Journal of
Information Technology & Software Engineering

Bajnaid et al., J Inform Tech Softw Eng 2015, 5:3
DOI: 10.4172/2165-7866.1000160

Research Article Open Access

Towards Ontology-based SQA Recommender for Agile Software Development

Nada O Bajnaid[1]*, Rachid Benlamri[2], Algirdas Pakstas[3] and Shahram Salekzamankhani[4]

[1]King Abdulaziz University, Saudi Arabia
[2]Lakehead University, Ontario, Canada
[3,4]London Metropolitan University, UK

Abstract

Agility is heavily dependent on tacit knowledge, skilled and motivated employees, and frequent communications. Although, the Agile Manifesto claims fast and light software development process while maintaining high quality, it is however not very clear how current agile practices and methods attain quality under time pressure and unstable requirements. In this paper, we present an ontological approach for process-driven Quality Assurance support for agile software development. Challenges related to the role of Quality Assurance in agile projects are addressed by developing a process-driven recommender that provides tailored resources to user's queries. The proposed ontological model embeds both conceptual and operational SQA knowledge about software processes and their requirements, including quality attributes, SQA measurements, SQA metrics and related SQA techniques and procedures.

Keywords: Agile software development; Context-awareness; E-learning; Ontology-based reasoning; Software quality assurance

Introduction

Software is a key element in daily human activities. Areas such as communications, transportation, health, finances, and education are highly dependent on software systems that range from simple to highly complex life critical systems. People are increasingly relying on software and demanding higher quality products than ever before. Therefore, producing high quality software based on Software Quality Assurance (SQA) techniques and standards becomes one of the most important objectives of software development and maintenance activities. Furthermore, as the complexity of producing software increases, the need for training highly qualified software engineers became critical. This is mainly due to: (a) the fast changing discipline; (b) inability to deal with large complex problems in a limited educational setup; and (c) the variety of methods, techniques, and technological tools used in this field [1,2]. Moreover, educators in this area have different backgrounds, programming language preferences, and usually use different jargons which lead to a variety of understanding and overlapping of meanings of the same software engineering terms or concepts. This may results in a lack of communication between the same team members and ambiguity in understanding requirements and defining system specifications. Therefore, the need for new support learning tools in the workplace is crucial.

Another challenge for software development is related to current rapid changes in technical and business environments with the need to deliver high quality software quickly which resulted in moving from traditional software development methods to agile development methods [3]. Although, the Agile Manifesto claims fast and light software development process while maintaining high quality, it is however, not very clear how current agile practices and methods attain quality under time pressure and unstable requirements. Developers need to know how to revise or tailor their agile methods in order to attain the required level of quality [4]. Knowledge quickly becomes outdated as a result of the shortened product life spans. In such rapidly changing environments, while companies struggle to keep their staff knowledgeable about the new technologies, training departments have to provide training and learning tools at the workplace that are efficient and adapted to current technological needs [5]. One way of achieving this goal is to embed quality tasks in every action and step of the software development process from requirement definition to post-delivery evolution [6]. In practice, this is a challenging task, due to the fact that developing software within schedule and budget has usually higher priority than achieving quality characteristics. In addition, achieving quality requires combining knowledge of different Software Engineering (SE) sub-disciplines, from software analyst to SQA experts [7].

In this paper an attempt is made to address the above-mentioned problems using an ontological approach in developing a process-driven recommender system that supports practitioners towards developing SQA compliant software. The focus of the paper is on the SQA ontology development that includes both domain (SQA concepts) and operational (SQA Processes) knowledge. Such ontology is used as the backbone to build a context-aware SQA recommender system that suggests useful resources, called in this paper Learning Objects (LOs) that deal with all SQA aspects of learner's current software development process [8]. The proposed process-driven recommender provides, just-in-time, and in a contextualized way, all necessary resources to enable software developers deal with SQA issues immediately after coding so that they can refactor while the code is still fresh in their mind. Such facility is a key requirement to address the role of SQA in agile software development.

The rest of the paper is organized as follows: the proposed SQA ontology model is presented in section 2 and an evaluation of the model is discussed in section 3. Section 4 presents experimental results of the developed system. Related work is given in section 5, and finally, section 6 summarizes the main findings of this study and suggests furthers research work.

Modelling the SQA Domain Knowledge

Standardization plays an important role in software engineering

***Corresponding author:** Nada O Bajnaid, King Abdulaziz University, Saudi Arabia, Tel: +966567747640; E-mail: nbajnaid@kau.edu.sa

Received September 25, 2015; **Accepted** October 12, 2015; **Published** October 26, 2015

Citation: Bajnaid NO, Benlamri R, Pakstas A, Salekzamankhani S (2015) Towards Ontology-based SQA Recommender for Agile Software Development. J Inform Tech Softw Eng 5: 160. doi:10.4172/2165-7866.1000160

by providing organizations with agreed upon and well-organized practices that assists users of software development methods in their work. There has been many progress made by different bodies to develop Software Engineering standards, resulting in the forming of the ISO/IEC Joint Technical Committee 1 (JTC1) workgroup in order to guarantee consistency and coherency among standards. Moreover, the IEEE Computer Society and the ISOJTC1-SC7 agreed to harmonize terminology among their standards. However, despite all these efforts, inconsistencies and terminology conflicts still appear between standards even within the same organization. Besides, there is still no single standard that embraces the whole Software Quality Assurance (SQA) knowledge. This paper presents two main contributions. The first contribution is an SQA ontology model that includes new conceptual knowledge based on the latest quality standards (ISO/IEC 20510 [9] and ISO/IEC 20523 [10]) and operational knowledge about SQA software development with special focus on agility. The second contribution however, is a recommender system that provides contextual SQA knowledge to support the software process being developed.

