage 3: Performing negotiations

The negotiation process is performed iteratively; we denote its iteration as i. For every i, the following activities are performed:

1. Developer takes into account two Qualities associated with $a_{i-1}^{\mathrm{val\,D}}$: his/her own Quality $q^{\mathrm{D}}_{a_{i-1}^{\mathrm{val\,D}}}$ and the Quality associated by Business Stakeholder $q^{\mathrm{BS}}_{a_{i-1}^{\mathrm{val\,D}}}$.
2. If $r\left(q^{\mathrm{BS}}_{a_{i-1}^{\mathrm{val\,D}}}\right) < r\left(q^{\mathrm{D}}_{a_{i-1}^{\mathrm{val\,D}}}\right)$ the Developer adjusts Assessment Functions in his QV to come closer to those of the Business Stakeholder: now getting $r\left(q^{\mathrm{DNew}}_{a_{i-1}^{\mathrm{val\,D}}}\right) < r\left(q^{\mathrm{D}}_{a_{i-1}^{\mathrm{val\,D}}}\right)$. This adjustment can be useful for the subsequent executions of the harmonization process (Assessment Functions can stay adjusted); also the set of adjusted Assessment Functions can be the input to the knowledge base.
3. Developer (after reaching his $\mathrm{CPB}^{\mathrm{D}}$ with $r\left(q^{\mathrm{DNew}}_{a_j^{\mathrm{val\,D}}}\right)$) makes Business Stakeholder experience the new version of Example Property State $a_i^{\mathrm{val\,D}}$ and Business Stakeholder assesses his/her experience again producing $q^{\mathrm{BS}}_{a_i^{\mathrm{val\,D}}}$; in doing this, every side tries to insist on some preferable Qualities, in particular:
   a. Developer tries to keep the ranks for Quality closer to the initial $\mathrm{CPB}^{\mathrm{D}}$ or even to $\mathrm{URB}^{\mathrm{BS}}$ to spend fewer resources.
   b. Business Stakeholder tries to keep the ranks for Quality closer to the $\mathrm{UAB}^{\mathrm{BS}}$ as it is better correspond to what he/she had in mind initially.
4. The negotiation process eventually ends with agreeing upon a version of the Example Property State shown to the Business Stakeholder: the *Accepted Property State* $a^{\mathrm{val\,D}*}$.

3.3.5.5.1 Example of negotiations

For the first negotiation iteration ($i = 1$) the relevant activities of the running example are conceptualized as follows:

1. David takes into account $q^{\text{David}}_{\text{FRG}_0^{\text{val David}}} = \text{good}$ and $q^{\text{Sean}}_{\text{FRG}_0^{\text{val David}}} = \text{bad}$.
2. As $r\left(q^{\text{Sean}}_{\text{FRG}_0^{\text{val David}}}\right) < r\left(q^{\text{David}}_{\text{FRG}_0^{\text{val David}}}\right)$ David adjusts his QV to make possible obtaining $q^{\text{David New}}_{\text{FRG}_0^{\text{val David}}} = \text{satisfactory}$ (so it is ranked closer to the Quality provided by Sean).
3. David produces $\text{FRG}_1^{\text{val David}} = \{0.1\,\text{ms}_{\text{response\_time}}\}$, which he again assesses as $q^{\text{David New}}_{\text{FRG}_1^{\text{val David}}} = \text{good}$ (but with adjusted QV) and makes Sean experience and assess it.
4. Sean now obtains $q^{\text{Sean}}_{\text{FRG}_1^{\text{val David}}} = \text{good}$.
5. The sides agree on the Accepted Quality State $\text{FRG}^{\text{val David}\,*} = \{0.1\,\text{ms}_{\text{response\_time}}\}$.

3.3.6 State of the art: Addressing harmonization process activities

3.3.6.1 Addressing organization sides

We start from approaches that provide generic conceptualizations of the organizational Entities. There are several approaches to such conceptualizations besides UFO-C, in particular, Boella and van der Torre (2006) introduces a foundational ontology of organizations and roles, Bottazzi and Ferrario (2005) define a basic ontology of organizations that is grounded in DOLCE foundational ontology, and Dietz and Habing (2004) describe a meta-ontology for organizations. Other approaches are grouped under the umbrella title of Enterprise ontology; they address different components of the real-world enterprise. Generic enterprise ontologies are presented in Abramowicz et al. (2008), Albani et al. (2006), and Almeida and Cardoso (2011). Another approach is grounded in UFO (Barcellos and de Almeida Falbo, 2009), and the practical ontology-based implementation of the set of such concepts is defined for the Jade agent development framework (Baldoni et al., 2010).

3.3.6.2 Addressing quality subjects

Most of the ontologies of organizations and social aspects (in particular UFO-C, the foundational ontology from Boella and van der Torre, 2006 and the practical implementation from Baldoni et al., 2010) include notions for the organizational roles that could be applied to Quality Subjects; more specific conceptualization approaches include an ontology of social roles (Boella and Van Der Torre, 2007). Baldoni et al. (2006) treat these roles as affordances.

While dealing with the approaches that address the abilities and intentions of Quality Subjects, first of all, it is necessary to categorize those addressing the actors who participate in the software process. Some conceptualization approaches include the notion of stakeholder, but do not connect this notion to the quality assessment process by disallowing per-stakeholder assessments (Gilb, 1988; Kim and Lee, 2005); other techniques allow such assessments (Chung and do Prado Leite, 2009; Chung et al., 1999; Jureta et al., 2009a). A generic attempt to define the foundation of knowledge-related abilities of stakeholders can be based on a knowledge-management ontology (Holsapple and Joshi, 2006). For the field of requirements engineering, the most detailed treatment of stakeholder capabilities is provided in CORE (Jureta et al., 2009a); this treatment is discussed below.

The closest notion to UFO intentions and the derived concepts introduced in this chapter is the concept of *stakeholder speech mode* introduced by the CORE ontology (Jureta et al., 2009a), which represents an uttered stakeholder intention with reference to some aspects of the UoD.

