V. Shekhovtsov, H.C. Mayr, Managing Quality Related Information in Software Development Processes. CAiSE-Forum-DC 2014, CEUR Workshop Proceedings, vol. 1164: CEUR-WS.org, 2014, pp. 73-80

Managing Quality Related Information in Software Development Processes

Vladimir A. Shekhovtsov, Heinrich C. Mayr

Institute for Applied Informatics, Alpen-Adria-Universität Klagenfurt, Austria
{Volodymyr.Shekhovtsov,Heinrich.Mayr}@aau.at

Abstract. An effective communication between the parties in the software development process is important for coming to and complying with appropriate agreements on the quality of the prospective software. Such communication is impaired when developers and business stakeholders perceive quality differently. To address this problem, we aim at a solution that supports understandability and reusability of quality-related communicated information, and the quality of decisions based on this information. In this paper, we first introduce a set of knowledge structures for representing communicated information and then discuss how to map raw communication data into these structures.

Keywords: elicitation, semantic annotation, software development process, software quality, communicated information

1 Introduction

Software development processes require a continuous involvement of the affected business stakeholders in order to be successful (this requirement, in particular, is reflected by the ISO/IEC standard for software life cycle processes [6]). A prerequisite of such involvement is establishing an appropriate communication basis for the different parties. In particular, such a basis is needed for coming to terms and agreements on the quality of the software under development. Without this, quality defects are often detected only when the software is made available for acceptance testing.

Clearly, besides of enabling effective communication, the communicated quality-related information has to be managed properly and made available during the software development lifecycle; moreover, as past-experience may help to take the right decisions, such information should be provided in a way that allows for easy access (e.g., via an issue management system) and analysis.

The QuASE project[1] [11] aims at a comprehensive solution for these issues. In particular, this solution will provide support for managing (1) the **understandability** of quality-related *communicated information*, (2) the **reusability** of that information, and (3) the **quality of decisions** based on that information.

[1] QuASE: Quality Aware Software Development is a project sponsored by the Austrian Research Promotion Agency (FFG) in the framework of the Bridge 1 program (http://www.ffg.at/bridge1); Project ID: 3215531

In this paper, we concentrate on the knowledge structures representing quality-related communicated information and on the mapping of raw communication data into these knowledge structures.

The paper is structured as follows. Section 2 introduces the sources of quality related information and defines the knowledge structures representing QuASE quality characteristics. Section 3 describes the mapping of communicated information into these knowledge structures. After a short discussion of related work in Section 4, the paper concludes with a summary and an outlook on future research (Section 5).

2 Knowledge Structures for Representing Quality Related Communicated Information

Usually, industrial software development projects keep communicated information within *repositories* such as

1. *project databases* controlled by issue management systems, e.g., JIRA [7], MantisBT [6] and others; such databases contain communicated information in form of so-called issues that (generalizing communication units as bug reports or feature requests) and related discussions;
2. *file-based repositories* containing meeting minutes, requirement and design specifications etc.; these files are usually kept in some kind of a directory tree under the control of configuration management systems; these documents, as a rule, are updated less frequently as compared to issues and the relevant discussion threads;
3. *wiki-based systems*.

Consequently, QuASE considers these types of repositories as sources of quality-related communicated information and therefore provides interfaces to them.

The raw data collected from these sources are interpreted and mapped into the QuASE *QuIRepository* the structure of which (defining a generic metamodel representing semantic relationships) is depicted in figure 1 and subsequently explained.

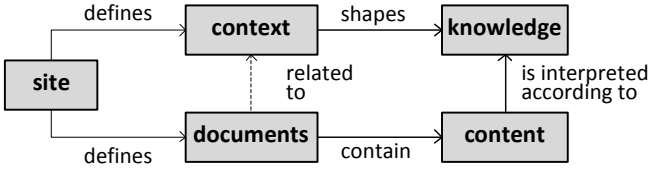

Fig. 1. Generic structure of QuIRepository

1. ***QuASE site:*** owner of the given QuASE installation, e.g. a software provider.
2. ***QuASE context:*** units having particular views on communicated information, e.g. projects, organizations and their departments, involved people (stakeholders) etc.. Context units are characterized by context attributes and can be connected to other units; a context configuration, for example, could include the representation of the

whole organizational hierarchy or the whole portfolio of projects defined for a particular IT company.

