Towards Conceptualizing Quality-Related Stakeholder Interactions in Software Development

Vladimir A. Shekhovtsov, Heinrich C. Mayr, and Christian Kop

Institute for Applied Informatics, Alpen-Adria-Universität Klagenfurt, Austria
shekvl@yahoo.com, {mayr,chris}@ifit.uni-klu.ac.at

Abstract. The paper addresses the issue of organizing quality-related interaction between business stakeholders and software developers relying on established common vocabulary. It establishes a conceptual representation for the process of such interaction. This conceptualization is based on a set of notions representing software quality and its particular incarnations; they are used to define the activities of the interaction process. The process is conceptualized on two levels: a coarse-grained level defining the set of generic activities and the conditions of launching these activities and a fine-grained level describing particular interaction steps in detail. The conceptualization is expected to be shared as a part of upcoming ontology of stakeholder quality perception and assessment.

1 Introduction

Software processes cannot be successful without involving the respective business stakeholders. One important category of such involvement concerns collecting stakeholder opinions and expectations on quality of the prospective system. Unfortunately, organizing such kind of involvement still remains an open problem mainly because of the difficulties in establishing the necessary communication channels between the involved parties (on the customer side and on the developer side), especially carrying quality-related information (quality-carrying channels). This difficulty can be largely attributed to the fact that these parties speak different languages while talking about quality and it usually takes a long time to agree on common vocabulary.

We propose to address this problem by establishing the intelligent support for dealing with quality-related information in the software process. It involves collecting rich descriptions of quality-related issues involving business stakeholders into a semantic repository and using this information to predict the reaction of the parties to the future issues of similar kind or to facilitate coming to the common language by these parties – with a purpose of maintaining effective quality-carrying communication channels between the customer side and the developer side. This is a goal for the ***QuASE*** project (***Qu***ality-***A***ware ***S***oftware ***E***ngineering) [19, 25] established in cooperation with two local software development companies.

The part of the above problem addressed in this paper is the issue of establishing the conceptual representation for the process of quality-related stakeholder interaction. We propose to base such representation on the common conceptualization of the quality itself which needs to be flexible enough to be able to reflect different semantics acquired

H.C. Mayr et al. (Eds.): UNISCON 2012, LNBIP 137, pp. 73–86, 2013.

by quality on different stages of this process and for different participants; we define the set of concretizations of the quality concepts reflecting the variety of these semantics.

The paper is structured as follows. Section 2 describes the case study and outlines the applied empirical methodology. Section 3 defines basic quality-related and process-related concepts and conceptualizes the stages of the interaction process. Section 4 describes the introduced concepts using a formal set-builder notation. It is followed by the description of the related work, the conclusions, and the future research directions.

2 The Case Study

We are going to base our explanation on the following case study. We investigate the software process for the IT firm (company *YY*) working with the customers in social care field to make their support systems accessible to the mobile end users (e.g. social workers). For this category of projects, customer organizations already have their own established support systems (with IT departments supporting these systems) which are made extendable by providing the interface which needs to be utilized by *YY* to develop their solution. The following research questions, among others, were stated in a process of the free-form detailed interviewing of the IT staff of the company:

1. Which participants are involved in which stages of the interaction process?
2. Is the quality understood differently on different stages of the interaction process and by different participants? If so, what are the differences?
3. Is the process represented differently for different categories of stakeholders? If so, what are the differences?
4. What is the character of the initial preconceptions on quality possessed by the customers on the start of the negotiations?
5. What could be the distance between the quality initially requested by customer and the quality specified in a contract after the negotiation?

The following conceptualization is based, in part, on the transcribed answers to these questions; to obtain this, we applied the basic techniques of *coding* and *conceptualization* stages of the grounded theory [3, 8]. We do not claim applying the complete process of grounded theory as defined e.g. in [23]; this will be the target of future research with a goal of establishing the conceptualization of the whole process of stakeholder perception of quality; for this purpose, additional cases are planned to be involved. The obtained evidence is also combined with our own experience and the knowledge incorporated into the software quality standards i.e. ISO/IEC 25010 [15].

3 The Stakeholder Interaction Process

3.1 Basic Quality-Related Definitions

We start the conceptualization of the process in question from the definition of the basic concept of quality. This definition follows [24] in relying on the body of work of formal ontology, represented by DOLCE [21] and CORE [17] ontologies.

In the subsequent description, we use underlined bold italic font with the appropriate capitalization for the concepts introduced for the first time and underlined italic font for the concepts mentioned after their introduction.