We categorize the techniques addressing stakeholder intentions according to these modes:

1. Directive mode defines explicit goals also addressed, for example, in Chung and do Prado Leite (2009), Kassab et al. (2009), Zhu and Gorton (2007), or in Boehm and In (1996), Krogstie (1998) as quality constraints. See also the notion of a soft goal in Chung et al. (1999) and derived works for a goal that could be only satisfied to some degree.
2. Declarative/assertive mode defines assumptions or the values of attributes (Kayed et al., 2009).
3. Expressive mode is used to declare evaluations also addressed, for example, in Choi et al. (2008) and Tian (2004).

3.3.6.3 Generic process-based techniques

Prior to aligning the state-of-the-art approaches to harmonization levels, we need to briefly consider generic process-level techniques aimed at conceptualizing software quality negotiation.

Software process conceptualization techniques (Acuna and Sanchez-Segura, 2006; Adolph et al., 2012; Münch et al., 2012) propose conceptual foundations for formalizing software process activities, in particular those involving business stakeholders. Some of these approaches are grounded in foundational ontologies. In particular, UFO-based software process conceptualizations are defined in Guizzardi et al. (2008). The most relevant among these techniques is an approach by Adolph et al. (2012) that proposes a conceptualization of the development activities aimed at reaching common understanding between the parties in this process. We discuss this approach in more detail at the end of this section while addressing the treatment of harmonization levels.

Quality-driven process support approaches aim at using quality to drive the whole software process or its particular tasks. Among these approaches are Quality-Driven Re-Engineering (Tahvildari et al., 2003) approaches targeting quality-driven development restricted to the architectural design phase (Kim et al., 2010) and techniques that aim at making the entire software process driven by quality without organizing direct interaction with stakeholders (de Miguel et al., 2008; Grambow et al., 2011; Hummel et al., 2010; Matinlassi, 2005).

3.3.6.4 Addressing harmonization activities

There are several categories of methods addressing the issue of involving stakeholders in the development process as a means for performing the harmonization activities; in describing them, we follow the classification of the methods to represent the quality of the prospective system proposed by Bosch (Bosch, 2000), which includes request- and scenario-based techniques, prototyping, and simulation.

Request-centered techniques implement the way of questioning the business stakeholders and processing their opinions. Empirical techniques (Drew et al., 2006) such as surveys, interviews, brainstorming, questionnaires, or checklists are used (Lauesen, 2002; McManus, 2004) together with software engineering-specific techniques such as CRC cards (Bellin and Suchman-Simone, 1997).

Scenario-centered techniques (Carroll, 1995) organize scenarios of stakeholder interaction with software under development to harmonize qualities; they often rely on request-centered techniques to process the opinions of business stakeholders. In the manual scenario-centered approach, the stakeholders are requested to go through the scenarios (without interacting with the prospective software or its executable model) and express their opinions. The techniques of this kind are widely used to evaluate software architectures (Kazman et al., 1999; Williams and Smith, 2002) and to elicit quality requirements (Barbacci et al., 2003; Gregoriades and Sutcliffe, 2005; Lassing et al., 2002; Sindre and Opdahl, 2005). In industry, such scenarios are often based on user stories (Cohn, 2004): Together they form an approach to gather user requirements in the agile software process (Leffingwell, 2011).

Prototyping techniques use system prototypes as an aid for business stakeholders (Arnowitz et al., 2007), helping them to express their quality preferences. The notion of traditional prototype refers to a scaled-down version of the final system running in its intended context (Floyd, 1984; Lim et al., 2008). They are used in requirement engineering, being embedded into scenarios (Sutcliffe and Ryan, 1998; Tolstedt, 2002) and during the software design activities (Hartmann, 2009). In industry, prototypes are often supplemented by mockups that are simplified versions of prototypes not integrated into usage scenarios and often not requiring writing any code (Schneider, 2007). Implementing mockup-like solutions exclusively as sketches on paper is also known as *paper prototyping* (Snyder, 2003).

Simulation-based techniques use simulation in a sense of "the process of designing and creating a computerized model of a real or proposed system for the purpose of conducting numerical experiments" (Kelton et al., 2004) to model quality characteristics in a way that can be used to support stakeholder involvement. Performance prototyping techniques (Hennig et al., 2003) allow performance simulations to be used as alternatives for prototypes. Other performance simulation solutions are described in Bause et al. (2008), Cortellessa et al. (2008), and Driss et al. (2008). Reliability simulations are proposed in Gokhale et al. (1998), Grishikashvili Pereira and Pereira (2007), and Looker et al. (2007). Some solutions embed quality simulations in context of usage scenarios (Egyed, 2004; Haumer et al., 1999) or complete business processes (Fritzsche et al., 2008, 2009; Jansen-Vullers and Netjes, 2006; Marzolla and Balsamo, 2004; Rozinat et al., 2009; Tewoldeberhan and Janssen, 2008).

An important generic approach addressing different levels of harmonization is provided by Adolph et al. (2012). In this work, the harmonization process is treated in detail: It is oriented at reconciling perspectives, where the perspectives refer to the views of different process participants. However, the authors propose the approach for language and view harmonization but not specifically related to quality issues. Consequently, our work may be considered as a specialization of this approach targeting the harmonization of quality-related perspectives.

3.4 PRACTICAL RELEVANCE

In this section, we describe how our research relates to the practice of software development. First, we describe the empirical studies that formed the starting point for our conceptualizations. These studies have been carried out in cooperation with industrial partners in the framework of the QuASE project. After that, we outline the practical application of our conceptualizations in knowledge-oriented solutions.

3.4.1 Empirical studies

We conducted empirical studies from March 2012 to July 2013 in cooperation with five IT organizations having the following diverse characteristics:

1. A: middle-sized product-oriented IT company targeting the healthcare sector that follows the product line approach in organizing its development projects.
2. B: smaller-sized solution-oriented IT company also offering consulting services.
3. C: small consulting and solution development company targeting SAP applications.

