IMPLEMENTING TOOL SUPPORT FOR EFFECTIVE STAKEHOLDER COMMUNICATION IN SOFTWARE DEVELOPMENT – A PROJECT REPORT[1)2)]

Vladimir A. Shekhovtsov, Heinrich C. Mayr,
Sergii Ianushkevych, Matija Kucko,
Vladyslav Lubenskyi, Stefan Strell [3)]

Abstract
In this paper, we describe the implementation of the web-based QuASE tool aimed at supporting effective stakeholder communication in software development. The motivation for conducting the QuASE project aimed, in part, at producing the proof-of-concept implementation of this tool lies in the fact that the communication-related information available in the project repositories (such as Jira databases) is not always managed properly and is not made available for analysis during software development lifecycle. The tool supports the scenarios of managing the understandability of communicated information, the reusability of that information, and the quality of decisions based on that information. The tool is implemented following three-tier architecture, we introduce the tiers defined for this architecture and outline the implementation of the components comprising these tiers.

1. Introduction

This paper is devoted to the implementation aspects of the software tool developed as a result of the QuASE (Quality Aware Software Engineering) project. The motivation for the project is derived from the fact that software development processes require a continuous involvement of all the affected parties including business stakeholders. A prerequisite of such involvement is establishing an appropriate communication basis for these parties. Besides of enabling effective communication, the communicated 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 and analysis.

[1] Funded by Österreichischen Forschungsförderungsgesellschaft mbH (FFG), Projekt 838531

[2] Project partners: Institute for Applied Informatics, Alpen-Adria-Universität Klagenfurt, ilogs mobile software GmbH, CICERO CONSULTING GmbH, Bernhard Lanner GmbH, trinitec IT Solutions & Consulting GmbH

[3] Institute for Applied Informatics, Alpen-Adria-Universität Klagenfurt, Austria {volodymyr.shekhovtsov, heinrich.mayr, sergii.ianushkevych, mkucko, vladyslav.lubenskyi, sstrell}@aau.at

Our motivation has been supported by an extensive qualitative research in cooperation with the consortium partners. This allowed us to conceptualize (by ontological means) the process of harmonizing the stakeholder views on quality [8] and to see the need of establishing software support for such harmonization. The QuASE project aims at a comprehensive solution for these issues [11]. In particular, we provide proof-of-concept software support for managing (1) the understandability of communicated information, (2) the reusability of that information, and (3) the quality of decisions based on that information. Fig.1 presents different scenarios supported by that solution:

1. User A and B use the QuASE tool to achieve a common understanding of some aspects of a software under development that are described in a document X. In this scenario, the QuASE knowledge base provides the means of improving the understandability of the document by translating or explaining its content where needed. Fig.1 shows the case where a source document X_A of user A is transformed into a version X_B that can be understood by B.
2. User A uses the QuASE tool for decision support. As a means of such support, the QuASE Knowledge Base facilitates the analysis of previous communications and the parties involved there-in; such information may be reused to improve future communications, for predicting particular attributes, or for supporting decisions on the strategy of communication with the given party in the given context.