The Concept of Quality. Following and extending our earlier work [24], we define four components of the concept of quality of the SUD (software under development):

1. A set of involved ***Quality Characteristics*** (e.g. "performance" or "reliability") understood per DOLCE [21, pp. 16-18] as the categories for perceivable and measurable entities characterizing particular individuals. Every *Quality Characteristic* is accompanied by a set of ***Quality Spaces*** - conceptual spaces per Gardenfors [11]. These spaces consist of perceivable and measurable ***Quality Dimensions*** (e.g. "throughput" or "response time" for performance) with associated ***Metrics*** defining how the individuals are positioned according to the particular dimension (e.g. "the metric for calculating the throughput"). Shared *Quality Spaces* indicate a common agreement about the way of measuring *Quality Characteristic* e.g. "the response time is measured in seconds"; private *Quality Spaces* indicate the lack of such agreement e.g. "the reliability can be high or low".
2. A set of relationships among particular *Quality Characteristics* (such as those specifying their hierarchy or interdependencies).
3. A set of relationships among *Quality Characteristics* and ***Quality Subjects*** (participants of a quality-related software process activity, e.g. business stakeholders). Such relationships can e.g. define stakeholder speech acts [17, p. 181] connecting private quality spaces to stakeholders. In this, they follow CORE by categorizing ways of perceiving quality by quality subjects as a part of communicated information. Examples of speech acts are directive acts (ordering something to be done) which define precise quality constraints (if the accompanying *Quality Space* is shared) or qualitative softgoals [7] if such *Quality Space* is private.
4. A set of relationships among *Quality Characteristics* and ***Quality Objects*** (elements of a SUD decomposition the *Quality Characteristics* to be connected to, e.g. functional entities defined by a SUD architecture).

An example of the latter two relationships is the constraint "the response time (*Quality Dimension*) of the component *M* (*Quality Object*) must be below 0.5 sec according to the stakeholder *K* (*Quality Subject*)". Based on the above definitions, we introduce the ***SUD Quality*** concept as the tuple comprising all four abovementioned sets.

Quality Facets and Incarnations. Based on the above concept, we provide additional definitions necessary for our conceptualization. In a way of defining more specific views of quality such as the quality at specific stages of the software process, as seen by specific stakeholders etc., we define ***Quality Facet*** as a *SUD Quality* defined for a particular subset of *Quality Characteristics*, *Quality Subjects*, and *Quality Objects*. Also we define facet-specific set of *Quality Spaces*.

The ***Quality Point*** in *Quality Space* defines a position of the *Quality Object* according to this space's *Quality Dimensions* (e.g. "a position of the component *M* according to the throughput"). After specifying a single *Quality Point* in every *Quality Space* defined for a *Quality Facet* and taking a set of all such *Quality Points* we can

define the ***Quality Incarnation*** by connecting this set to a specific software process activity and a specific purpose to serve. It concretizes the *SUD Quality* by conceptualizing particular quality levels at particular stages of the software process. An example of such incarnation can be "the (particular) performance and reliability of the component *M* perceived by the stakeholder *K* at the start of the negotiation". This concept is illustrated on Fig.1 together with the concepts it depends upon.

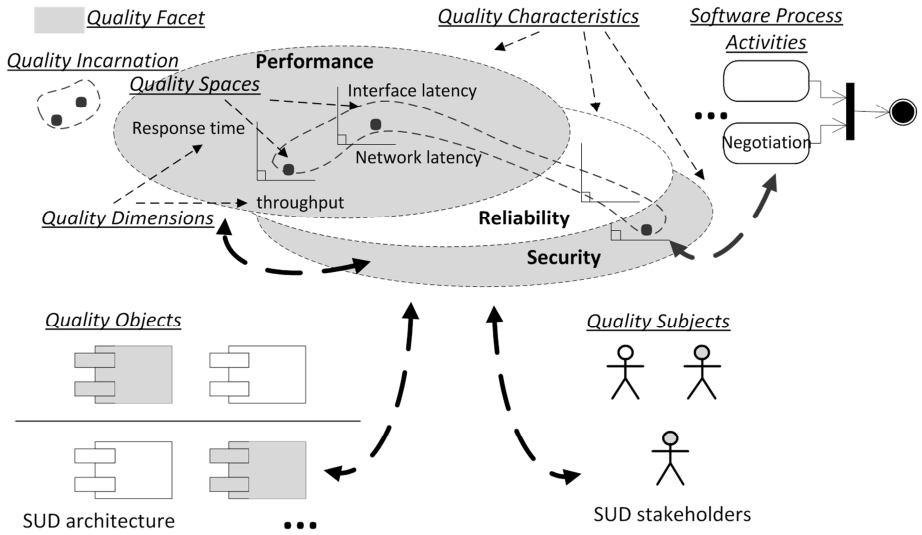

Fig. 1. Basic quality-related concepts

3.2 Basic Interaction-Related Definitions

Quality Realms. We propose to treat all the quality concepts emerging on different stages of the interaction process and all the interactions with stakeholders involving these quality concepts as belonging to specific ***Quality Realms*** defined as particular subdomains (fields of interest, concerns) in the software engineering domain uniformly influencing the treatment of the software quality for all the involved activities and participants. The quality concepts emerging at different stages of the interaction process belong to two *Quality Realms*:

1. ***User Satisfaction Realm*** for the relevant activities and participants related to the interests of the SUD stakeholders; for this realm, the quality is treated from the point of view of their satisfaction (quality in use [14, 15]);
2. ***Implementation Realm*** for the relevant activities and participants related to the interests of the IT people implementing the SUD; here, the quality is treated as the set of objective characteristics of the system (e.g. its external quality [13, 15]).