4. D: middle-sized consulting and solution development company with an expertise in quality assurance and management.
5. E: IT service department of a large service-providing company.

3.4.1.1 Conducting interviews

The main data collection method was to conduct structured and semi-structured interviews. Twenty-nine interviews were conducted in two time intervals: 14 interviews in March-June 2012 and 15 interviews in April-July 2013. In the first period, the interviews involved staff of A and B; in the second period the three other companies were involved. The total duration of the interviews was about 40 h; all the interview conversations have been digitally recorded and then transcribed. In total, about 150 questions were discussed (3-4 per hour).

For every company, we interviewed people belonging to three main categories: top managers (CEO/CIO), project managers, and software engineers; the latter group has been subdivided into line developers, software architects, analysts, and requirement engineers. For some companies, such as company B, it was not possible to put the people (except top managers) into a particular category because this company expects from its staff the ability to perform different software engineering and project management activities.

Prior to an interview, its plan has been agreed with the participant, but interviewers were allowed to ask additional questions that were generated by the answers on previous questions. Three sets of questions (related to top managers, project managers, and developers) were discussed with the relevant staff members of all five companies; the questions belonging to these sets were adjusted as a result of the previous interviews, but core questions remained the same. This allowed for the comparison of the results obtained in different contexts.

Three interviews with staff members of A were of a different kind: They were devoted to discussing the issues arising in the company's ongoing projects; consequently, both project managers and involved developers participated. One of the projects under discussion was in trouble, so the possible reasons for such situation were discussed, too.

3.4.1.2 Postmortem analysis

In addition to the interviews, a postmortem analysis of already finished projects was performed in A, B, and D. Prior to performing the analysis, a documentation meeting was held with the responsible staff members. The documents available for analysis included requirement specifications, meeting minutes, excerpts from the data collected in the company IMS or wiki.

3.4.1.3 Data processing

After transcription of the interviews and performing postmortem analysis, the collected data was processed as follows:

1. We started from performing open coding (Corbin and Strauss, 2008; Strauss and Corbin, 1998) of the transcripts with a goal of obtaining an initial set of concepts based on the indicators presented in the data.
2. After the initial set of concepts was established (as a result of the first period's interviews), we compared the indicators from the new interview transcripts with the existing concepts and revised

these concepts if necessary; the exact sets of questions for the interviews belonging to the second time interval was based on the answers of the previous interviews.
3. We performed axial coding (Strauss and Corbin, 1998) with the goal of establishing the relationships between the concepts belonging to the initial set.
4. The obtained preliminary set of concepts and their relationships was validated by the staff of the relevant company-partner; during that validation, necessary modifications were proposed and discussed.

3.4.1.4 Soundness factors

The following factors contributed to the soundness of the empirical studies and, as a result, the soundness of their results:

1. Diversity of the represented companies, ability to ask compatible interview questions in different contexts and compare the results.
2. Ability to interview company staff workers belonging to different groups with the sets of questions specifically targeting the particular group.
3. Ability to supplement the interview data with the data obtained as a result of postmortem analysis.
4. Using sound qualitative research principles in conducting the interviews and processing their results. In particular, by sampling of the people involved in the interviews and by defining the scope of the interviews we followed the principles of theoretical sampling that are part of the grounded theory approach (Strauss and Corbin, 1998).
5. Ability to validate the obtained set of concepts by the partner companies.

3.4.2 Practical application

A software tool based on the presented conceptualizations of the harmonization process is currently under development as a part of the QuASE project. It aims at providing support for

1. Acquiring and formalizing domain knowledge about handling quality-related issues in software processes (we conceptualize such issues as the triggers for launching quality harmonization processes).
2. Collecting the raw information about such issues from the involved parties and converting it into operational knowledge (using available domain knowledge to ensure conversion correctness).
3. Using the collected knowledge for establishing a quality-related communication basis for the involved parties, supporting decision making, reusing quality-related experience, and predicting future quality-related requirements of the involved parties.

3.4.2.1 The QuASE process

We established a four-stage *QuASE process* (Figure 3.7).

1. The *elicitation stage* is devoted to acquiring the raw information about quality-related issues and the relevant harmonization processes from the different parties in a software process and converting this information into a set of semantic knowledge structures that reflect the views of these parties.
2. The *integration stage* is devoted to converting and integrating (including conflict resolution) the results of stage 1 into a "global view."

FIGURE 3.7

Stages of the QuASE process.

3. During the *analysis stage* the analytical tasks such as facilitating knowledge reuse or predicting the handling of the future issues are solved based on the global view.
4. The *dissemination stage* is devoted to converting and externalizing the global view back into the form that reflects the views of the different parties.

Due to space restrictions, we will describe in detail only the elicitation stage of the QuASE process.

3.4.2.2 QuOntology and QuIRepository

The process is based on managing the knowledge about quality-related issues and relevant harmonization processes in a repository that serves as an experience base (*QuIRepository*) and is structured along an ontology (*QuOntology*). QuOntology incorporates, among others, the set of concepts presented in the previous sections. It aims at sharing the conceptual knowledge about software quality, supplying the semantics to supplement the information about quality-related issues and relevant harmonization processes before converting it into the knowledge structures to be stored into QuIRepository; it serves as a foundation of the QuIRepository by defining its structure.

QuIRepository is intended for storing the operational knowledge about quality-related issues as instances of the QuOntology concepts. For the general principles of establishing the QuIRepository, the concept of experience-knowledge base (Basili et al., 1994; Schneider, 2009) is extended to enable collecting and sharing quality-related experience. QuIRepository has to be organized into subsections corresponding to the knowledge supporting the various stages of the QuASE approach by applying knowledge modularization techniques (Stuckenschmidt et al., 2009). In this, we follow Babar (2009) and Feldmann et al. (1999), who propose a knowledge base that facilitates software engineering activities on different stages.