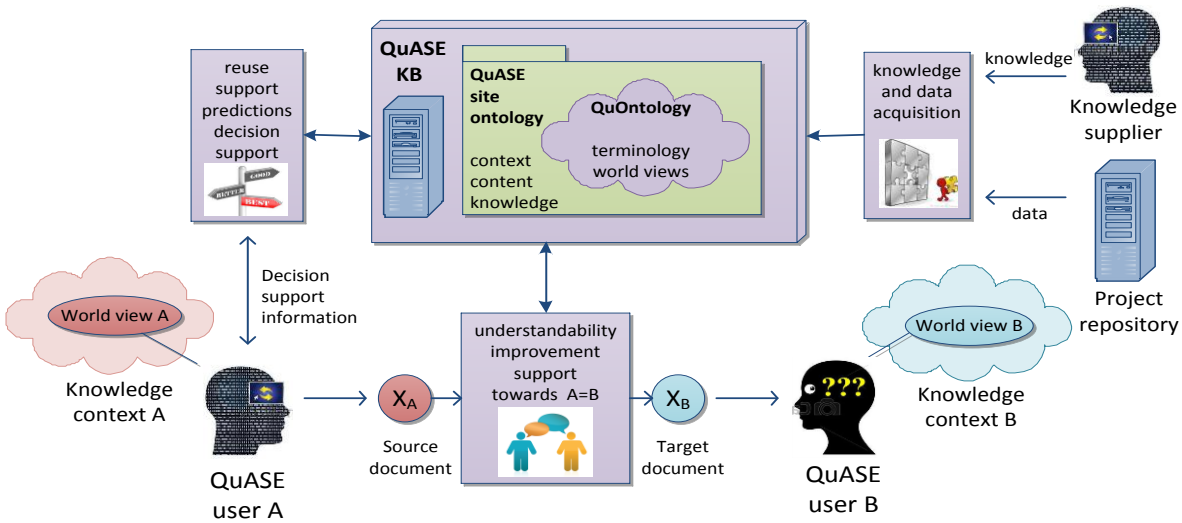

Figure 1. General activities of the QuASE approach

For supporting such scenarios, fundamental knowledge related to the communication domain has to be gathered and stored in the QuASE knowledge base: either automatically from the project repositories such as JIRA issue tracking databases, or interactively from knowledge suppliers. The QuASE knowledge base consists of two clearly separated parts:

1. The QuASE site ontology defining the site-specific communication environment (who communicates, which documents store the communicated information etc.);
2. The QuOntology containing knowledge related to the communicated information (both common and site-specific); it also can contain aset of core notions acting as a glue between the knowledge specific to different parties. This assures that the views of communicating parties can be harmonized.

The QuASE system includes the following components:

1. A metamodel-based *QuASE site modeling tool* (built on top of ADOxx metamodeling framework) supporting a graphical QuASE DSL for describing the site-specific communication environment (communication context, holders of communicated information, communicated knowledge) and the mapping between the model and the particular project repository (e.g. Jira database);
2. A converter (*QuASE ontology builder utility*) transforming the models expressed in QuASE DSL into OWL2 representations of the QuASE site ontology;
3. The knowledge acquisition engine (*QuASE knowledge base builder utility*) converting the data from project repositories into individuals corresponding to the established ontologies based on the mapping specified by means of QuASE site modeling tool;
4. The interactive web-based *QuASE terminology editor tool* acquiring the information related to communicated knowledge from knowledge suppliers and converting this information into the QuOntology-based representation;
5. The interactive web-based *QuASE tool* implementing the end-user support scenarios, as well as acquisition of the extra information not originally found in project repositories from knowledge suppliers (such as user-specified attribute values or the information about decisions connected to the elements of communication environment).

The implementation of the understandability management [10] support in QuASE applies natural language processing techniques. This is to localize the source of understandability conflicts in communicated information (available in plain text as attributes of QuASE knowledge base individuals) and to provide conflict resolution through terminology translation and issuing targeted explanations.

The implementation of the analysis support in QuASE applies machine learning techniques of Apache Mahout [5] to the set of QuASE knowledge base individuals: similarity search for implementing information reuse, regression analysis for predicting attribute values, and hybrid (partially supervised) learning for classification-based recommendation and decision support.

The main advantage of the QuASE approach is that after mapping a project repository, its data is immediately converted into knowledge available for understandability management and analysis. Moreover, as the mapping is flexible by means of using special-purpose DSL to express the environment, the knowledge and the mapping, this allows large amounts of existing data to be the subject of applying these techniques. Thus, we address one of the main challenges of Software Analytics "*How can we make data useful to a wide audience, not just to developers but to anyone involved in software?*" [1]. In fact, the QuASE system can be seen as a bridge which connects end users, the data in project repositories and the (extendable) set of machine learning and natural language processing techniques: these techniques become applicable to the repository data automatically after the particular repository and communication environment are described by means of the QuASE site DSL.

As a result of completion of the project, the theoretical foundations of the approach and the proof-of-concept implementation of the software system have been established in full; the site configurations corresponding to the environments of two consortium partners have been implemented; one of them involves populating the knowledge base from a project repository with over 9300 issues resulting in over 1.6 million OWL 2 axioms; the system demonstrated an acceptable performance in this case.

In this paper, we limit ourselves with the implementation aspects of the web-based QuASE tool, and, in particular, its understandability management module, the implementation specifics of the other components will be the target of separate publication.

The rest of the paper is structured as follows: we start from defining the architecture of the QuASE tool in Section 2 and outlining the implementation of its main tiers and the components comprising these tiers. Next two sections provide more detailed descriptions of the functional components of the tool: in the Section 3 we describe the implementation of the understandability management component. This section is followed by the description of the future work together with the conclusions.

2. QuASE Tool Architecture

The QuASE tool implements client-server three-tier architecture (Fig.2). In this architecture,

1. Data storage tier handles storing the data and knowledge into the internal QuASE database and the knowledge base, respectively;
2. Application logic tier handles the operations defined on the available knowledge and the administrative logic,
3. Web application tier consists of rich web application based on Angular JS library integrated with plugin-based extension of the particular issue management system (e.g. Jira plugin [3]).