Process Participants. For our case, we define three main categories of participants:

1. ***Business Stakeholders*** do not necessary have an IT experience; they are end users of the SUD. They do not always interact with the *Developers* directly. Both their understanding of quality and their role in the quality interaction process belong to the *User Satisfaction Realm*.
2. ***Developers*** are the IT professionals responsible for developing of the software system itself. Both their understanding of quality and their role in an interaction process belong to the *Implementation Realm*.
3. ***Software Integrators*** are the IT professionals responsible for running the software platform deployed at the customer organization's site, they perform integration of the solutions provided by *Developers* into that platform. This role is twofold: on the one hand, they perceive the quality as IT professionals (i.e. as belonging to the *Implementation Realm*), on the other hand, they are the prospective customers of the SUD so their participation in quality-related stakeholder interaction belongs to the *User Satisfaction Realm*.

As we can see, there are two ***Stakeholder Roles*** which need to be defined as belonging to the particular *Quality Realm*:

1. related to the stakeholder perception of quality (distinguishing ***Business Thinkers*** and ***IT Thinkers***);
2. related to the stakeholder participation in the interaction process (distinguishing ***Business Actors*** and ***IT Actors***).

Quality realms and stakeholder roles are depicted on Fig.2.

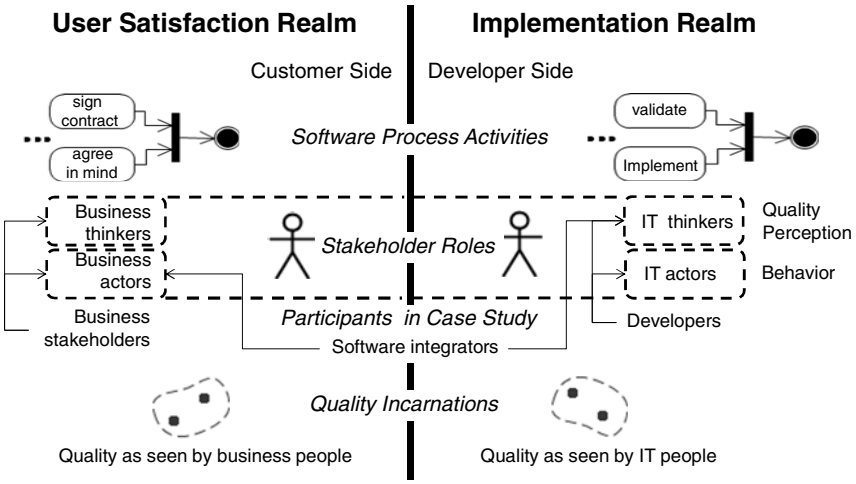

Fig. 2. Quality realms and stakeholder roles

3.3 Coarse-Grained Quality Awareness Process

Continuous Quality Awareness. Before elaborating the quality negotiation process, it is necessary to explain its place in the complete development lifecycle. To do this, we need to explain the concept of the ***Continuous Quality Awareness*** defined as the

Developers ability to check, throughout the software process, the desired *SUD Quality* (from the *Business Actor* point of view) against the current implementation stage.

To facilitate *Continuous Quality Awareness*, it is necessary to establish the set of ***Awareness Support Activities*** to be performed (triggered) continuously on different software process to provide the means of getting the correct stakeholder opinions on quality. Quality-related interaction can be considered such an activity.

Process Outline. In defining the process of interaction, we will proceed on two levels. First, we define the coarse-grained interaction process (***Awareness Support Process***). This process extends the general software process by introducing a means of triggering the more specific quality negotiation activities if necessary.

Awareness Support Process relies on the concept of the ***Interaction Trigger***. Such trigger corresponds to some event which initiates executing the set of actions for the quality-related interaction. We can distinguish three possible ways of defining such triggers:

1. Making the set of triggers correspond to the predefined points in the software process, e.g. they can be fired at its milestones; this way, the exact time of firing is known beforehand;
2. Making the triggers correspond to the predefined events in the software process, e.g. defining them as firing after making all important design decisions (e.g. "introducing new caching solution which could affect the performance"); this way, though the exact time of firing cannot be known in advance, we can know in advance the conditions of firing; these triggers are fired at the changes of the ***SUD State*** understood as the set of internal variables affecting *SUD Quality*.
3. Defining ad-hoc (or on-demand) triggers that can be fired on demand of the participating actor; this way, neither the time nor the condition can be known in advance; with such triggers, real *Continuous Quality Awareness* can be achieved (stakeholders opinions on quality can be taken into account at any time).

Now we can define the set of actions to be executed on firing the interaction trigger. We distinguish four basic coarse-grained activities:

1. ***Preparation Activity*:** the necessary preliminary actions are performed (e.g. initial version of the *SUD Quality* is produced);
2. ***Negotiation Activity*:** the sides agree to the particular incarnation of *SUD Quality* (*Pre-Implementation Negotiation*);
3. ***Implementation Activity*:** as a result of the implementation activities the SUD is transitioned to some next state possessing the particular *SUD Quality*;
4. ***Checking Activity*:** the implemented *SUD Quality* is compared against the negotiated one; if necessary, *Post-Implementation Negotiation* is performed; it is also possible to perform e.g. checking the direction of overall quality trend (indicated by e.g. the distance between the initial and agreed versions of *SUD Quality*).