3. ***QuASE documents:*** units shaping communicated information: they serve as containers for such information or organize such containers. We distinguish *content holders* and *content directories* organizing the holders. Examples of document units are issues and their sets, issue attribute values, requirement specifications and their structural elements. For the case of issues, the issues or their sets are examples of content directories, whereas issue descriptions and discussion opinions are examples of content holders. Document units can be related to particular context element. A detailed description of the context and document concepts is target of a separate publication.
4. ***QuASE knowledge:*** quality and domain knowledge that is subject of communication and harmonization. We organize it into knowledge modules representing particular views. The configuration of these modules reflects the configuration of context, i.e., the modules and their relationships correspond to context elements and their interrelationships. Below, these modules will be described in more detail.
5. ***QuASE content:*** the information that has to be communicated. It is shaped by context units and interpreted according to the respective knowledge. Dealing with the content is decoupled from dealing with their holder documents; i.e., we can think of this content as of a uniform stream of data (which is given as tagged natural text in the current QuASE implementation). On the other hand, while dealing with documents, we abstract from their content and delegate dealing with this content to the generic content processing routines.

The QuASE knowledge modules are organized into a modular ontology (QuOntology) thus providing a framework for translating between world views. Initial research on QuOntology has been published in [13], whereas the current version of the relevant conceptualizations is presented in [11].

QuOntology is organized in three layers [see also Fig. 2]:

1. ***QuOntology core*** represents a stable subset of the knowledge available from research and industrial practice; this knowledge does not depend on the particular problem domain and the particular context. We use the Unified Foundational Ontology (UFO) [4, 5] as a foundation for QuOntology core.
2. ***Domain ontologies*** [5] represent the specifics of the particular problem domain which is addressed by the given software under development (finance, oil and gas etc.); domain ontology concepts specialize core concepts; as a part of the project, we implement for this layer an ontology for quality in the software domain [11].
3. ***Context ontologies*** represent the knowledge related to particular components of the QuASE context: they contain organization-specific, project-specific etc. concepts. These concepts specialize the generic concepts of the upper level ontologies but also may be specializations of other context ontologies; we implement for this layer ontologies for business-specific and IT-specific views on quality;

To deal with changes in the structure of context and document units, we will provide a notation for specifying context and document configurations, which will be support-

ed by the metamodeling tool ADOxx (http://www.adoxx.org); this will allow the responsible people (knowledge suppliers) to create and modify the desired configuration. The relevant database structure will be generated based on this configuration.

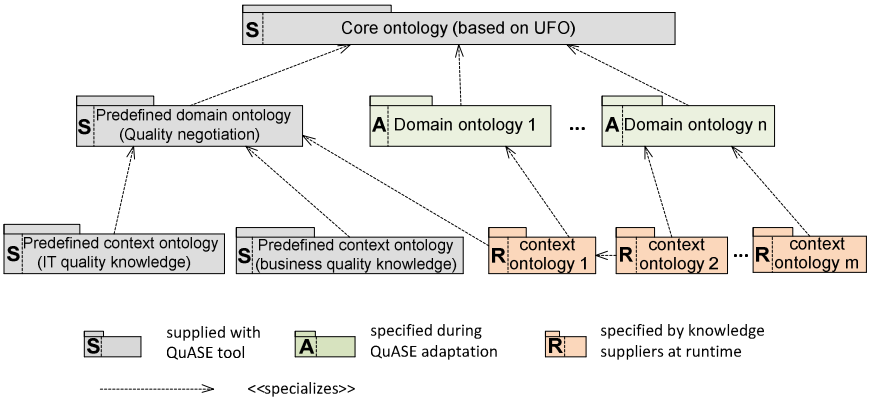

Fig. 2. QuASE ontology layers (adapted from [12])

3 Mapping communicated data into QuIRepository structures

The correspondence of communicated data and QuIRepository structures is made explicit by *mapping specifications* of a particular *mapping mode*.

For brevity, in this section we restrict ourselves to the mapping of repositories that are controlled by an issue management system (e.g. JIRA databases) further referred to as *mapping sources*. Mappings involving other categories of repositories (e.g. file-based repositories or wikis) are based on similar principles. Also, we omit the treatment of the mapping of concept relationships.

3.1 Mapping context structure

To define appropriate **mappings for the context concepts** we distinguish the following mapping modes:

1. ***Direct mode***: a given communicated context-related structure (e.g. a JIRA database table) is mapped one-to-one into a QuIRepository context structure;
2. ***Join mode***: several communicated context-related structures are mapped into a single QuIRepository context structure;
3. ***Split mode***: a single communicated context-related structure is mapped into several QuIRepository structures;
4. ***Interactive mode***: the whole instance of the context concept has to be specified by the user through the respective user interface.