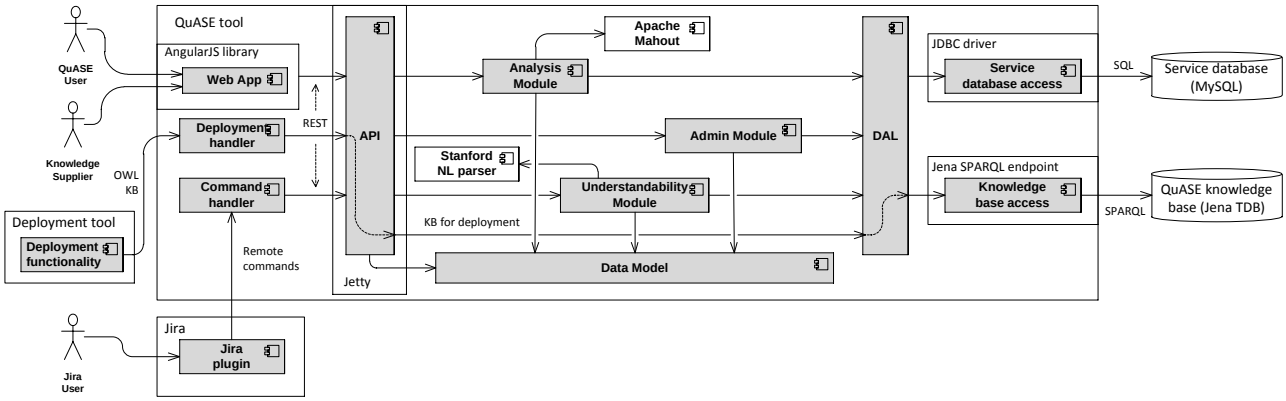

Figure 2. QuASE tool architecture

2.1. Data Storage Tier

QuASE deals with knowledge and should manage it using suitable data storage system. All the data available in project repositories and the data acquired from knowledge suppliers is transformed to knowledge; this knowledge has to be made available in the appropriate knowledge base. In addition, QuASE tool has its own internal data, used for administrative purposes. To address these needs, Data Storage Tier supports dealing with two categories of storage: the storage for QuASE knowledge base, and the storage for its internal data.

1. We use Apache Jena TDB (http://jena.apache.org/documentation/tdb) Java-based triple store as a storage for the QuASE knowledge base, its choice is justified in [11]. It provides a SPARQL query interface by means of Jena SPARQL endpoint.

2. The interface for internal data storage has been made database independent, but in the default configuration of the tool we use MySQL for this purpose.

2.2 Application Logic Tier

The application logic tier of the QuASE tool consists of the following four categories of components:

1. Data Access Layer (encapsulates details of accessing Data Storage Tier);
2. functionality modules (implementing support for understandability management and analysis scenarios together with administrative functions);
3. API Module (serves as an interface of the whole tier to be used by the Presentation Tier);
4. Data Model (implementing storage-independent data model shared among other modules).

In this section, we will not describe the functionality modules responsible for understandability management and analysis, their description will be provided in the separate sections.

Data Access Layer hides the details of accessing the Data Storage Tier. It introduces higher level of abstraction to manage and query Apache Jena TDB and the internal database including the functionality for building complex SPARQL queries.

Administrative Module handles all service functions, such as authorization, authentication, bookmarks functionality, which give user a possibility to create a bookmark for any combination of the function and the input data.

API Module is implemented based on embedded Jetty web-server. It provides REST interface, which supports authorization without using cookies (it makes possible to use any kind of HTTP client in Web Application Tier).

Data Model Module is shared among all other modules. It implements the data model, which is common for every application logic tier component (Fig 2). It implements a set of wrappers for individuals, data and object properties fetched from the knowledge base. A particular knowledge base individual is represented by MEOverview structure, which includes the IRI (Internationalized resource identifier), the name of the individual, and its unique numeric id. An individual is supplemented with a set of related data properties (represented by MEAttribute structure) and a set of object properties representing relations (these properties are represented by MERelation structure). All modules use the data model to read the knowledge from the QuASE knowledge base and to serialize and deserialize data before sending it to the client.

2.3. Web Application Tier

Web Application. The main component of Web Application Tier is a single-page JavaScript application, implemented with AngularJS library. It intera1cts with the embedded server through the REST API, provided by API Module. The internal structure of the web application corresponds to the structure of functional modules in Application Logic Tier. The layer implements primary security checks and input data validation.

This tier can be extended by any number of independent, third-party HTTP clients, such as plugins for Jira or any other Issue Management System, another web applications and so on. Two such clients are described below.