We can distinguish two different sequences of coarse-grained activities depending on the type of the *Interaction Trigger*. For the triggers of the first and second kind, i.e. with firing conditions corresponding to the structure of the software process (e.g. the development milestones or design decisions) the flow of activities is depicted on

Fig.3. Here the *Checking Activity* is executed at the end of the sequence (after the *Implementation Activity*) to check if the result of the implementation achieved the agreed *SUD Quality*.

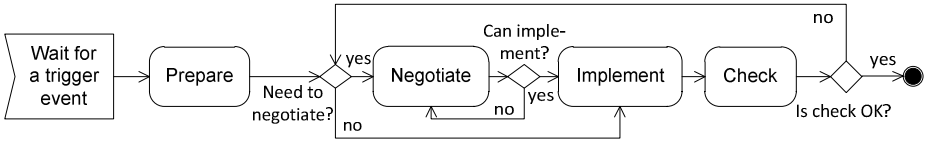

Fig. 3. Coarse-grained activities to be executed at milestone or design decision

For on-demand triggers, the sequence of activities is depicted on Fig.4. Here it is necessary to execute the *Checking Activity* on the trigger event first to check if the current *SUD Quality* corresponds to its agreed values. In this situation, both the *Negotiation Activity* and the *Implementation Activity* need to be executed only if the check was not successful.

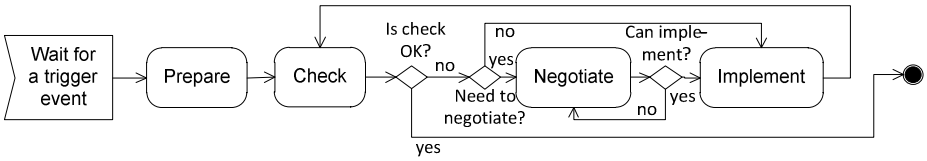

Fig. 4. Coarse-grained activities to be executed on demand

3.4 Preparation Activity and Corresponding Quality Incarnations

Before describing this activity in detail, we describe the corresponding *Quality Incarnations*. We start with those belonging to the *User Satisfaction Domain*. The informal schema of the fine-grained activities and the corresponding *Quality Incarnations* is depicted on Fig.5.

Expected Quality and Its Boundaries. The ***Expected Quality*** can be defined as the *Quality Incarnation* representing an image of the *SUD Quality* that is conceived of by the individual *Business Actor* as a negotiable quality he/she wants the final SUD version to provide. Such *Quality Incarnation* needs to be taken into account every time when the software process deals with *Business Actors*' opinions, but it exists only in a *Business Actor*'s mind (e.g. "now I (the stakeholder *K*) think that the response time as seen through the user interface (UI) element *L* may not exceed 0.5 sec".) It can change throughout the interaction process as a result of the evolution of *Business Actor*'s quality vision. It belongs to different *Quality Realms* for the interactions with *Business Thinkers* and *IT Thinkers*.

Expected Quality Boundaries are the *Quality Incarnations* reflecting the fact that the *Business Actor* enters the process of interaction with specific quality-related pre-conceptions in mind constituting the *Expected Quality* permissible range*;* it depends on the *Business Actor*'s capabilities to accept the *SUD Quality*. The ***Best-Case***

Quality is the higher bound of the *Expected Quality*: if the *SUD Quality* as perceived by *Business Actor* equals or exceeds the *Best-Case Quality*, the particular SUD version can be immediately accepted as completely satisfying the *Business Actor*; no additional interactions are necessary. The ***Last-Resort Quality*** is the lower bound of the *Expected Quality*: if the *SUD Quality* is below the *Last-Resort Quality* boundary, the SUD version can be immediately rejected as dissatisfying the *Business Actor*: e.g. "if I (the stakeholder *K*) will get the response time seen through the UI element *L* of above 15 sec I will think twice about dealing with these developers again".