Specifications for **mapping context attributes** are nested into the specifications defined for context concepts. We distinguish the following mapping modes:

1. ***Direct mode****:* a single communicated attribute (e.g. defined as an attribute in a context-related relation) is mapped into a single QuIRepository context attribute. The data is extracted without any user interaction. Example: mapping the "project name" attribute of a JIRA project table to the "project name" attribute of the corresponding QuIRepository "project" context unit;
2. ***Calculated mode****:* one or several communicated attributes are mapped to a QuIRepository context attribute based on a predefined metric function;
3. ***Interactive mode****:* the QuIRepository context attribute cannot be derived automatically from the communicated data; in this case, the QuASE tool shows an elicitation user interface and collects the concept information from the expert user.

3.2 Mapping document structures

1. ***Mapping document concepts****:* is defined similarly to the direct context mapping mode: the communicated document structure is mapped to a specific QuIRepository document structure. As an example, a JIRA "issue" table is mapped to the "issue" document structure, whereas the comments to the issues or issue descriptions are mapped into, correspondingly, "comment" or "issue description" content holders.
2. ***Mapping document attributes***: For the document attributes, the mapping is defined through the same three modes as specified above for context attributes; the main difference is due to the fact that it is possible to distinguish calculated attributes based on the content held directly by the document unit (for the case of content holders) or by the related holders (for the case of content directories).
3. ***Mapping content holders****:* For content holders, the approach is to delegate all the processing of mapping the content to the specific content-mapping activities such as text-based semantic annotation as specified in the following section.

3.3 Mapping content

Mapping content stream into QuASE concepts is performed by associating concepts with text fragments of the stream. To perform such an association, the QuASE system:

1. scans the natural language content stream looking for candidate context-specific terms;
2. associates tags with candidate terms that correspond to available knowledge in QuOntology; applying a tag indicates that the corresponding term can be associated with a QuOntology concept in at least one ontology module;
3. makes the tags act as anchors for connections to the related QuOntology concepts; to do this, for every tagged term the tool looks for concepts in all available context ontology modules.

The *Term knowledge context* then is the set of all concepts found for the given term; it defines all possible context-specific views of this term, and allows for switching between such views.

Fig.3 visualizes the process of associating context-specific terms with tagged documents exemplified by JIRA issues.

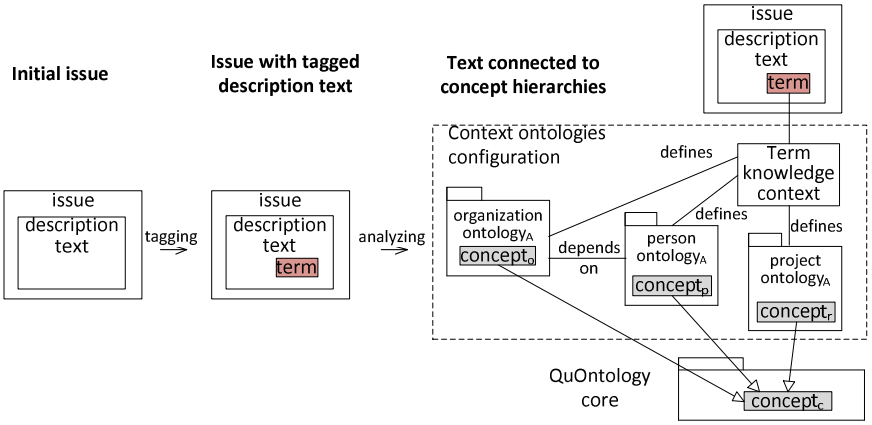

Fig. 3. Associating context-specific terms with tagged JIRA issues (domain ontology layer is omitted)

4 Related work

In this section, we discuss two categories of the related work: (1) approaches addressing the complete set of goals for QuASE, and (2) approaches addressing the particular task of obtaining the data from project repositories for analytical purposes.

4.1 Knowledge and experience management solutions

The approach discussed here belongs to the category of solutions that facilitate storing, reusing, adapting, and analyzing the development knowledge. In particular such solutions apply the existing body of research on knowledge management to the field of software engineering [1, 2]. A more specific category of solutions is related to managing past software engineering experience; they are known as experience management solutions [10].

With respect to our aims, these solutions bear the following shortcomings: (1) they do not specifically address quality-related issues, which is true especially for those issues that are available from existing repositories like issue management systems; (2) they collect the experience only as viewed from the developer side; the business stakeholder's view is mostly ignored, and it is not possible to switch between views while considering collected experience.