Deployment Handler. As the QuASE knowledge base is formed outside of the QuASE tool boundaries and often at different host (by the knowledge base builder utility), QuASE tool supports uploading this knowledge base (its OWL-based representation in RDF/XML format) by the deployment support tool run at the remote host (where the knowledge base is built). After uploading, the file containing this representation is handled by the Deployment Handler component implemented as a part of Web Application tier. This component receives the uploaded file and calls API to make this representation of the knowledge base written to the TDB triple store.

Command Handler. Current implementation of the Jira plugin functionality communicates with QuASE tool through the set of commands embedded in HTTP messages. Such messages are handled by the Command Handler component, which then calls the API function corresponding to the received message.

3. Implementing QuASE Understandability Management Module

In this section, we describe the implementation of context selection and understandability management scenarios provided by the QuASE understandability management module.

3.1. QuASE core knowledge structures

Prior to describing the implementation of the understandability management and analysis modules in detail, it is necessary to introduce the set of QuASE core knowledge structures [11, 12]. We will follow [11] in the description of these structures.

Fig. 3 depicts QuASE core knowledge structures as they are defined in the site ontology. We distinguish the following concepts [11]:

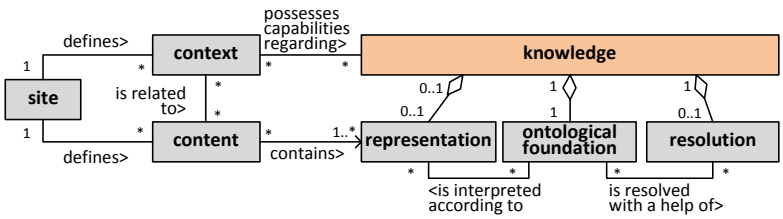

Figure 3. QuASE core site ontology structures (from [11])

1. *site:* owner of the given QuASE installation, e.g. a software provider.
2. *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. In addition, the set of context units can include the categories of such units (i.e. "IT company", "Business stakeholder") as it is possible to define the views on communicated information belonging to such categories (i.e. the view on quality belonging to IT companies).
3. *content:* units shaping communicated information originated from project repositories: they serve as containers for such information or organize such containers. Examples of content units are

issues and their sets, issue attribute values, and requirement specifications. Content units can be related to context units.

4. *knowledge:* units encapsulating quality and domain knowledge that is subject of communication and harmonization. Every QuASE knowledge unit is a triple (*o*,*v*,*r*) composed of the following components:
   a) *ontological foundation*: a reference to the conceptualization of the particular piece of quality or domain knowledge through ontological means: such conceptualizations form a modular ontology (*QuOntology*) providing a framework for translating between world views.
   b) *representation:* the representation of the knowledge unit in a format that could be perceived by the communicating parties (e.g. plain text). Representation units are contained in content units and can be connected to each other.
   c) *resolution means:* the means of resolving understandability conflicts related to the particular knowledge unit (such as explanations or external references).
5. Context units possess *capabilities* to deal with knowledge units; in particular, these capabilities could refer to the ability of understanding a given knowledge unit at hand (i.e. its representation); or of explaining a knowledge unit using resolution means.

3.2. Implementing context selection scenarios

The understandability management scenarios require the source and target contexts to be selected first. We treat the context as a possible path through the configuration of context units (connected with "Propagate capabilities" relations) ending with the particular context unit individual. The path starts from the individual belonging to one of the candidate *context roots*: the ontology classes which reside on top of the relation chain.

The context selection scenarios are implemented through the chain of *context selectors*: every such selector allows for selecting the context unit class through the class selector (drop-down box) and then selecting the individual belonging to that class through the individual selector (list box) (Fig.4). The context selectors are chained based on context relations connecting the corresponding classes in the original model: after selecting the individual on the upper level the lower-level selector is shown allowing for selecting the classes and elements of the new level. The set of classes available for the next-level selector corresponds to the existing connections (object relation individuals) between upper-level and lower-level individuals (only the classes with individuals existing in the knowledge base as targets for such connections are available for selection). The process of selection could stop at any level regardless of the availability of the connected bottom-level individuals.

An example of such selection could be as follows. On the top level, the root classes include Project and Person Category: after selecting one of these classes through the class selector, the set of corresponding individuals becomes available (e.g. the set of projects presented in the knowledge base). After selecting the particular individual (e.g. the particular project), a next-level selector is shown which allows for selecting the next-level context unit individual. In the model, we defined context relations connecting projects to Jira users, project versions, and components, as a result, the set of available classes for next-level selector could be a subset of these classes depending on availability of the connected individuals, e.g. for the projects connected to users and components, only these classes will be available for selection (Fig.3). If the class is connected recursively to itself (e.g. representing hierarchical relationship) the next-level selector could contain the set of individuals of the same class belonging to the lower level of hierarchy.