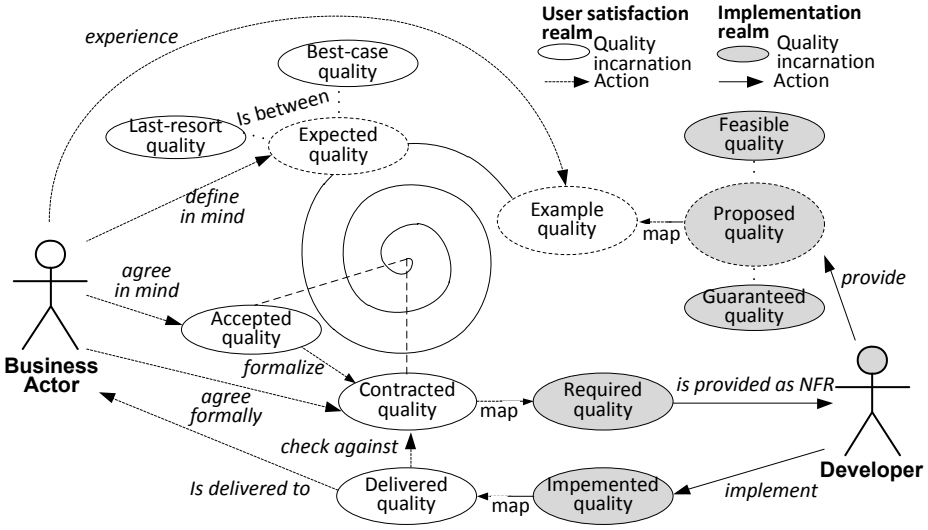

Fig. 5. Fine-grained interaction activities and corresponding quality incarnations

Proposed and Example Quality and Their Boundaries. We define our conceptualization for the case when the interaction with *Business Actor* is performed from the *Developer* side by making the *Business Actor* experience some version of quality which could be then the subject of negotiation: a ***Proposed Quality***. It denotes a set of values for quantified quality attributes of the SUD reflecting the current state of its development. The SUD is expected to have that quality if implemented exactly as defined at the given development stage. *Proposed Quality* can be produced based on a (e.g. simulation) model, a system prototype, or the complete system version. An example could be "the vertical prototype (or the performance simulation) of the component *M* shows the response time of 0.34 sec in the context *T*").

From the *Business Thinker* side, it is difficult to experience and assess the *Proposed Quality* directly as it is just a set of (objective) values which reflects the current state of the SUD (*external quality* in terms of [13, 15]), so it belongs to the *Implementation Realm* (e.g. "the component *M* is the element of the SUD architecture not directly visible to the stakeholder *K*"). Following [22], we propose to map this *Quality Incarnation* into the one belonging to the *User Satisfaction Realm*: an ***Example Quality***. Its values need to reflect the quality in use according to e.g. ISO/IEC 9126-4 [14] and the future version of ISO/IEC 25024; they can be experienced by *Business*

Thinker naturally (e.g. "map the response time of the component *M* into the response time as shown through the UI element *L* which is understandable by the stakeholder *K*"). This mapping is not necessary in case of interacting with *IT Thinkers*.

Proposed Quality boundaries are the *Quality Incarnations* reflecting the range of the *Developer* capabilities to produce the SUD with the particular level of quality. The higher bound (***Feasible Quality***) reflect the best quality that can be produced given the available resources, whereas the lower bound (***Guaranteed Quality***) reflects the level of skills and self-esteem of *Developers* which makes producing the SUD with lower quality unacceptable for them.

Preparation Activity. Launching this activity entails executing the following fine-grained activities necessary for the preparation of the participants to the quality-related stakeholder interaction.

1. ***Establish Expected Quality Boundaries***: *Business Stakeholder* establishes in his/her mind both *Best-Case Quality* and *Last-Resort Quality*.
2. ***Establish Proposed Quality Boundaries***: *Developer* learns about or establishes both *Feasible Quality* and *Guaranteed Quality*;
3. ***Produce Initial Proposed Quality***: *Developer* produces some version of *Proposed Quality* (through e.g. SUD prototype) falling between its boundaries; it is then mapped into *Example Quality* if necessary; to be concise, below we refer to such quality (either mapped or unmapped) as *Example Quality.*

3.5 Negotiation Activity

Executing the *Negotiation Activity* entails executing the set of fine-grained activities:

1. ***Experience Proposed Quality*:** the interaction begins from the *Developer* making *Business Actor* experience the produced *Example Quality*, after this, *Business Actor* compares it in his/her mind to the *Expected Quality* boundaries. As a result, two mutually exclusive activities can be executed:
   - ***Accept the SUD State*:** If the *Example Quality* exceeds the *Best-Case Quality*, the SUD state possessing the particular *Example Quality* is accepted without further interactions; such experience can be of the great service to the project as it establishes the feeling of trust between *Business Actors* and *Developers*.
   - ***Reject the SUD State*:** if the *Example Quality* falls below the *Last-Resort Quality*, the particular version of the SUD is rejected without the possibility of further interaction; it can be damaging to the whole project.
2. ***Negotiate Specific Quality*:** If the *Example Quality* falls into the acceptable *Expected Quality* range, the process follows the ***Negotiation Iterations***; on every such iteration, the following activity is performed:
   - ***Adjust the Proposed Quality* and *Expected Quality*:** *Developer* makes *Business Actor* experience the new version of *Example Quality* and *Business Actor* assesses his/her experience again; in doing this, every side tries to insist on some preferable quality values: in particular, *Developer* tries to keep the *Example Quality* closer to

the *Guaranteed Quality* to spend fewer resources, whereas *Business Actor* tries to keep the *Expected Quality* closer to the *Best-Case Quality* as it is better correspond to what he/she has in mind initially. Throughout this iterative process both sides learn about the capabilities of each other: as a result, their *Quality Incarnations* converge to the agreement point. It could look like: "the developers show the stakeholder *K* (through the UI element *L*) the vertical prototype of the component *M* with a (UI-mapped) response time of 0.7 sec, *K* does not quite like it, so they produce a new version with a response time of 0.5 sec and *K* agrees on that".