In this study we have used the ISO, all IEEE standards (ISO 20510, ISO 20523, IEEE 12207, IEEE 610.12, IEEE 00100, PMBOK 2008, CMMI v1.2) and SWEBOK to build a consistent SQA ontology that includes both domain and operational knowledge. Table 1 shows the main SQA-related classes that were extracted from the SWEBOK with their instances. The conceptual structure of the proposed SQA ontology is illustrated in Figure 1. The latter shows the various relationships used to define all SQA concepts, SQA-related software development processes, SQA measurements and SQA metrics. The main class in the domain ontology is SQA Concept class which is used to conceptualize and to represent the knowledge of the SQA domain. The figure also shows the major sub-classes of "SQA Concept". The arrows represent relationships (object properties) between domain classes (head of the arrow) and range classes (tail of the arrow). The "is-a" property linksan SQA concepts to its instances (individuals).

Figure 1 does not show all instances of the SQA measures due to space limitation and readability purpose. Also, applicable SQA sub-characteristics and measures are not limited to the ones listed in the ontology. The ontology is designed so that additional measures can be easily added for particular purposes to allow the ontology to evolve. In Table 2, we provide details of the various properties used in the SQA ontology. For each property, the table presents its domain, range, inverse property (if any) and cardinality. Quality sub-characteristics and their measures are crucial aspects of the SQA ontology. Measurement (quality sub-characteristics in ISO/IEC 20510) plays an important part in software development as it indicates the quality of the product being developed [11]. For any software quality product, measures associated with its attributes should collectively reflect likely user satisfaction with the use of the product and therefore the product entire quality [12] (Tables 1 and 2) (Figure 1).

According to Pressman's categorization of software metrics, quality metrics measure the extent to which customer requirements are fulfilled and indicate how closely software conforms to explicit (Functional Requirements - FR) and implicit (Non-FR) customer requirements. In this study, software measurements and measures are at the heart of the SQA ontology design. All aspects of SQA measurement and measures, as described in the ISO/IEC 25010 and ISO/IEC 25023 standards, are reflected in the proposed SQA ontology. In practice, these are instantly retrieved at the request of the software developer while engaging in a related software process. To the best of our knowledge, the proposed SQA ontology is the first to cover all SQA measurements and measures of the ISO/IEC quality standard. Due to space limitation, only measurements associated with "Maintainability" and "Reliability" quality attributes are chosen for illustration in Table 3.

SQA conceptualization is supplemented with an additional semantic layer that describes SQA operational knowledge, mainly SQA processes for both standard and agile software development. To support agility which relies on individual's tacit knowledge that is very much based on usual work practices and methods, some agile software development resources [13,14] were used to encode related SQA processes in the ontology as shown in Table 4 and highlighted in boldface font in Figure 1 (Tables 3 and 4).

Evaluating the SQA Ontology

High quality ontology can easily be reused and shared with confidence among applications and domains. Additionally, in case of re-use, the ontology may help to decrease maintenance costs [15]. To assess these two qualities, it is important to conduct an evaluation study that should also include assessing the usefulness of the ontology

SQA Ontology Class	List of Individuals
SQAProcess	Validation, verification, audit, review, inspection, joint review, technical review, management review, testing, quality assurance, SW design quality evaluation.
Quality Characteristic	Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Portability
Sub-characteristic	Functional Completeness, Functional Correctness, Maturity, Capacity, Fault Tolerance, Recoverability, Learnability, Operability, Installability, Interoperability, Appropriateness Recognizability, Time Behavior, Resource Utilization, Accessibility, User Error Protection, Availability, Functional Appropriateness, Adaptability, User Interface aesthetics, Analyzability, Modifiability, Testability, Non-repudiation, Replaceability, Coexistence, Confidentiality, Integrity, Accountability, Authenticity, Modularity, Reusability
Measure	Functional implementation coverage, Correctness, Computational accuracy, Functional appropriateness measure, Response time, Turnaround time, Throughput, CPU utilization, Memory utilization, I/O devices utilization, No. of online requests, No. of simultaneous access, Bandwidth of transmission system, Available coexistence, Connectivity with external system, Data exchangeability, Description completeness, Demonstration capability, Completeness of user documentation, Operational consistency, Message clarity, Customizing possibility, Input validity checking, Avoidance of incorrect operation, Appearance customizability of user interface, Physical accessibility, Fault removal, Test coverage, Mean time between failure MTBF, Service time ratio, Mean down time, Failure avoidance, Redundancy, Mean recovery time, Access controllability, Data encryption, Data corruption prevention, Utilization of digital signature, Access auditability, Authentication methods, Condensability, Execution of reusability, Audit trial capability, Diagnosis function sufficiency, Localization degree of correction impact, Modification complexity, Modification success rate, Functional completeness of embedded test functions, Autonomous testability, Test restartability, Hardware environment adaptability, System software environmental adaptability, Organizational environment adaptability, Installation time efficiency, Ease of Installation, User support function consistency, Functional inclusiveness, Continuous usage of data,
Deliverable	Operation report, problem report, audit strategy, design, fault removal report, requirement specification, QA plan, source code, review report, test cases, test report, test specification, user manual, user monitoring record, validation plan, verification plan.
Resource	Check list, complexity analysis, control flow analysis, meeting, prototyping, simulation, use cases, walk through.

Table 1: List of SQA Ontology Classes and their Instances.