3.4.2.3 Elicitation stage

This stage is devoted to (1) acquiring the raw information on quality-related issues and the relevant harmonization processes from the different parties of the software process, (2) supplementing it by semantics obtained from QuOntology, and (3) storing the resulting knowledge into QuIRepository. We concentrate on collecting *party-dependent knowledge* based on the views and understanding of the particular party (i.e., its specific language) from the following sources:

1. The systems deployed at the developer side to support the software process (IMS, wikis etc.).
2. The process of *instrumented stakeholder interaction,* where business stakeholders interact with QuASE-enhanced substitution artifacts (e.g., prototypes and mockups) in the usual way; but the information about stakeholders' reactions to stimuli (e.g., mouse clicks or selected control paths) is transparently captured and collected into QuIRepository.
3. Direct stakeholder communication specifying the assessments of quality characteristics, such as usability assessments expressed while interacting with a substitution artifact such as mock-up or prototype.

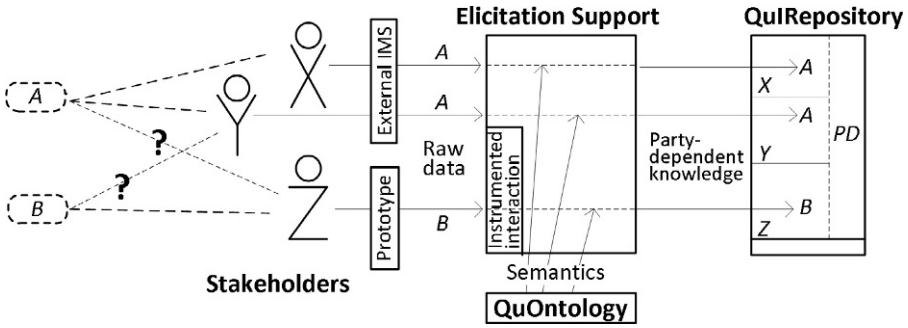

FIGURE 3.8

QuASE process elicitation stage knowledge conversions.

Figure 3.8 illustrates an exemplary situation. Suppose that the stakeholders X (*project manager*) and Y (*developer*) state their knowledge on the harmonization process related to issue A (*a particular performance issue*) in their terminology, whereas stakeholder Z (*business person on the customer side*) uses his terminology for stating his knowledge on the harmonization process related to issue B (*a particular usability issue*). Neither Y knows the terminology of business stakeholder Z, nor does Z understand the terminology of the IT-persons X and Y. These problematic links are denoted by question marks in Figure 3.8.

X and Y provide their knowledge about issue A via an external IMS, Z provides his knowledge about issue B via instrumented interaction with a prototype. This raw data is supplemented with the semantics from QuOntology (comprised, in particular, of the concepts defined in this chapter). The results are stored into the party-dependent knowledge section PD of QuIRepository, where each view has its separate subsection. Knowledge about a particular issue and the relevant harmonization process can be found in different subsections corresponding to the respective views.

To establish a theoretical basis for the acquisition of the raw information from the parties and converting it into knowledge, it is necessary to conceptualize the data to be collected. The set of concepts presented in this chapter are the basis of this conceptualization. In addition, the concept of *quality-related issue* has to be defined generalizing the artifacts available in existing IMS; in particular, it is necessary to treat software requirements as specific categories of issues (Shekhovtsov et al., 2013; Weinreich and Buchgeher, 2010) and associate conceptualized harmonization processes such as defined in Section 3.3 with the particular issues triggering them. To formalize the process of acquiring party-dependent knowledge with a support of QuOntology, we apply the techniques of ontology-based knowledge acquisition (Abecker and Elst, 2009; Aroyo et al., 2006; Lebbink et al., 2002; Wang et al., 2002) and experience-knowledge transformation (Sanin et al., 2007).

3.5 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

3.5.1 Basic conclusions

In this chapter, we presented a set of concepts describing the process of harmonizing the views on quality possessed by the different sides involved in a software process. These concepts

1. Are based on applying empirical techniques to the qualitative data obtained from the business partners in the framework of the QuASE project.
2. Are grounded in a set of basic concepts provided by the UFO.

Combining these two characteristics of the provided conceptualizations allowed us

1. To supply these conceptualizations with real-world semantics supported by empirical evidence and, as a result, more truthful to the view harmonization domain.
2. To make explicit their ontological commitments.

Our conceptualizations could be used as a reference framework for applications in the domain of harmonizing the views on quality; the conceptual models utilized by these tools could be based on these concepts to achieve a higher degree of reuse and semantic interoperability. Reaching the above two goals contributes to the effectiveness of these practical applications as allows conceptual models to be at the same time based on reality, verifiable and conceptually clear.

3.5.2 Future research and implementation directions

Future directions of research include reaching a deeper degree of grounding in the existing foundational and upper-level ontologies, in particular, UFO-C as an ontology of social connections between the actors, ontologies of organizations and roles, and ontologies related to knowledge acquisition and representation.

From the implementation point of view, we are in a process of applying the proposed conceptualizations by implementing a complete common ontology of quality-related stakeholder interactions (QuOntology) that is intended for sharing the conceptual knowledge about the domain and for supporting the harmonization process. In particular, the QuASE tool will support the following stages of a harmonization process based on QuOntology:

1. Acquiring the raw information about quality-related issues (in a sense of issue management systems such as JIRA) from the different parties in a software process and converting this information into a set of semantic knowledge structures reflecting the views of these parties (with a process of conversion here and during the following stages supported by the necessary semantics available from QuOntology).
2. Converting and integrating (including conflict resolution) the results of the previous stage into a "global view" (separating party-independent and party-specific knowledge).
3. Converting and externalizing the global view back into the form that reflects the views of the different parties.

Acknowledgment

The QuASE Project is sponsored by the Austrian Research Promotion Agency (FFG) in the framework of the Bridge 1 program (http://www.ffg.at/bridge1); Project ID: 3215531.

References

Aagedal, J., de Miguel, M.A., Fafournoux, E., Lund, M.S., Stolen, K., 2004. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Technical report TR 2004-06-01, Object Management Group.