3. ***Agree Upon Quality*:** the negotiation process eventually ends with agreeing upon a version of the *Example Quality* shown to the *Business Actor*: the ***Accepted Quality***.

The above activities are defined for the case when the negotiation between *Business Actor* and *Developer* takes place through a sequence of *Example Quality* incarnations shown to *Business Actor* and an evolution of the *Expected Quality* incarnations in *Business Actor*'s mind; in reality, simpler sets of activities could be performed (e.g. missing the second step so the initial *Example Quality* is either immediately accepted or immediately rejected). Main concepts related to this activity are shown on Fig.6.

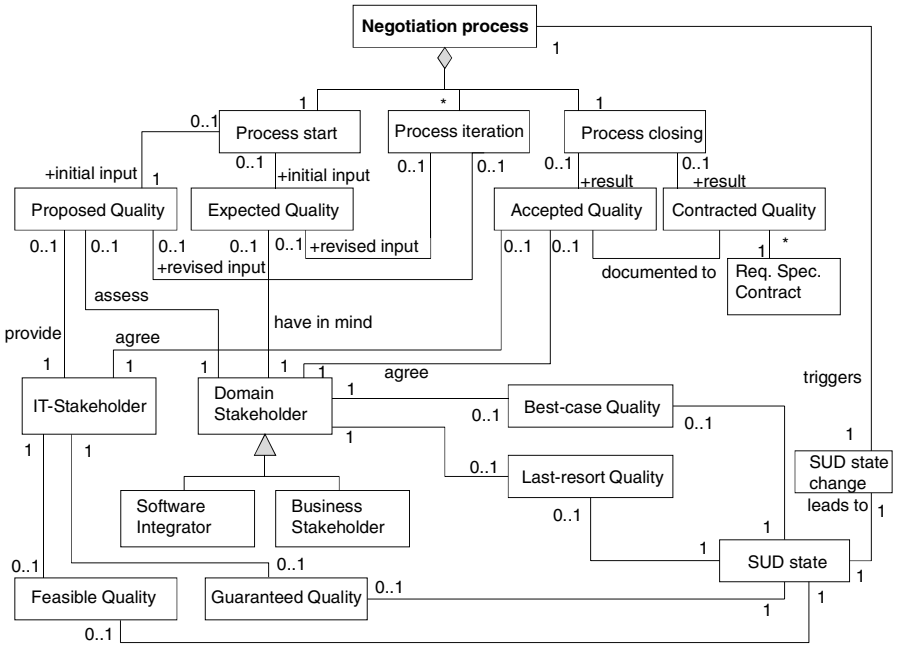

Fig. 6. Main concepts of the stakeholder negotiation activity

3.6 Implementation and Checking Activities

Contracted and Required Quality. As a result of the iterative interactions, the sides agree on *Accepted Quality*. Before communicating it to the *Developer*, it is necessary to transfer it into the official, documented form: a ***Contracted Quality*** (e.g. "the

contract states that the response time seen through the user interface element *L* must not exceed 0.5 sec"). For interacting with *Business Stakeholders*, it needs to belong to the *User Satisfaction Realm*, this is not necessary for the interaction with *IT Thinkers*.

In case of *Business Stakeholder*-involving interaction, the *Contracted Quality* cannot be directly used to guide the software development activities to be performed by *Developers*. To do so, it needs to be mapped into ***Required Quality*** belonging to the *Implementation Realm*, it can be seen as the quality expressed in the requirement specification contract as a set of SUD non-functional requirements (NFR); so this mapping can be seen as a part of the NFR elicitation process (e.g. **"**the NFR *R* states that the response time of the component *M* must not exceed 0.5 sec**"**). For the interaction with *IT Thinkers*, *Contracted Quality* can be directly used as *Required Quality*.

Implementation Activity and Related Quality Incarnations. After the *Required Quality* was defined and made available to the *Developers*, the necessary implementation-related actions are performed as a part of coarse-grained *Implementation Activity*. As a result, the SUD transitions itself into the new development-related state (corresponding to e.g. the milestone). We refer to this state as possessing the ***Implemented Quality*** belonging to the *Implementation Realm*; in case of interaction with *Business Stakeholders* it needs to be mapped into the *User Satisfaction Realm* to obtain the ***Delivered Quality*** (e.g. "we delivered the SUD version with the response time of 0.45 sec as seen through the UI element *L*").

Checking Activity. After the *Implementation Activity* is completed (or right on a trigger event in case of asynchronous launching of the interaction), the obtained *Delivered Quality* needs to be experienced by the *Business Actors* as a part of the *Checking Activity*. Here, various checks can be performed, the most common one is checking if *Delivered Quality* corresponds to the *Contracted Quality*. If this is the case, next round of the *Awareness Support Process* can be started by executing the *Preparation Activity* aligned with the next ***SUD State Change*** (e.g. with the process stage bounded by the next milestone), otherwise it corresponds to the major development problem with a need of emergency actions.