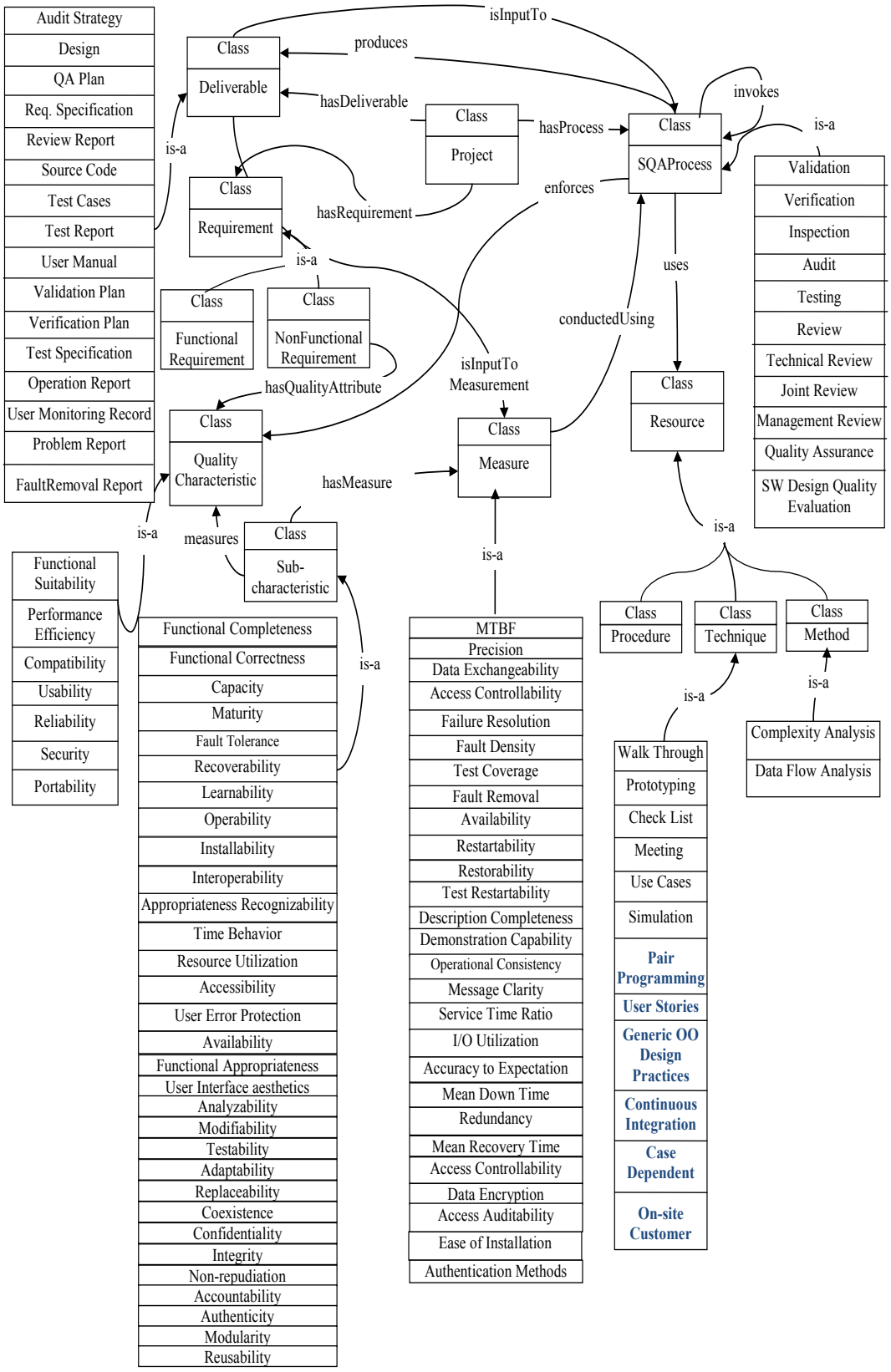

Figure 1: The SQA Conceptual Model.

Name	Domain	Range	Cardinality	Inverse Property
hasProcess	Project	SQAProcess	Multiple: a project may have more than one process	-
Enforces	SQAProcess	Quality Characteristic	Multiple: a process may have more than one attribute	enforced By
uses	SQAProcess	Resource	Multiple: a process may use more than one resource	is Used By
isInputTo	Deliverable	SQAProcess	Multiple: a process may have more than one deliverable as input	has Input
measures	Sub-characteristic	Quality Characteristic	Single: a quality sub-characteristic can be used to measure specific quality characteristic	is Measured By
invokes	SQAProcess	SQAProcess	Multiple: a process might invoke other processes	-
produces	SQAProcess	Deliverable	Multiple: A process might produce one or more products	is Produced By
hasMeasure	Quality Characteristic	Measure	Multiple: a quality sub-characteristic may have one or more measures	is Measurement Metric of
conductedUsing	Measure	SQAProcess	Multiple: a quality measure may be conducted using one or more process(es)	-

Table 2: SQA Ontology Properties.

Quality characteristic	Sub-characteristic	Measure	Input to measure	ISO/IEC 12207 Ref.
Maintainability	Analyzability	Audit trial capability	Problem report Operation report	Testing
		Diagnostic function support	Problem report Operation report	Testing
	Modifiability	Change access rate		
		Modification complexity	Problem report Operation report Maintenance report	Testing
		Localization degree		
	Testability	Test restartability		
Reliability	Maturity	Fault removal	Test report	Testing Validation Quality Assurance
		Mean Time Between Failure	Test report Operation report	Testing
		Test Coverage	Req. specification User manual Test report Operation report	Testing Validation Quality Assurance
	Fault tolerance	Failure avoidance	Test report Operation report	Testing Validation
	Recoverability	Mean recovery time	Test report Operation report	Testing Validation
		Restartability	Test report Operation report	Testing Validation

Table 3: Maintainability and Reliability Knowledge as in ISO/IEC 20510/25023.