4.2 Solutions for obtaining information from project repositories

Approaches that aim at obtaining information from project repositories for analytical purposes, typically belong to the research area of *mining software repositories* [8]. Particular examples of such approaches include automatic categorization of defects [14], building software fault prediction models based on repository data [15], and using repositories to reveal traceability links [9]. Other approaches use repository information to analyze the applicability of specific development practices [3].

Repository mining solutions use software repositories as sources of quantitative code- and coding process-related information (such as the frequency of bugs, the time spent on various tasks, information about commits into repositories etc.). In contrast to that, QuASE uses repositories as sources for communicated information by looking into issue descriptions, negotiation opinions, wikis, and requirements documents.

In addition to the difference in the general goals, the QuASE approach differs from these solutions in the following implementation-related aspects: (1) it conceptualizes the process of collecting information from repositories as mapping operations controlled by mapping specifications; (2) it is based on an established set of conceptual structures that represent context and document units, content stream, and view-specific ontological knowledge.

5 Conclusions and future work

In this paper, we outlined a solution that is intended to support understandability and reusability of quality-related information, and thus may help to improve the quality of decisions in the software development process. The QuASE provides a knowledge-oriented interface to information that is communicated and collected in the course of software development projects. For this purpose, we introduced a set of knowledge structures addressing quality characteristics; these include context-, document-and content-specific structures as well as the structures for knowledge that defines particular views. We then defined the various kinds of modes for mapping communicated information (such as the data available in the project databases or document repositories) into these knowledge structures.

Ongoing research within the framework of the QuASE project aims at realizing the following features based on the defined conceptual structures:

1. *Understandability support:* document units are analyzed with respect to potential understandability problems for the target context (e.g., when they units are to be presented to a non-expert business stakeholder; identified problems are solved by translating or explaining the problematic terms using the respective knowledge structures.
2. *Reusability support:* for a given document, all similar ones (with respect to the required knowledge level) are searched based on the attributes of the documents and/or context units.
3. *Quality of decisions support:* recommendations for dealing with documents and context elements during the communication; forecasts of metrics values, and performing “what-if” analyses for particular decisions.

References

1. Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (eds.): Managing Software Engineering Knowledge. Springer, Berlin-Heidelberg (2003)
2. Bjørnson, F.O., Dingsøyr, T.: Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. Information and Software Technology 50, 1055-1068 (2008)
3. Ernst, N.A., Murphy, G.C.: Case studies in just-in-time requirements analysis. In: 2012 IEEE Second International Workshop on Empirical Requirements Engineering (EmpiRE). pp. 25-32. IEEE (2012)
4. Guizzardi, G.: Ontological foundations for structural conceptual models. University of Twente (2005)
5. Guizzardi, G., Falbo, R., Guizzardi, R.S.: 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 (2008)
6. ISO/IEC 12207:2008, Information technology – Software life cycle processes. pp. 122. International Organization for Standardization, Geneva (2008)
7. JIRA Issue Tracking System, http://www.atlassian.com/software/jira, accessed 08.05.2014
8. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice 19, 77-131 (2007)
9. Kagdi, H., Maletic, J.I., Sharif, B.: Mining software repositories for traceability links. In: ICPC'07. pp. 145-154. IEEE (2007)
10. Schneider, K.: Experience and Knowledge Management in Software Engineering. Springer, Berlin-Heidelberg (2009)
11. Shekhovtsov, V., Mayr, H.C., Kop, C.: Harmonizing the Quality View of Stakeholders. In: Mistrik, I., Bahsoon, R., Eeles, R., Roshandel, R., Stal, M. (eds.): Relating System Quality And Software Architecture. Elsevier (2014, in print)
12. Shekhovtsov, V.A., Mayr, H.C.: Towards Managing Understandability of Quality-Related Information in Software Development Processes. In: Murgante, B., Misra, S., Carlini, M., Torre, C., Nguyen, H.Q., Taniar, D., Apduhan, B., Gervasi, O. (eds.): Proc. ICCSA'14. Lecture Notes in Computer Science. Springer (2014, in print)
13. Shekhovtsov, V.A., Mayr, H.C., Kop, C.: 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 Information Processing, Vol. 137, pp. 73-86. Springer, Berlin-Heidelberg (2013)
14. Thung, F., Lo, D., Jiang, L.: Automatic defect categorization. In: WCRE'12. pp. 205-214. IEEE (2012)
15. Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R.: Mining software repositories for comprehensible software fault prediction models. Journal of Systems and Software 81, 823-839 (2008)