4 Formal Description

In this section, we use set-builder notation to describe the introduced concepts. A *SUD Quality* is defined as a tuple $Q(C,U,F)=\langle C,R_q(C),R_s(C,U),R_f(C,F)\rangle$, where $C=\{c=\langle P(c),S(c)\rangle\}$ is a set of involved *Quality Characteristics* (comprised of a set of properties $P(c)$ and a set of *Quality Spaces* $S(c)$), $R_q(C)=\{r_q(C'\subseteq C)\}$ is a set of relationships among *Quality Characteristics* (*C'* is a set of involved characteristics, *C* is the set of available characteristics), $R_s(C,U)=\{r_s(C'\subseteq C,U'\subseteq U)\}$ is a set of relationships among *Quality Characteristics* and *Quality Subjects* (*U'* is a set of involved subjects, *U* is the set of available subjects), $R_f(C,F)=\{r_f(C'\subseteq C,F'\subseteq F)\}$ is a set

of relationships among *Quality Characteristics* and *Quality Objects* (*F'* is a set of involved objects, *F* is the set of available objects).

A *Quality Facet* is defined as $\overline{Q} = Q\left(\overline{C} \subseteq \mathbf{C}, \overline{U} \subseteq U_{BS} \cup U_{SI} \cup U_{ITS}, \overline{F} \subseteq \mathbf{F}\right)$, where **C** and **F** are the sets of existing *Quality Characteristics* and *Quality Objects*, U_{BS}, U_{SI}, U_{ITS} are the sets of participants belonging to the particular categories. In addition, $\overline{S}(\overline{Q}) = \left\{\bigcup S(c) \middle| c = \langle P(c), S(c)\rangle \in \overline{C}\right\}$ defines a facet-specific set of *Quality Spaces*.

A *Quality Incarnation* for the interaction process activity *a* is defined as a tuple $I(\overline{Q}) = \left\langle P(I(\overline{Q})), \left\{v \middle| s(v) \in \overline{S}(\overline{Q})\right\}, a \in A\right\rangle$, where $P(I(\overline{Q}))$ is the set of properties, *A* is a set of defined process activities, $s(v)$ is a *Quality Space* for a *Quality Point* *v*.

5 Related Work

Our research is related to both quality conceptualization and software process conceptualization so we need to consider the state of the art in these two research fields.

We published a detailed literature review of the available quality conceptualization techniques in [24], so we refer to this paper for the complete treatment of this issue. Among these techniques, it is possible to distinguish model-based [6, 9, 10, 27, 28] and ontology-based [16-18, 20] approaches, for the purpose of our research we are mostly interested in the latter. We found that these techniques are not completely suitable for our problem as they mostly concerned with conceptualizing the quality itself. They, as a rule, do not relate the proposed conceptualizations to the software process activities as seen from the both sides of the development process while performing the quality-related negotiation (e.g. they do not include the concept of quality realm or conceptualize the inter-realm mappings, also, the negotiation support is limited).

Software process conceptualization techniques [1, 2, 4, 26], on the other hand, often miss appropriate quality treatment: they do not include the notions of quality related to different software process activities. Among these techniques, we distinguish the work by Adolph and Kruchten [4] where they propose the conceptualization of the software process centering on the notion of the negotiation between the sides of the development process with a purpose of reaching common understanding; being the high-level conceptualization, it does not include the specifics of the negotiation process related to quality-carrying communication. We can see our research as an extension to that work focusing on establishing quality-carrying communication channels.

6 Conclusions and Future Work

In this paper, we established the conceptualization for the process of quality-related stakeholder interaction in software development. As basic notions, it provides the concept of *SUD Quality* (based on the notions of formal ontology) together with the conceptual notions of *Process Participant*, *Quality Realm*, and the set of quality values (*Quality Incarnation*). The proposed process conceptualization is based on the

notion of *Continuous Quality Awareness* defining the set of interaction activities as an extension of the generic software process making possible on-demand stakeholder involvement with a purpose of checking the current *SUD State*. It defines the necessary process activities, the relevant workflow, and *Quality Incarnations* related to these activities. Its advantages are as follows:

1. It establishes a common ground for organizing the knowledge about the specifics of establishing quality-carrying communication channels into the semantic repository – a knowledge base. This information could be reused directly while organizing future interactions; it also could be used to predict the behavior of the parties while encountering similar quality-related issues;
2. It facilitates coming to a common language by different parties in the software process by providing the set of common quality-related concepts that could be utilized by all these parties while participating in quality-related interaction activities;
3. It can help in establishing process-oriented support solutions facilitating quality-related stakeholder involvement into the software process.

In future, in a framework of the QuASE project [19, 25], we plan to integrate the proposed conceptualization of the interaction process into a common ontology (*QuOntology*) which has to incorporate the knowledge about software quality and its perception by business stakeholders and developers, the processes of stakeholder quality assessment, the ways of producing quality to be proposed to stakeholders and other knowledge related to this domain. After establishing QuOntology, following the Ontology-based Software Engineering paradigm [5, 12], we plan to make it play a central role in implementing the tool support for the stakeholder involvement into the software process by e.g. facilitating semantic-based reuse of the information about quality-related interactions involving business stakeholders or the prediction of the stakeholder behaviour for the future interactions of this kind.