Term	Ontology Concept	Related Ontology Concepts
User Stories	Technique	usedBy→ joint review and Verification
Pair Programming	Technique	usedBy→ Quality Assurance
Generic OO Design Practices	Technique	usedBy→ Quality Assurance
Continuous Integration	Technique	usedBy→ Validation and Verification
Case Dependent	Technique	usedBy→ Quality Assurance
On-site Customer	Technique	usedBy→ Joint Review
Iterative Incremental Development (IID)	Technique	usedBy→ Verification, Validation, Qualification Testing, and Joint Review

Table 4: Agile Terminology and SQA Processes.

for the purpose it was built for, and evaluating other attributes such conceptual coverage and clearness. A common approach is to evaluate the ontology according to a set of ontology design principles and criteria, such as those reported [16,17]. For example, it should be possible to extend the ontology to cover new needs and uses. Also, it is important to leave some representational choices (such as concept roles, relations, and constraints) open so that they can be made available at a late stage based on the actual needs of the problem solving or application. However, the most important three assessments according to Corcho [17] that should be conducted to evaluate an ontology are verification, validation and assessment. Verification refers to building the ontology correctly. In other word, it ensures that the ontology functions correctly in the real world. Validation refers to whether the ontology definitions really model the domain for which the ontology was created for. Ontology validation ensures that the correct ontology was built. The goal is to show that the world model is compliant with the formal model. Finally, assessment focuses on judging the ontology from users' points of view (human judgment).

Many attributes were used to develop the above-mentioned three

ontology assessments. The most used attributes are:

- **Completeness**: all knowledge that is expected to be in the ontology is either explicitly stated in it or can be inferred.
- **Consistency**: refers to whether a contradictory knowledge can be inferred from a valid input definition.
- **Conciseness**: ensures that the ontology is free from any unnecessary, useless, or redundant definitions.
- **Expandability**: refers to the ability to add new definitions without altering the already stated semantic.

Many ontology evaluation approaches have been adopted in the literature depending on the purpose of the evaluation and the type of the ontology being evaluated [18]. In survey ontology evaluation approaches are classified as follows [18]:

1. Those based on comparing the ontology to a "golden standard" which might be an ontology itself;
2. Those based on using the ontology in an application and evaluating the results (application-based ontology evaluation);
3. Those involving comparison with a source of data (e.g. a collection of documents) about the domain to be modeled by the ontology; and
4. Those where evaluation is done by humans who try, through a survey for instance, to assess how well the ontology meets a set of predefined user requirements and standards.

The first approach is not applicable in our case due to the lack of a "golden standard" Software Engineering ontology. However, the remaining three evaluation techniques have been used to assess the proposed SQA ontological model. An application-based ontology evaluation was conducted using the developed prototype recommender system as shown in section 4.2. The third approach was adopted during the ontology development stage where the evolving conceptual model (shown in Figure 1) was compared to the sources of knowledge as shown in section 4.1. The fourth approach was also used in this study by developing an ontology assessment questionnaire that was distributed among SE specialists. The results of the survey are presented in section 4.3.

Verifying the developed ontology

During implementation, the developed ontology was verified for consistency using the Protégé consistency checker tool which automatically checks the consistency and conciseness of the developed ontology. Only inconsistent classes will be displayed by the tool. Figure 2 shows the result generated by Protégé and the Racer Pro reasoning for the consistency checking [19] where no inconsistence classes are listed. Syntax checking is performed by Protégé OWL (Web Ontology Language) plug in which generates OWL statements during creation of the ontology using the Graphical User Interface. The plug in ensures that the generated OWL statements adhere to the rules of the OWL language.

In addition, the visualization tab (another Protégé plug in), enables a view of the graph representation of the ontology to ensure the ontology is consistent with the conceptual model (Figure 2).

Validating the ontology using the SQAES web application

Application-based (or task-based) evaluations offer a useful framework for measuring practical aspects of ontology deployment such as the responses provided by the system, the degree of explanation capability offered by the system, and the ease of use of the query component [20]. A proof of concept prototype consisting of an SQA recommender system has been designed and implemented [8]. To develop some scenarios, we have built an upper ontology, mainly for modeling learners' profile and learners' context. The upper ontology consists of three interrelated sub-ontologies, namely Developer (learner) Sub-ontology, Software Development Sub-ontology, and the SQA Domain Sub-ontology. Figure 3 shows the general structure of the upper ontology with the relationships among the sub-ontologies. The Developer Sub-ontology represents the developer's activity profile, which consists of already consumed learning resources. The developer's activity profile and related knowledge are organized into ontology concepts and relationships. This allows adapting and delivering LOs relevant to the software process currently in hand.

The SQA Domain Sub-ontology captures general concepts and properties about the SQA knowledge domain. The main class in this ontology is SQA Concept that is used to conceptualize and represent

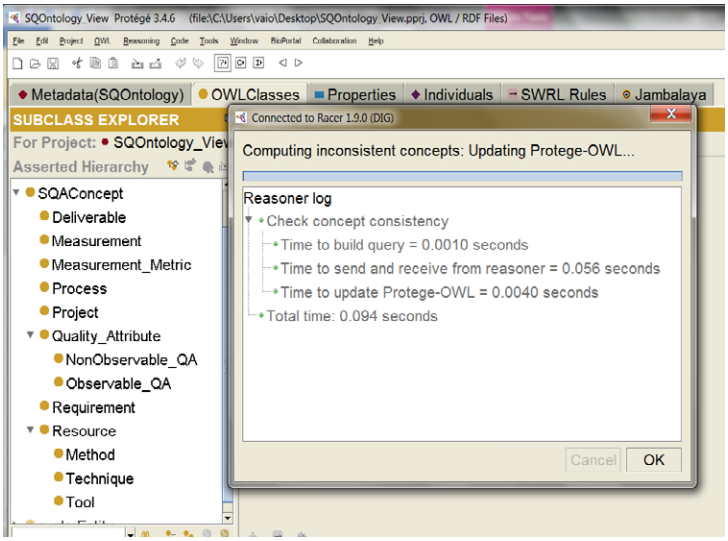

Figure 2: Protégé Consistency Checking Result for SQA Concepts.