The implementation of context selection is supported in the site ontology as follows: the ontology builder looks for the all available chains of context relations in search of the context root classes, in this search, the availability of the individuals on all the levels of hierarchy is taken into consideration.

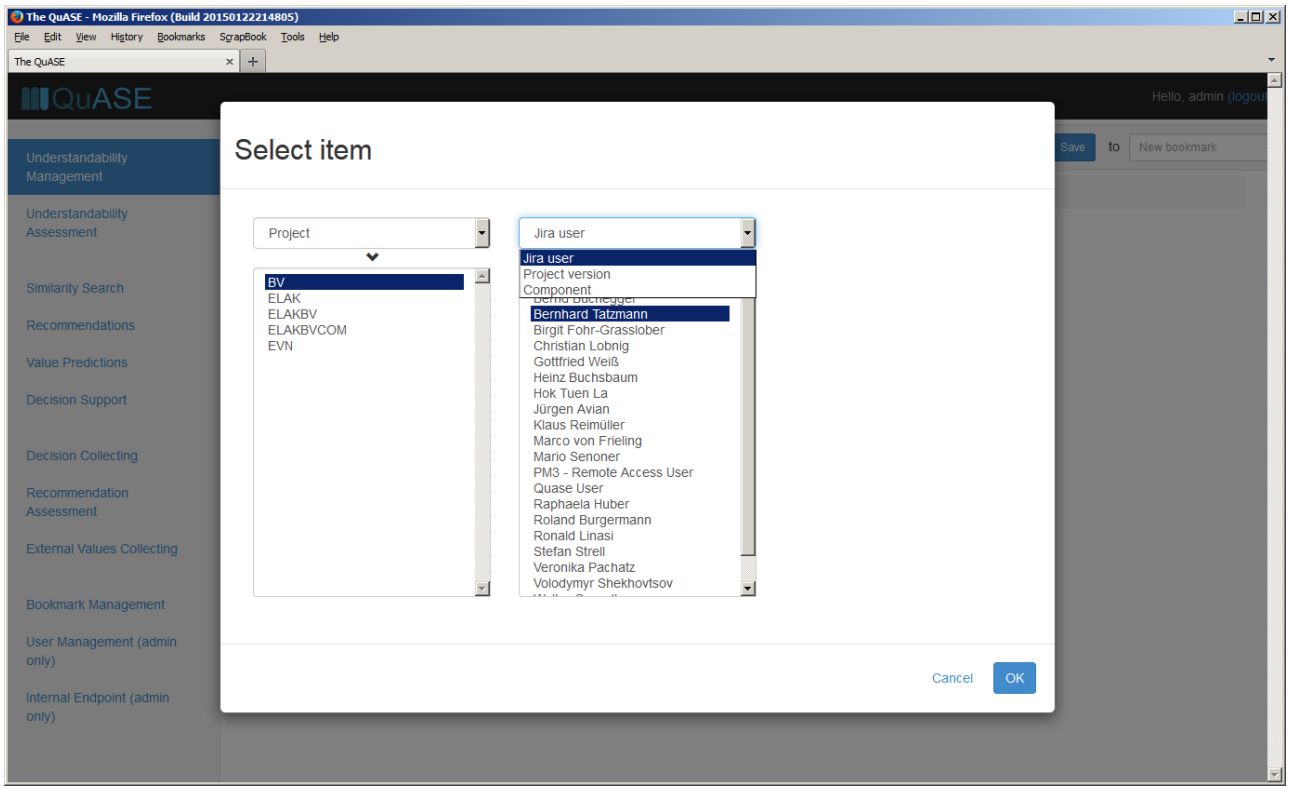

Figure 4. Context selection in the QuASE tool

The understandability management module on initializing the context selector, looks at the subclasses of Context Root class, after that, at any level of selection, two queries are executed by means of DAL commands: the first query selects the classes available at the particular level (with individuals connected to the upper-level selected individual), the second query looks for individuals belonging to the selected class.

3.2. Implementing understandability improvement scenarios

After the source and target contexts are selected, the understandability improvement scenario asks the user to select the document connected to the leaf element of the source context (e.g. to the particular user, or particular person category) with the context-content relation in the knowledge base. After such selection is made, the document is instantly shown to the user with the improvement algorithm immediately applied.

The understandability improvement algorithm works with a particular text (e.g. the value of the content unit attribute or the raw text fragment provided by the user), the source and target contexts.