Abecker, A., Elst, L., 2009. Ontologies for knowledge management. In: Staab, S., Studer, R. (Eds.), Handbook on Ontologies (International Handbooks on Information Systems). Springer, Berlin/Heidelberg, pp. 713–734.

Abramowicz, W., Filipowska, A., Kaczmarek, M., Pedrinaci, C., Starzecka, M., Walczak, A., 2008. Organization structure description for the needs of semantic business process management. In: 3rd International Workshop on Semantic Business Process Management Colocated with 5th European Semantic Web Conference.

Acuna, S.T., Sanchez-Segura, M.I. (Eds.), 2006. New Trends in Software Process Modeling. World Scientific Publishing, Singapore.

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empir. Softw. Eng. 16, 487–513.

Adolph, S., Kruchten, P., Hall, W., 2012. Reconciling perspectives: a grounded theory of how people manage the process of software development. J. Syst. Softw. 85, 1269–1286.

Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P., 2007. ElicitO: a quality ontology-guided NFR elicitation tool. In: Sawyer, P., Paech, B., Heymans, P. (Eds.), REFSQ 2007. In: Lecture Notes in Computer Science, vol. 4542. Springer, Berlin/Heidelberg, pp. 306–319.

Albani, A., Dietz, J.L., Zaha, J.M., 2006. Identifying business components on the basis of an enterprise ontology. In: Interoperability of Enterprise Software and Applications. Springer, Berlin/Heidelberg, pp. 335–347.

Almeida, J.P.A., Cardoso, E., 2011. On the elements of an enterprise: towards an ontology-based account. In: Proceedings of the 2011 ACM Symposium on Applied Computing. ACM, New York, pp. 323–330.

Apvrille, L., Courtiat, J.-P., Lohr, C., de Saqui-Sannes, P., 2004. TURTLE: a real-time UML profile supported by formal validation toolkit. IEEE Trans. Softw. Eng. 30 (7), 473–487.

Arnowitz, J., Aretn, A., Berger, N., 2007. Effective Prototyping for Software Makers. Morgan Kaufmann, San Francisco, CA.

Aroyo, L., Denaux, R., Dimitrova, V., Pye, M., 2006. Interactive ontology-based user knowledge acquisition: a case study. In: ESCW'06.

Aßmann, U., Zschaler, S., 2006. Ontologies, meta-models, and the model-driven paradigm. In: Calero, C., Ruiz, F., Piattini, M. (Eds.), Ontologies for Software Engineering and Software Technology. Springer, Berlin/Heidelberg, pp. 255–279.

Babar, M.A., 2009. Supporting the software architecture process with knowledge management. In: Babar, M.A., Dingsøyr, T., Lago, P., van Vliet, H. (Eds.), Software Architecture Knowledge Management. Springer, Berlin/Heidelberg, pp. 69–86.

Baldoni, M., Boella, G., Genovese, V., Mugnaini, A., Grenna, R., van der Torre, L., 2010. A middleware for modeling organizations and roles in jade. In: Programming Multi-Agent Systems. Springer, Berlin/Heidelberg, pp. 100–117.

Baldoni, M., Boella, G., Van Der Torre, L., 2006. Modelling the interaction between objects: roles as affordances. In: Knowledge Science, Engineering and Management. Springer, Berlin/Heidelberg, pp. 42–54.

Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C.B., Wood, W.G., 2003. Quality Attribute Workshops (QAWs), third ed. Carnegie Mellon University, Pittsburgh, PA.

Barcellos, M.P., de Almeida Falbo, R., 2009. Using a foundational ontology for reengineering a software enterprise ontology. In: Advances in Conceptual Modeling-Challenging Perspectives. Springer, Berlin/Heidelberg, pp. 179–188.

Barcellos, M.P., Falbo, R.D.A., Dalmoro, R., 2010. A well-founded software measurement ontology. In: Proceedings of the 6th International Conference on Formal Ontology in Information Systems (FOIS 2010), Toronto-Canadá.

Basili, V., Caldiera, G., Rombach, D.H., 1994. Experience factory. In: Marciniak, J.J. (Ed.), Encyclopedia of Software Engineering. Wiley, New York, pp. 469–476.

Bause, F., Buchholz, P., Kriege, J., Vastag, S., 2008. A framework for simulation models of service-oriented architectures. In: SIPEW 2008. In: Lecture Notes in Computer Science, vol. 5119. Springer, Berlin/Heidelberg, pp. 208–227.

Bellin, D., Suchman-Simone, S., 1997. The CRC Card Book. Addison-Wesley, Reading, MA.

Bertoa, M., Vallecillo, A., Garc¡a, F., 2006. An ontology for software measurement. In: Calero, C., Ruiz, F., Piattini, M. (Eds.), Ontologies for Software Engineering and Software Technology. Springer, Berlin/Heidelberg, pp. 175–196.

Boehm, B., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merritt, M.J., 1978. Characteristics of Software Quality. North Holland, New York.

Boehm, B., In, H., 1996. Identifying quality-requirements conflicts. IEEE Softw. 13 (2), 25–35.

Boella, G., van der Torre, L., 2006. A foundational ontology of organizations and roles. In: Baldoni, M., Endriss, U. (Eds.), Declarative Agent Languages and Technologies IV. Lecture Notes in Computer Science, vol. 4327. Springer, Berlin-Heidelberg, pp. 78–88.

Boella, G., Van Der Torre, L., 2007. The ontological properties of social roles in multi-agent systems: definitional dependence, powers and roles playing roles. Artif. Intell. Law 15 (3), 201–221.

Bosch, J., 2000. Design and Use of Software Architectures. Addison-Wesley, Reading, MA.

Bottazzi, E., Ferrario, R., 2005. A path to an ontology of organizations. In: Guizzardi, G., Wagner, G. (Eds.), VORTE'05, pp. 9–16.

Burgués, X., Franch, X., Ribó, J.M., 2005. A MOF-compliant approach to software quality modeling. In: Delcambre, L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, O. (Eds.), Conceptual Modeling—ER 2005. In: Lecture Notes in Computer Science, vol. 3716. Springer, Berlin/Heidelberg, pp. 176–191.