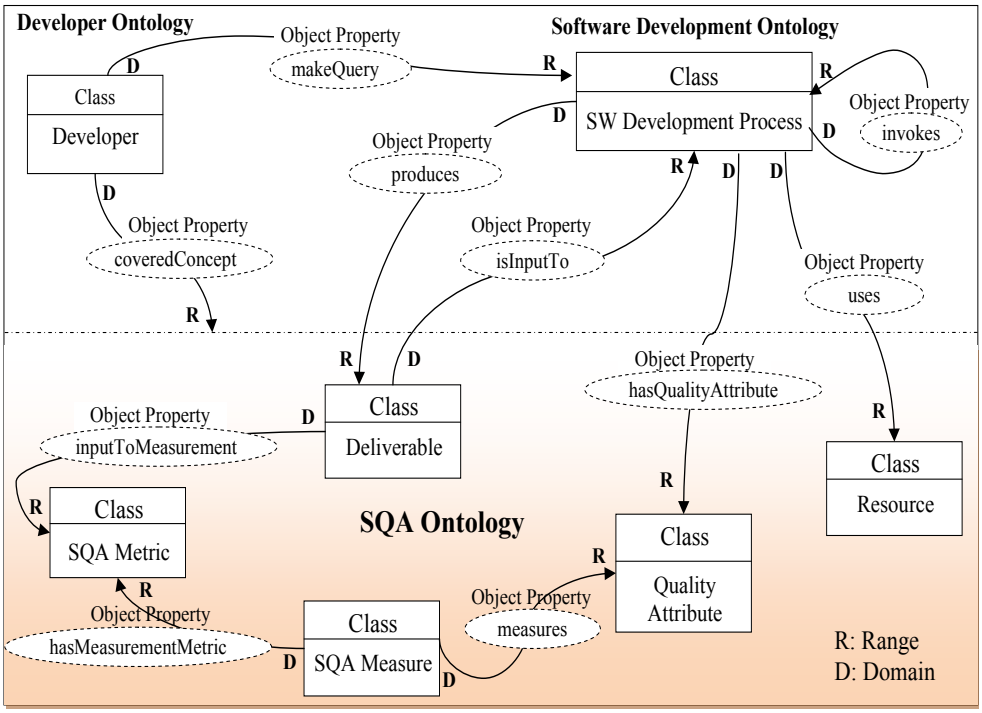

Figure 3: SQA Upper Ontology.

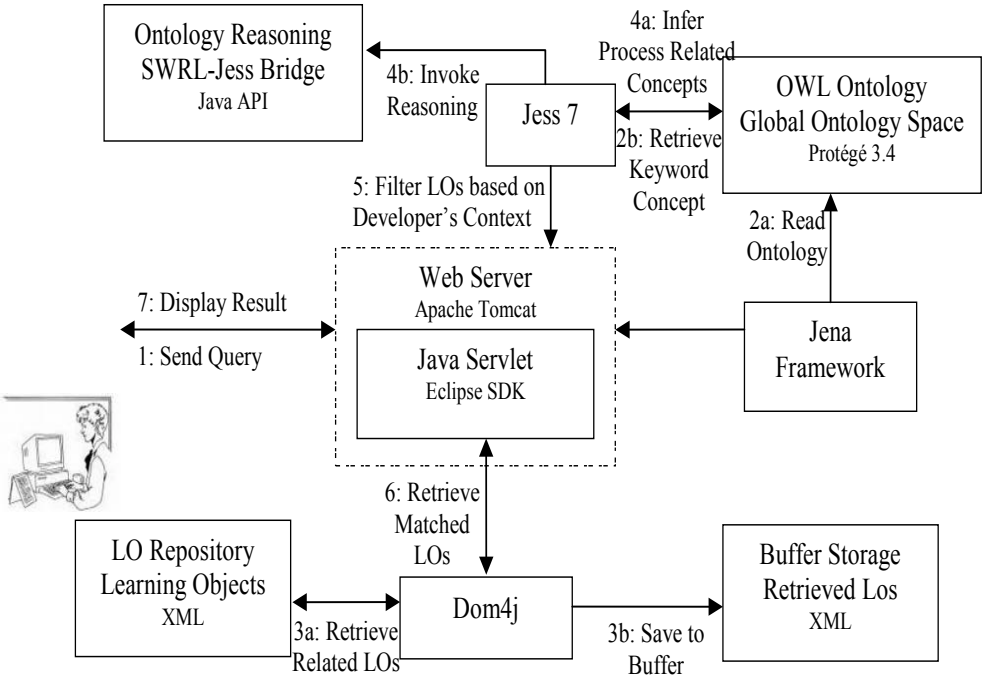

Figure 4: Software Components of the Recommender System.

all concepts of the software quality ontology and related operational knowledge. The property make Query associates process-related keywords entered by the developer to the most relevant concept in the Software Development Process Sub-ontology. The property is Mapped To links the concept class to the learning object class. The property is Mapped To is used to map learning objects' metadata to the SQA ontology concepts and thus allow sharing of resources. The property Consumed Learning Object tracks LOs previously consumed by a specific learner. The sequence of steps in a typical learning scenario is illustrated in Figure 4 (Figures 3 and 4).

As illustrated in Figure 4, the sequence of steps in a typical learning

scenario is given below:

1. The developer navigates (or queries for) an SQA concept.

2. The system retrieves the SQA concept(s) related to the developer's queried one.

3. Then, the system retrieves associated LOs from the LO repository using the term(s) extracted in step 2.

4. The system then infers other SQA related concepts using relationships such as Necessary Requirement, Optional Requirement, Used Resource, ensures QA, Produced Product, and Invoked Process.

5. LOs associated to new terms extracted in step 4 are retrieved from the LO repository.

6. The system will then check for previously covered concepts and LOs, which are then removed from the list of recommendations.