1) Split the text into tokens by means of NLP parser (we use Stanford parser (http://nlp.stanford.edu/software/lex-parser.shtml) for this purpose), filter out the tokens that should not be considered (e.g. loan words);

2) For every token found on stage 1:

   a) Find the matching RU (representation unit) in the subset of the knowledge base consisting of the RUs connected to KUs (knowledge units) which are connected with capabilities to the *source* context unit (source KU set),
   b) If the matching RU is found, take the corresponding KU and check if that KU has the capability connected to the target context

      i) If such capability exists, the topic is understood to the target context directly, there is no understandability conflict for this token, mark the token as *non-conflict term* and continue with the next token;
      ii) If such capability does not exist, there is a possible understandability conflict. In this case, take the OU (ontological unit) connected to this KU and check if this OU is present in the set of OU connected to the subset of KU connected with capabilities to the target context (target KU set)

         (1) If the OU is present in this set, the term can be improved (translated and/or explained). In this case (a) obtain all KU corresponding to this set, those that have RU connected to them provide the alternatives for translation, those that have resolution units connected to them provide the alternatives for explanation, (b) mark the token as *translation or explanation inducting term* and continue with the next token;
         (2) If the OU is not found in the target KU set, the term cannot be improved**,** mark the token as *unresolved conflict term* and continue with the next token.

   c) If the matching RU is not found, continue with the next token.

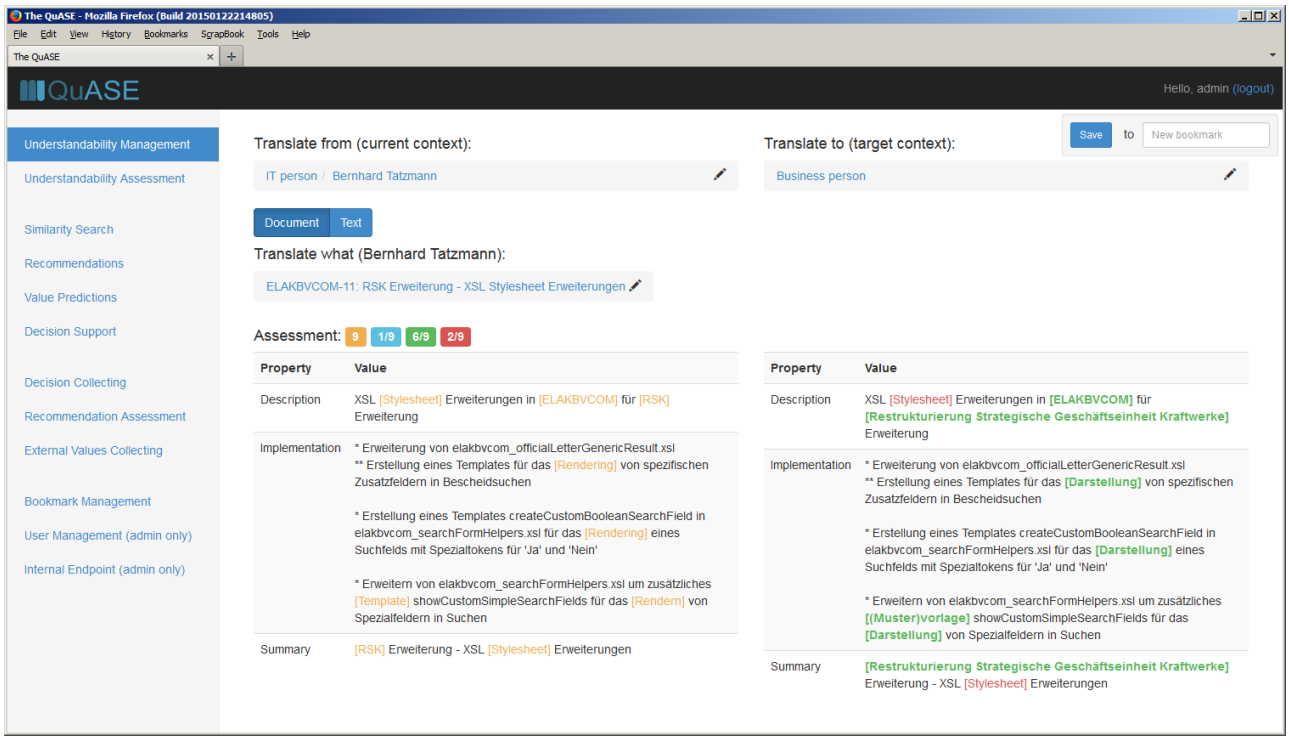

Figure 5. Translation-based understandability improvement in the QuASE tool

The implementation of this algorithm is invoked in the QuASE tool as follows:

1) The information consisting of the IRI of the document, and the hierarchy of current and target contexts is sent to the API as the list of ModelElementOverview structures;
2) Based on the document IRI, all attributes of this document that can be the subjects of understandability improvement are selected by means of SPARQL query;
3) The algorithm is applied repeatedly to the source texts defined as values of the raw text elements connected to the attributes defined on stage 2;
4) The information about the count of tokens belonging to every category is saved as the translation statistics; in addition, the *marked terms list* (*MTL*) is formed for every processed source text, a MTL element contains the label (source term), its explanation, translation, and the position in the source text;
5) After the translation is complete, the MTL is sent to the client web application together with the corresponding source text;
6) The client application renders the source text based on the obtained MTL (Fig.5) highlighting the problematic terms and their translations (translated terms are shown in green, unresolved conflict terms - in red); the values of the statistics related to the selected document are also shown after the "Assessment" label.