Cachero, C., Calero, C., Poels, G., 2007. Metamodeling the quality of the web development process' intermediate artifacts. In: Baresi, L., Fraternali, P., Houben, G.-J. (Eds.), WISE 2007. In: Lecture Notes in Computer Science, vol. 4607. Springer, Berlin/Heidelberg, pp. 74–89.

Carroll, J.M. (Ed.), 1995. Scenario-Based Design. Wiley, New York.

Carvallo, J.P., 2005. Systematic construction of quality models for COTS-based systems (Ph.D. thesis). Universitat Politècnica de Catalunya, Barcelona.

Choi, Y., Lee, S., Song, H., Park, J., Kim, S.H., 2008. Practical S/W component quality evaluation model. In: 10th IEEE International Conference on Advanced Communication Technology (ICACT'08). IEEE Press, New York, pp. 259–264.

Chung, L., do Prado Leite, J., 2009. 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. Lecture Notes in Computer Science, vol. 5600. Springer, Berlin/Heidelberg, pp. 363–379.

Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., 1999. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston, MA.

Cohn, M., 2004. User Stories Applied: For Agile Software Development. Addison Wesley, Reading, MA.

Coleman, G., O'Connor, R., 2008. Investigating software process in practice: a grounded theory perspective. J. Syst. Softw. 81, 772–784.

Cooper, A., 2004. The Inmates are Running the Asylum. Sams, Indianapolis, IN.

Corbin, J., Strauss, A., 2008. Basics of Qualitative Research, third ed. Sage Publications, Thousand Oaks, CA.

Cortellessa, V., Pierini, P., Spalazzese, R., Vianale, A., 2008. MOSES: modeling software and platform architecture in UML 2 for simulation-based performance analysis. In: QoSA 2008. In: Lecture Notes in Computer Science, vol. 5281. Springer, Berlin/Heidelberg, pp. 86–102.

de Almeida Falbo, R., Nardi, J.C., 2008. Evolving a software requirements ontology. In: XXXIV Conferencia Latinoamericana de Informática, Santa Fe, Argentina, pp. 300–309.

de Miguel, M.A., Massonet, P., Silva, J.P., Briones, J., 2008. Model based development of quality-aware software services. In: ISORC'08. IEEE, New York, pp. 563–569.

Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S., 2009. Software quality models: purposes, usage scenarios and requirements. In: 7th International Workshop on Software Quality (WoSQ 09). IEEE Press, New York, pp. 9–14.

Dietz, J.L., Habing, N., 2004. A meta ontology for organizations. In: On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops. Springer, Berlin/Heidelberg, pp. 533–543.

Drew, P., Raymond, G., Weinberg, D. (Eds.), 2006. Talk and Interaction in Social Research Methods. Sage Publications, Thousand Oaks, CA.

Driss, M., Jamoussi, Y., Jezequel, J.-M., Hajjami, H., Ghezala, B., 2008. A discrete-events simulation approach for evaluation of service-based applications. In: ECOWS'08. IEEE, New York, pp. 73–78.

Dromey, R.G., 1996. Cornering the chimera. IEEE Softw. 13, 33–43.

Egyed, A., 2004. Dynamic deployment of executing and simulating software components. In: Component Deployment. Lecture Notes in Computer Science, vol. 3083. Springer, Berlin/Heidelberg, pp. 113–128.

Falbo, R.A., Guizzardi, G., Duarte, K.C., 2002. An ontological approach to domain engineering. In: 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02). ACM Press, New York, pp. 351–358.

Feldmann, R.L., Geppert, B., Rößler, F., 1999. An integrating approach for developing distributed software systems—combining formal methods, software reuse, and the experience base. In: ICECCS'99. IEEE, New York, pp. 54–63.

Fenton, N., Pfleeger, S.L., 1997. Software Metrics: A Rigorous and Practical Approach. PWS Publishing, Boston, MA.

Floyd, C., 1984. A systematic look at prototyping. In: Approaches to Prototyping. Springer, Berlin/Heidelberg, pp. 1–17.

Fritzsche, M., Gilani, W., Fritzsche, C., Spence, I., Kilpatrick, P., Brown, J., 2008. Towards utilizing model-driven engineering of composite applications for business performance analysis. In: ECMDA-FA'08, pp. 369–380.

Fritzsche, M., Picht, M., Gilani, W., Spence, I., Brown, J., Kilpatrick, P., 2009. Extending BPM environments of your choice with performance related decision support. In: BPM 2009. In: Lecture Notes in Computer Science, vol. 5701. Springer, Berlin/Heidelberg, pp. 97–112.

Gärdenfors, P., 2000. Conceptual Spaces: A Geometry of Thought. MIT Press, Cambridge, MA.

Gilb, T., 1988. Principles of Software Engineering Management. Addison Wesley, Reading, MA.

Gokhale, S.S., Lyu, M.R., Trivedi, K.S., 1998. Reliability simulation of component-based software systems. In: ISSRE'98, pp. 192–201.

Grambow, G., Oberhauser, R., Reichert, M., 2011. Contextual injection of quality measures into software engineering processes. Int. J. Adv. Softw. 4 (1–2), 76–99.

Gregoriades, A., Sutcliffe, A., 2005. Scenario-based assessment of nonfunctional requirements. IEEE Trans. Softw. Eng. 31 (5), 392–409.

Grishikashvili Pereira, E., Pereira, R., 2007. Simulation of fault monitoring and detection of distributed services. Simul. Model. Pract. Theory 15, 492–502.

Guizzardi, G., 2005. Ontological foundations for structural conceptual models, University of Twente.

Guizzardi, G., Falbo, R., Guizzardi, R.S., 2008. Grounding software domain ontologies in the unified foundational ontology (UFO): the case of the ODE software process ontology. In: Proceedings of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments, pp. 244–251.