7. The suggested LOs are then provided to the developer for investigation, and then the same cycle can be repeated again.

In the developed prototype recommender system, ontology reasoning is performed to personalize software development services based on the developer's context. The system filters out the available LOs based on the developer's usage profile and guided by related ontology-based reasoning. The output is a set of SQA resources that are directly related to the developer selected query (i.e. SQA process being developed). This developer centric adaptation is enabled by integrating knowledge components from the three sub-onto logies. Ontological rules are applied to dynamically infer metadata that can be used to customize offered LOs.

Besides the OWL ontology reasoning rules (sub Class of, sub Property of, inverse of, etc.) which are necessary to navigate and search for ontology concepts and properties, the SQA knowledge base is extended with a set of user defined rules to allow inferring higher-level conceptual context from relevant low-level ones.

The prototype system aims at guiding software developers through the necessary SQA practices by providing resources that deal with SQA related aspects of the software process in hand and hence improves product quality in an agile software development environment. This is achieved by sensing the developer's current activity and suggesting relevant LOs (e.g. recommendations for good practices, example code, and graphical description of a related methodology/process) that deal with all SQA aspects related to the currently developed software process. The developer centric adaptation achieves its functionality in two steps. First, the reasoning unit of the proposed recommender system infers the core LOs that are directly related to the queried concept through the object property is Mapped To using the Core Learning Object rule illustrated by:

Developer (?D) ^ make Query (?D,?C) ^ is Mapped To (?C,?LO) ^ consumed Learning Object (?D,?LO) →core Learning Object (?C,?LO)

Related LOs are then inferred using different user defined SWRL (Semantic Web Rule Language) rules and depending on user's task. The output is a sequence of LOs that are generated as learning recommendations. Second, recommendations generated from the previous step are then semantically refined and adjusted according to the developer's profile where the system removes LOs that have already been consumed by the developer. The property consumed Learning Object links the learner (Software developer) to the learning objects that have already been consumed by the learner. The ontology model has been validated by developing many user scenarios using the prototype recommender system. Appendix A shows some of the SWRL rules that have been used to infer learning resources for all possible scenarios.

Ontology conciseness: The prototype recommender system provides the developer with a list of recommended LOs based on the initial query. However, this list may include many LOs that are out of context and therefore, might not be necessary for the user. To ensure conciseness, ontology axioms (i.e. a declaratively and rigorously represented knowledge which has to be accepted without proof) were added to prevent unnecessary knowledge. In ontology representation, axioms can be used to represent the meaning of concepts rigorously, and to answer questions on the capability of the built ontology using the ontology concepts. For example when the user queries the Validation concept, which is a process according to the SQA ontology, the system retrieves the core LOs associated with the Validation concept from the LO repository. Related concepts represent the list of recommended SQA concepts to be provided to the user for further investigation. However, this list may include some irrelevant LOs. In the example of Validation, by firing the Invokes rule, LOs associated with all software processes will be added to the list of recommendations. In theory (i.e. as per IEEE 12207 standard), only those processes that are associated with Review and Audit should have been added to the list and not all those listed in Figure 5.

To prevent such situation, recommendation refining is guaranteed by adding the so called "blocking axioms" to the ontology model. By referring back to our example related to Validation concept and according to ISO/IEC 9126 [21] standard, a Validation process produces Test Report and Validation Plan and requires Requirement Specification, Source Code, Test Report and User Manual as inputs. In addition, Validation has Efficiency and Functionality as quality attributes and uses Use-Cases, Iterative Incremental Development, Prototyping, Testing, Measurement, and Continuous Integration as resources. The above knowledge can be represented with the following axioms added to the Validation concept of the SQA ontology model:

∀ Produces only (Test Report or Validation Plan)

∀ invokes only (Review or Audit)

∀ ensures QA only (Efficiency or Functionality)

∀ uses only (Continuous Integration or Use case or Testing or Iterative Incremental Development or Prototyping or Measurement)

∀ has Input only (Requirement Specification or Source Code or Test Report or User manual)

In Table 5 we show few more axioms added to some of the concepts in the SQA ontology. The complete list of axioms cannot be presented due to space limitation (Figures 5 and 6) (Table 5).

Assessing the quality of the SQA ontology

Ontology assessment was conducted by judging the ontology content from SE specialists' point of view. The ontological conceptual model summarized in Figure 1 with a link to an assessment questionnaire has been sent to domain specialists inviting them to participate in the SQA ontology assessment process in order to verify its SQA domain coverage, structure, clarity, and extendibility. Collecting responses from domain experts was a challenging task due to the limited number of available experts in the SQA domain. It took more than seven months to get 16 responses out of a large number of invitations to participate in the online assessment. Although the sample is small, it is considered fairly acceptable to judge the developed SQA

Concept	Axioms
Efficiency	∀ is Ensured By only (Validation or Verification or SW_Design_Quality_Evaluation) ∀ measured By (Efficiency_Compliance or Resource_Utilization or Time_Behavior)
Failure Avoidance	∀ Conducted Using only (Joint_Review or Qualification_Testing or Validation or Verification) ∀ is Measurement Metric of only (Fault_Tolerance) ∀ has Measurement MetricInput only (Requirement_Specification or Review_Report or Test_Report)
Data Exchangeability	∀ Conducted Using only (Joint_Review or quality_Assurance or Validation) ∀ is Measurement Metric of only (Security) ∀ has Measurement MetricInput only (Requirement_Specification or Review_Report or Test_Report or Design or Operation_Report or Source_Code)
Test Coverage	∀ Conducted Using only (Qualification_Testing or quality_Assurance or Validation) ∀ is Measurement Metric of only (Maturity) ∀ has Measurement MetricInput only (Test_Report or Requirement_Specification or User_Manual)

Table 5: Some SQA Concepts with Related Axioms.