References

1. Acuna, S.T., Juristo, N. (eds.): Software Process Modeling. Springer, Heidelberg (2005)
2. Acuna, S.T., Sanchez-Segura, M.I. (eds.): New Trends in Software Process Modeling. World Scientific, New York (2006)
3. Adolph, S., Hall, W., Kruchten, P.: Using grounded theory to study the experience of software development. Empirical Software Engineering 16, 487–513 (2011)
4. Adolph, S., Kruchten, P.: Reconciling Perspectives: How People Manage the Process of Software Development. In: AGILE 2011, pp. 48–56. IEEE, New York (2011)
5. Bachmann, A., Hesse, W., Ru, A., Kop, C., Mayr, H.C., Vohringer, J.: A Practical Approach to Ontology-based Software Engineering. In: EMISA 2007. LNI, vol. P-119, pp. 129–142. GI, Bonn (2007)
6. Carvallo, J.P.: Systematic Construction of Quality Models for COTS-Based Systems. PhD Thesis. Universitat Politècnica de Catalunya, Barcelona (2005)
7. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston (1999)
8. Coleman, G., O'Connor, R.: Investigating software process in practice: A grounded theory perspective. The Journal of Systems and Software 81, 772–784 (2008)

9. Cote, M., Suryn, W., Georgiadou, E.: Software Quality Model Requirements for Software Quality Engineering. In: 14th International Conference on Software Quality Management, Quebec, pp. 31–50 (2006)
10. Deissenboeck, F., Juergens, E., Lochmann, K., Wagner, S.: Software quality models: Purposes, usage scenarios and requirements. In: WoSQ 2009, pp. 9–14. IEEE, New York (2009)
11. Gärdenfors, P.: Conceptual Spaces: A Geometry of Thought. MIT Press, Cambridge (2000)
12. Hesse, W.: Ontologies in the Software Engineering process. In: Proc. EAI 2005. Ceur-WS.org, vol. 141, pp. 3–16 (2005)
13. ISO/IEC 9126-2:2003: Software Engineering – Product Quality – Part 2: External Metrics. ISO (2003)
14. ISO/IEC 9126-4:2004: Software Engineering – Product Quality – Part 4: Quality-in-Use Metrics. ISO (2004)
15. ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. ISO (2011)
16. Jureta, I.: Essays in Information Management: Contributions to the Modeling and Analysis of Quality in Information Systems Engineering, PhD Thesis. Department of Business Administration. University of Namur (2008)
17. Jureta, I., Mylopoulos, J., Faulkner, S.: A core ontology for requirements. Applied Ontology 4, 169–244 (2009)
18. Kabilan, V., Johannesson, P., Ruohomaa, S., Moen, P., Herrmann, A., Ehlfeldt, R.M., Weigand, H.: Introducing the Common Non-Functional Ontology. In: Gonçalves, R.J., Müller, J.P. (eds.) Enterprise Interoperability II, pp. 633–645. Springer, Heidelberg (2007)
19. Kaschek, R., Kop, C., Shekhovtsov, V.A., Mayr, H.C.: Towards Simulation-Based Quality Requirements Elicitation: A Position Paper. In: Rolland, C. (ed.) REFSQ 2008. LNCS, vol. 5025, pp. 135–140. Springer, Heidelberg (2008)
20. Masolo, C., Borgo, S.: Qualities in formal ontology. In: Foundational Aspects of Ontologies Workshop at KI 2005, Koblenz, pp. 2–16 (2005)
21. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: The WonderWeb Library of Foundational Ontologies. WonderWeb Deliverable D18. Ontology Library (final). . ISTC-CNR, Trento (2003)
22. Raubal, M.: Formalizing Conceptual Spaces. In: Proceedings of Formal Ontology in Information Systems, FOIS 2004, pp. 153–164. IOS Press, Amsterdam (2004)
23. Sharmaz, K.: Constructing Grounded Theory. Sage Publications, Thousand Oaks (2006)
24. Shekhovtsov, V.A.: On the evolution of quality conceptualization techniques. In: Kaschek, R., Delcambre, L. (eds.) The Evolution of Conceptual Modeling. LNCS, vol. 6520, pp. 117–136. Springer, Heidelberg (2011)
25. Shekhovtsov, V.A., Mayr, H.C., Kop, C.: Stakeholder Involvement into Quality Definition and Evaluation for Service-Oriented Systems. Accepted for Publication in Proc. ICSE 2012 Workshops. IEEE, New York (2012)
26. Software Process Engineering Metamodel (SPEM) 2.0. OMG (2008)
27. Tian, J.: Quality-Evaluation Models and Measurements. IEEE Software 21, 84–91 (2004)
28. Wagner, S., Lochmann, K., Winter, S., Goeb, A., Klaes, M.: Quality models in practice: A preliminary analysis. In: ESEM 2009, pp. 464–467. IEEE, New York (2009)