Guizzardi, G., Wagner, G., 2005. Towards ontological foundations for agent modelling concepts using the unified fundational ontology (UFO). In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (Eds.), Agent-Oriented Information Systems II. In: Lecture Notes in Computer Science, vol. 3508. Springer, Berlin/Heidelberg, pp. 110–124.

Guizzardi, G., Zamborlini, V., 2013. A common foundational theory for bridging two levels in ontology-driven conceptual modeling. In: Software Language Engineering. Springer, Berlin/Heidelberg, pp. 286–310.

Hartmann, B., 2009. Gaining design insight through interaction prototyping tools (Ph.D. dissertation), Stanford University.
Haumer, P., Heymans, P., Jarke, M., Pohl, K., 1999. Bridging the gap between past and future in RE: a scenario-based approach. In: RE'99. IEEE CS Press, New York, pp. 66–73.
Hennig, A., Hentschel, A., Tyack, J., 2003. Performance prototyping—generating and simulating a distributed IT-system from UML models. In: ESM'2003. IEEE, New York.
Herre, H., 2010. General formal ontology (GFO): a foundational ontology for conceptual modelling. In: Poli, R., Healy, M., Kameas, A. (Eds.), Theory and Applications of Ontology: Computer Applications. Springer, Netherlands, pp. 297–345.
Holsapple, C.W., Joshi, K., 2006. Knowledge management ontology. In: Encyclopedia of Knowledge Management. Idea Group Reference, Hershey, PA, pp. 397–402.
Hummel, O., Momm, C., Hickl, S., 2010. Towards quality-aware development and evolution of enterprise information systems. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, Sierre, Switzerland, pp. 137–144.
IEEE, 1998. IEEE 1061-1998: IEEE Standard for Software Quality Metrics Methodology. IEEE Press, New York.
ISO, 2001. ISO/IEC 9126-1:2001: Software Engineering—Product Quality—Part 1: Quality Model. International Organization for Standardization, Geneva.
ISO, 2003a. ISO/IEC 9126-2:2003: Software Engineering—Product Quality—Part 2: External Metrics. International Organization for Standardization, Geneva.
ISO, 2003b. ISO/IEC 9126-3:2003: Software Engineering—Product Quality—Part 3: Internal Metrics. International Organization for Standardization, Geneva.
ISO, 2004. ISO/IEC 9126-4:2004: Software Engineering—Product Quality—Part 4: Quality-in-Use Metrics. International Organization for Standardization, Geneva.
ISO, 2007. ISO/IEC 24744:2007: Software Engineering—Metamodel for Development Methodologies. International Organization for Standardization, Geneva.
ISO, 2011. ISO/IEC 25010:2011: Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models. International Organization for Standardization, Geneva.
Jansen-Vullers, M., Netjes, M., 2006. Business process simulation—a tool survey. In: CPN Tools Workshop.
Jureta, I., Mylopoulos, J., Faulkner, S., 2009a. A core ontology for requirements. Appl. Ontol. 4 (3–4), 169–244.
Jureta, I.J., Herssens, C., Faulkner, S., 2009b. A comprehensive quality model for service-oriented systems. Softw. Qual. J. 17, 65–98.
Kassab, M., Ormandjieva, O., Daneva, M., 2009. An ontology based approach to non-functional requirements conceptualization. In: 4th International Conference on Software Engineering Advances (SEA'09). IEEE Press, New York, pp. 299–308.
Kayed, A., Hirzalla, N., Samhan, A.A., Alfayoumi, M., 2009. Towards an ontology for software product quality attributes. In: 4th International Conference on Internet and Web Applications and Services (ICIW '09). IEEE Press, New York, pp. 200–204.
Kazman, R., Barbacci, M., Klein, M., Carriere, S.J., 1999. Experience with performing architecture tradeoff analysis. In: ICSE'99. ACM, New York, pp. 54–63.
Kelton, W.D., Sadowski, R.P., Sadowski, D.A., 2004. Simulation with Arena, second ed. McGraw-Hill, New York.
Kim, E., Lee, Y., 2005. Quality Model for Web Services 2.0. OASIS.
Kim, S., Kim, D.-K., Park, S., 2010. Tool support for quality-driven development of software architectures. In: ASE'10. ACM, Antwerp, Belgium, pp. 127–130.
Kitchenham, B., Pfleeger, S.L., 1996. Software quality: the elusive target. IEEE Softw. 13 (1), 12–21.
Krogstie, J., 1998. Integrating the understanding of quality in requirements specification and conceptual modeling. ACM SIGSOFT Softw. Eng. Notes 23 (1), 86–91.