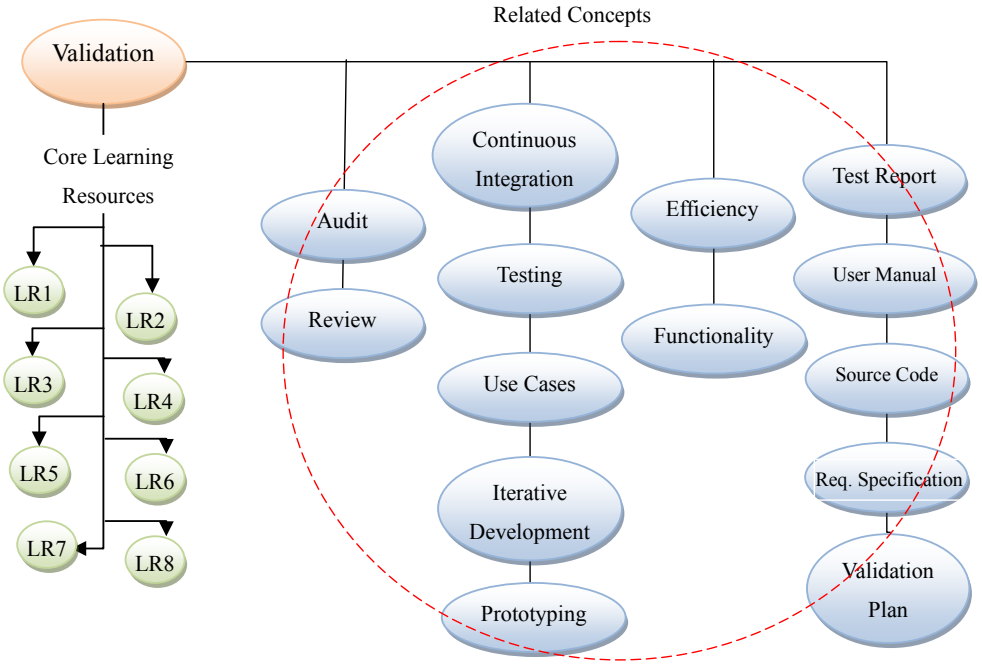

Figure 5: Process-Driven Recommendations for "Validation".

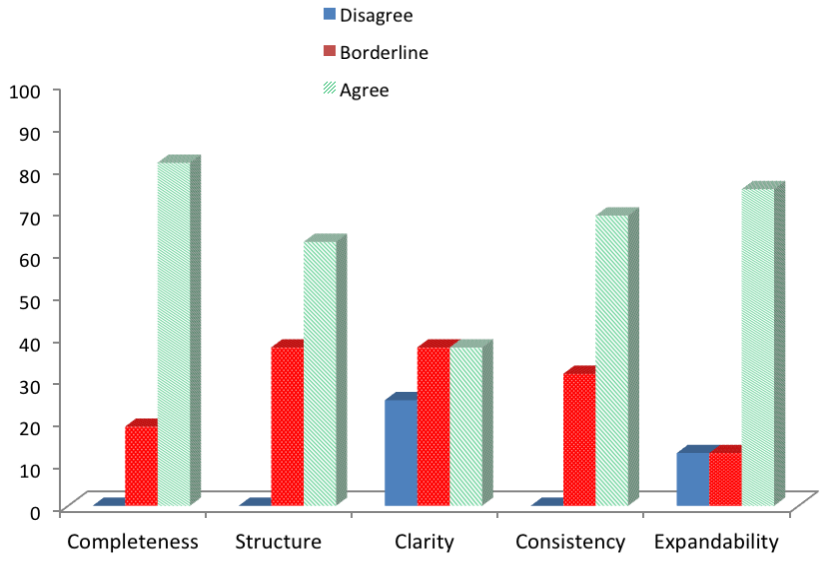

Figure 6: Participants' Assessments of the SQA Ontology.

domain ontology. The results of the survey are summarized in Figure 6 and an analysis is presented below, including experts' suggestions to enhance the SQA ontology.

Completeness: The majority of participants (81.3%) agreed that the ontology developed in this research covers the major concepts of the SQA domain. Few respondents however, think that it is missing "Testing" related concepts (unit testing, black and white box testing, system testing, etc.). Though, the current ontology is not heavily focused on testing techniques, it is worth investigating this aspect in future developments. Another suggestion was made to add concepts such as Software type, Software life cycle model, Architecture, Configuration management. However, we strongly believe that these are not SQA concepts. Nevertheless, these concepts can be added to the ontology if the latter is to be mapped to other SE areas or to an upper-level general purpose SE ontology.

Structure: A reasonable majority of the respondents (62.5%) agreed with the ontology taxonomy as is, while the remaining respondents did not have real disagreements. There were few remarks such as having Design comes after Review Report in the list of instances of the class Deliverable, which we consider semantically insignificant.

Clarity: This criterion obtained a borderline score (50%), just around the mean (3.13). However, we believe that this is a reasonably fair opinion due to the large number of overlapped and redundant SQA terms in available proposals and sources of SQA knowledge. It was noted that most reported disagreements were related to the confusion between Measurements and Metrics. A significant suggestion, which we have taken into consideration and have been incorporated in the ontology design, is to use the terms Quality Characteristic and Sub-characteristic instead of Quality Attribute and Measurements respectively. It was also suggested to replace the term Measurement Metric with the term Measure as per the latest quality standard ISO/IEC 25010. These recommendations were very useful in enhancing the ontology for clarity purpose.

Consistency: A reasonable majority of the responses (68.8) agreed that the developed ontology is consistent. Ontology consistency was verified using the Protégé consistency checker plug in.