While performing the understandability assessment, the statistic information is collected for the set of selected documents (more than one document can be selected), normalized and send to the client web application (Fig.6).

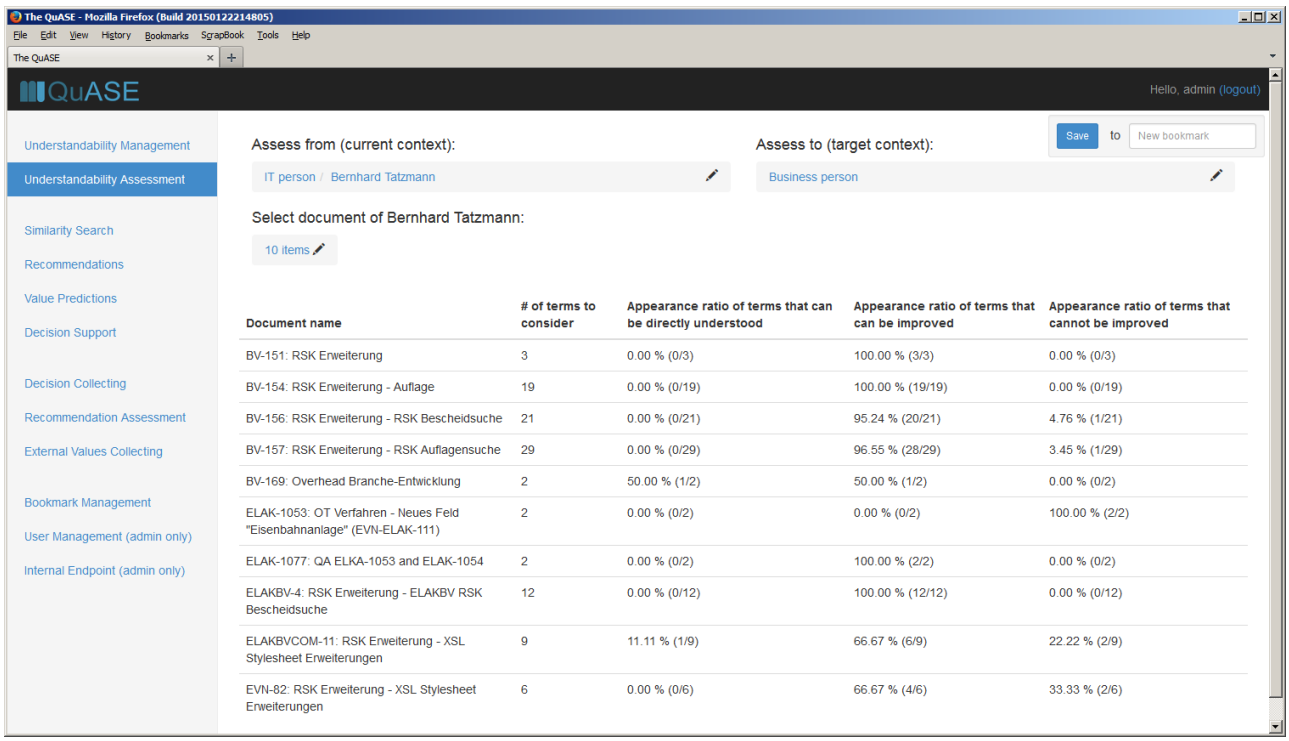

Figure 6. Understandability assessment in the QuASE tool

4. Conclusions

The QuASE project aims at the following five goals [9]:

Define the theoretical foundations, elaborate implementation procedures and a proof-of-concept tool support for

(G-1) acquiring and formalizing domain knowledge related to quality issues in the software process involving different parties, especially developers and business stakeholders;
(G-2) collecting the raw information about quality-related issues in a software process from the different involved parties; converting this information into operational knowledge; using available domain knowledge to ensure conversion correctness;
(G-3) using the collected knowledge for establishing a quality-related communication basis for the different parties involved in the software process, especially for developers and business stakeholders;
(G-4) supporting decision making in the software process, reuse of quality-related experience, and the prediction of the future quality-related behavior of the involved parties – based on the collected knowledge;

such that

(G-5) the tool support is easily to integrate into the existing development process of the software companies.