Lassing, N., Bengtsson, P., Bosch, J., Vliet, H.V., 2002. Experience with ALMA: architecture-level modifiability analysis. J. Syst. Softw. 61 (1), 47–57.
Lauesen, S., 2002. Software Requirements: Styles and Techniques. Addison-Wesley, Reading, MA.
Lebbink, H.-J., Witteman, C.L.M., Meyer, J.-J.C., 2002. Ontology-based knowledge acquisition for knowledge systems. In: Blockdeel, H., Denecker, M. (Eds.), BNAIC'02, pp. 195–202.
Leffingwell, D., 2011. Agile Software Requirements. Addison-Wesley, Reading, MA.
Lim, Y.-K., Stolterman, E., Tenenberg, J., 2008. The anatomy of prototypes: prototypes as filters, prototypes as manifestations of design ideas. ACM Trans. Comput.-Hum. Interact. 15 (2)Article 7.
Looker, N., Xu, J., Munro, M., 2007. Determining the dependability of service-oriented architectures. Int. J. Simulation Process Model. 3 (1–2), 88–97.
Magoutas, B., Halaris, C., Mentzas, G., 2007. An ontology for the multi-perspective evaluation of quality in E-government services. In: Wimmer, M.A., Scholl, J., Grönlund, Å. (Eds.), EGOV 2007. Lecture Notes in Computer Science, vol. 4656. Springer, Berlin/Heidelberg, pp. 318–329.
Marzolla, M., Balsamo, S., 2004. UML-PSI: the UML performance simulator. In: QEST'04. IEEE, New York, pp. 340–341.
Masolo, C., Borgo, S., 2005. Qualities in formal ontology. In: Hitzler, P., Lutz, C., Stumme, G. (Eds.), Workshop on Foundational Aspects of Ontologies (FOnt 2005), pp. 2–16.
Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., 2003. The WonderWeb Library of Foundational Ontologies. WonderWeb Deliverable D18. Ontology Library (final). ISTC-CNR, Trento.
Matinlassi, M., 2005. Quality-driven software architecture model transformation. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture. IEEE Computer Society, New York, pp. 199–200.
McCall, J.A., Richards, P.K., Walters, G.F., 1977. Factors in Software Quality, NTIS.
McManus, J., 2004. Managing Stakeholders in Software Development Projects. Butterworth-Heinemann, London.
Münch, J., Armbrust, O., Kowalczyk, M., Soto, M., 2012. Software Process Definition and Management. Springer, Berlin/Heidelberg.
OMG, 2003. UML Profile for Schedulability, Performance, and Time, version 1.0, Object Management Group.
OMG, 2008. Software Process Engineering Metamodel (SPEM) 2.0, Object Management Group.
Pastor, O., Molina, J.C., 2007. Model-Driven Architecture in Practice. Springer, Heidelberg.
Rodríguez, A., Fernández-Medina, E., Piattini, M., 2006. Capturing security requirements in business processes through a UML 2.0 activity diagrams profile. In: Roddick, J.F., Benjamins, V.R., Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R., Grandi, F., Han, H., Hepp, M., Lytras, M., Mišic, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (Eds.), ER 2006 Workshops and Tutorials. Lecture Notes in Computer Science, vol. 4231. Springer, Berlin/Heidelberg, pp. 32–42.
Rozinat, A., Wynn, M., van der Aalst, W., ter Hofstede, A., Fidge, C., 2009. Workflow simulation for operational decision support. Data Knowl. Eng. 68, 834–850.
Sanin, C., Szczerbicki, E., Toro, C., 2007. An OWL ontology of set of experience knowledge structure. J. Universal Comput. Sci. 13, 209–223.
Schneider, K., 2007. Generating fast feedback in requirements elicitation. In: Sawyer, P., Paech, B., Heymans, P. (Eds.), REFSQ 2007. In: Lecture Notes in Computer Science, vol. 4542. Springer, Berlin/Heidelberg, pp. 160–174.
Schneider, K., 2009. Experience and Knowledge Management in Software Engineering. Springer, Berlin/Heidelberg.
Shekhovtsov, V.A., 2011. On the evolution of quality conceptualization techniques. In: Kaschek, R., Delcambre, L. (Eds.), The Evolution of Conceptual Modeling. Lecture Notes in Computer Science, vol. 6520. Springer, Berlin/Heidelberg, pp. 117–136.
Shekhovtsov, V.A., Mayr, H.C., Kop, C., 2013. Towards conceptualizing quality-related stakeholder interactions in software development. In: Mayr, H.C., Kop, C., Liddle, S., Ginige, A. (Eds.), Information Systems:

Methods, Models, and Applications. Lecture Notes in Business Infomation Processing, vol. 137. Springer, Berlin/Heidelberg, pp. 73–86.
Sindre, G., Opdahl, A.L., 2005. Eliciting security requirements with misuse cases. Req. Eng. 10 (1), 34–44.
Snyder, C., 2003. Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces. Morgan Kaufmann, San Francisco, CA.
Strauss, A.L., Corbin, J.M., 1998. Basics of Qualitative Research: Grounded Theory Procedures and Techniques, second ed. Sage Publications, Thousand Oaks, CA.
Stuckenschmidt, H., Parent, C., Spaccapietra, S. (Eds.), 2009. Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization. Springer, Berlin/Heidelberg.
Susi, A., Perini, A., Mylopoulos, J., 2005. The tropos metamodel and its use. Informatica 29 (4), 401–408.
Sutcliffe, A., Ryan, M., 1998. Experience with SCRAM: a scenario requirements analysis method. In: ICRE'98. IEEE CS Press, New York, pp. 164–171.
Tahvildari, L., Kontogiannis, K., Mylopoulos, J., 2003. Quality-driven software re-engineering. J. Syst. Softw. 66 (3), 225–239.
Tewoldeberhan, T., Janssen, M., 2008. Simulation-based experimentation for designing reliable and efficient web service orchestrations in supply chains. Electron. Commerce Res. Apps. 7, 82–92.
Tian, J., 2004. Quality-evaluation models and measurements. IEEE Softw. 21 (3), 84–91.
Tolstedt, J.L., 2002. Prototyping as a means of requirements elicitation. In: SAE International Off-Highway Congress (SAE Technical Paper Series), vol. 2002-01-1466.
Wagner, S., Deissenboeck, F., 2007. An integrated approach to quality modelling. In: WoSQ'07. ACM, New York.
Walia, G.S., Carver, J.C., 2009. A systematic literature review to identify and classify software requirement errors. Inf. Softw. Technol. 51, 1087–1109.
Wang, X., Chan, C.W., Hamilton, H.J., 2002. Design of knowledge-based systems with the ontology-domain-system approach. In: SEKE'02. ACM, New York, pp. 233–236.
Weinreich, R., Buchgeher, G., 2010. Integrating requirements and design decisions in architecture representation. In: Babar, M.A., Gorton, I. (Eds.), ECSA'10. In: Lecture Notes in Computer Science, vol. 6285. Springer, Berlin/Heidelberg, pp. 86–101.
Westland, J.C., 2002. The cost of errors in software development: evidence from industry. J. Syst. Softw. 62, 1–9.
Williams, L.G., Smith, C.U., 2002. PASA: a method for the performance assessment of software architecture. In: 3rd Workshop on Software Performance, Rome, Italy. ACM Press, New York, pp. 179–189.
Zhu, L., Gorton, I., 2007. UML profiles for design decisions and non-functional requirements. In: Proceedings of ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK'07). IEEE Press, New York.