As a result of competing the project, all these original goals have been achieved in full. Here, we outline the completion of the goal G-3 related to the development of the understandability management module of the QuASE tool. The fulfillment of this goal is supported by:

1. The web-based QuASE tool supporting the scenarios of understandability assessment and improvement, in particular the scenarios of context selection, targeted terminology translation and issuing targeted explanations of the problematic terms; this tool has been completely implemented as a multi-tier Java-based solution.
2. Establishing these scenarios based on the knowledge structures allowing for collecting the necessary terminology from the knowledge suppliers and connecting the collected terminological knowledge to the particular elements of the communication environment by means of the terminological editor tool.

The obtained solution allows for describing the terminological knowledge, acquiring this knowledge into the QuASE knowledge base from the knowledge suppliers, connecting this knowledge to the knowledge about communication environment, and using this knowledge in understandability management scenarios supported by the interactive web-based tool.

5. Future work

The implemented solution, in accordance with the project proposal, is at the “proof-of-concept” stage, more work is necessary to improve its quality. Further cooperation between consortium partners aims at bringing the solution up to the level of released software product. The following activities are planned:

1. Implementing extended set of models and testing these models in corporate environments, such models include not only models based on Jira repositories, but also models for @enterprise, XEOX, OTRS and other systems;

2. Implementing incremental repository synchronization and other solutions aimed at improving QuASE scalability, performing extensive scalability testing of the QuASE solution in corporate environments on the large amounts of data;
3. Implementing advanced exception handling and other solutions aimed at improving QuASE reliability, performing extensive reliability testing of the QuASE solution;
4. Implementing automated and semi-automated procedures of gathering terminological knowledge from existing repositories;
5. In general, performing extensive acceptance testing for the provided solution.

In addition, the following research and development activities are planned based on the results of the project:

1. Providing the theoretical foundations and implementing practical support for extending understandability management procedures with more advanced natural language processing techniques;
2. Providing the theoretical foundations and implementing practical support for extending QuASE approach to additional representation formats for communicated information (besides natural language text) allowing dealing with such artifacts as e.g. generic conceptual models [2, 4] or process models [6, 7].

References

[1] GALL, H., MENZIES, T., WILLIAMS, L., ZIMMERMANN, T.: Software Development Analytics. Dagstuhl Reports 4, 2014, pp. 64–83

[2] GENERO, M., POELS, G., PIATTINI, M.: Defining and validating metrics for assessing the understandability of entity–relationship diagrams. Data & Knowledge Engineering 64, 2008, pp. 534-557

[3] KURUVILLA, J.: JIRA 5.x Development Cookbook. Packt Publishing, 2013

[4] MEHMOOD, K., CHERFI, S.S.: Data quality through model quality: a quality model for measuring and improving the understandability of conceptual models. in: MDSEDQS'09. ACM, 2009, pp. 29-32

[5] OWEN, S., ANIL, R., DUNNING, T., FRIEDMAN, E.: Mahout in Action. Manning, 2011

[6] RECKER, J.C.: Towards an understanding of process model quality. methodological considerations. in: Ljungberg, J., Andersson, M. (eds.): Proceedings 14th European Conference on Information Systems. 2006

[7] REIJERS, H.A., MENDLING, J.: A study into the factors that influence the understandability of business process models. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 41, 2011, pp. 449-462

[8] 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. Morgan-Kaufmann, 2014, pp. 41-73

[9] SHEKHOVTSOV, V., MAYR., H.C.: Towards intelligent handling of quality related issues in software development – a project report. in: Wuksch, D., Peischl, B., Kop, C. (eds.): Ausgewählte Beiträge zur Anwenderkonferenz für Softwarequalität Test und Innovation - ASQT'12. Österreichische Computer Gesellschaft, Wien, 2013, pp. 113-129

[10] 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.): ICCSA 2014, Proceedings, Part V. Lecture Notes in Computer Science, Vol. 8583. Springer, 2014, pp. 572-585

[11] SHEKHOVTSOV, V.A., MAYR, H.C., KOP, C.: Facilitating effective stakeholder communication in software development processes. in: Nurcan, S., Pimenidis, E. (eds.): CAISE'2014 Forum Post-proceedings. Springer, 2015, in print

[12] SHEKHOVTSOV, V.A., MAYR, H.C., LUBENSKYI, V.: QuASE: A Tool Supported Approach to Facilitating Quality-Related Communication in Software Development. in: da Silva, A.R., Silva, A.R., Brito, M.A., Machado, R.J. (eds.): QUATIC 2014. IEEE Press, 2014, pp. 162-165