Expandability: A good ontology is assumed to cover necessary concepts of the domain and structure them in a way that adding evolving concepts would not affect the existing structure. A satisfactory result was obtained for this criterion as the majority (75%) agreed on the expandability of the developed ontology. Some suggestions to include agile terminology with new quality measurements and metrics (as in ISO/IEC 25010) were taken into consideration and incorporated into the current SQA ontology design.

Experimental Results

In this section, we first present few working scenarios to show the most important features of the system. Then, we focus on another scenario that shows agile-based SQA software support features.

Working scenarios

In the first scenario we show the importance of using axioms in filtering relevant learning resources. The prototype recommender system provides the user with a recommendation list based on the initial query. The recommendations of the LOs suggested by the system include the core LOs of the queried concept and a few related topics based on the inferred SWRL rules. Figure 7 is a screen shot of the system when the user queries about the Validation process without the use of the ontology axioms.

In Figure 7 the system displays all SQA processes as invoked processes by the Validation process. Irrelevant knowledge have been prevented by adding axioms to the SQA ontology model. The results in Figure 8 show the system robustness when using ontology axioms as only contextually relevant resources are filtered. This validates the ontology conciseness and correctness when using ontology axioms (Figures 7 and 8).

In another scenario, the learner wants to know more about the Software Failure Avoidance concept. The screen shots shown in

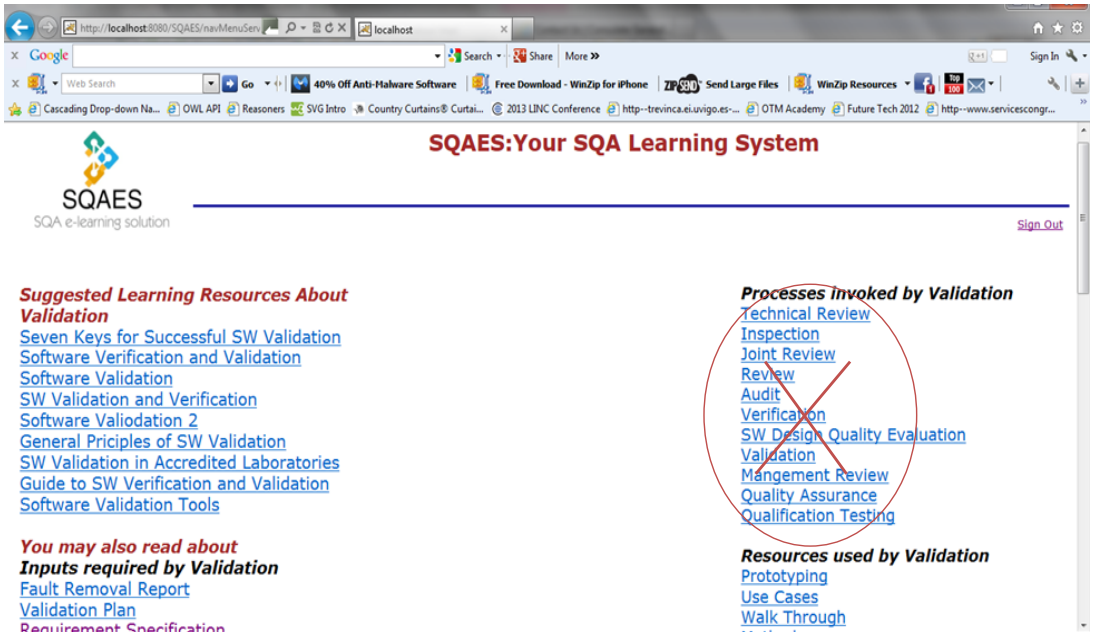

Figure 7: SQAES Response to the User's Query without Ontology Axioms.

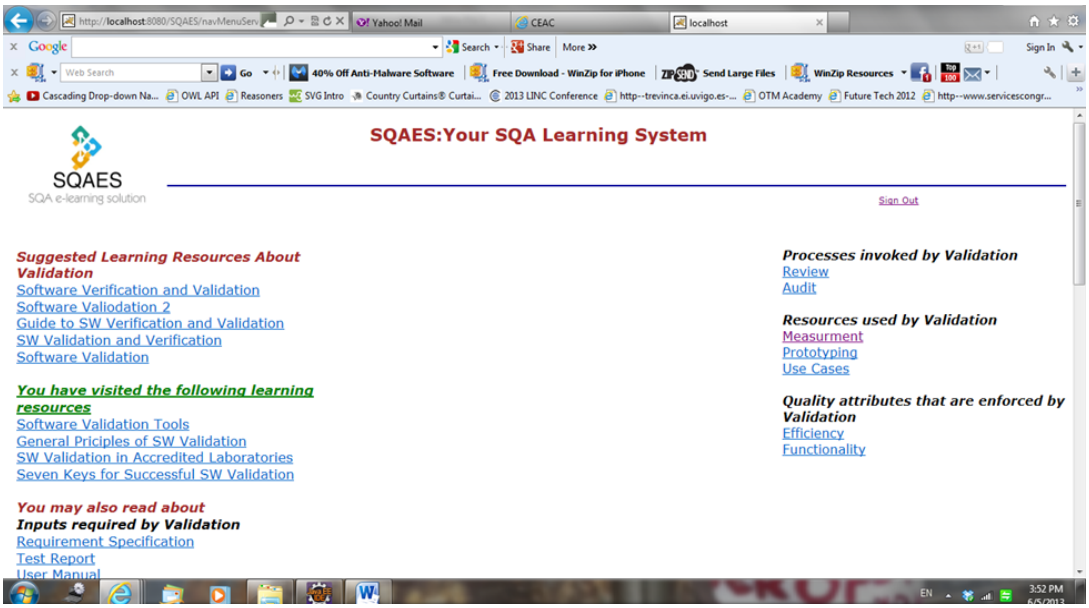

Figure 8: SQAES Response to the User's Query using Ontology Axioms.

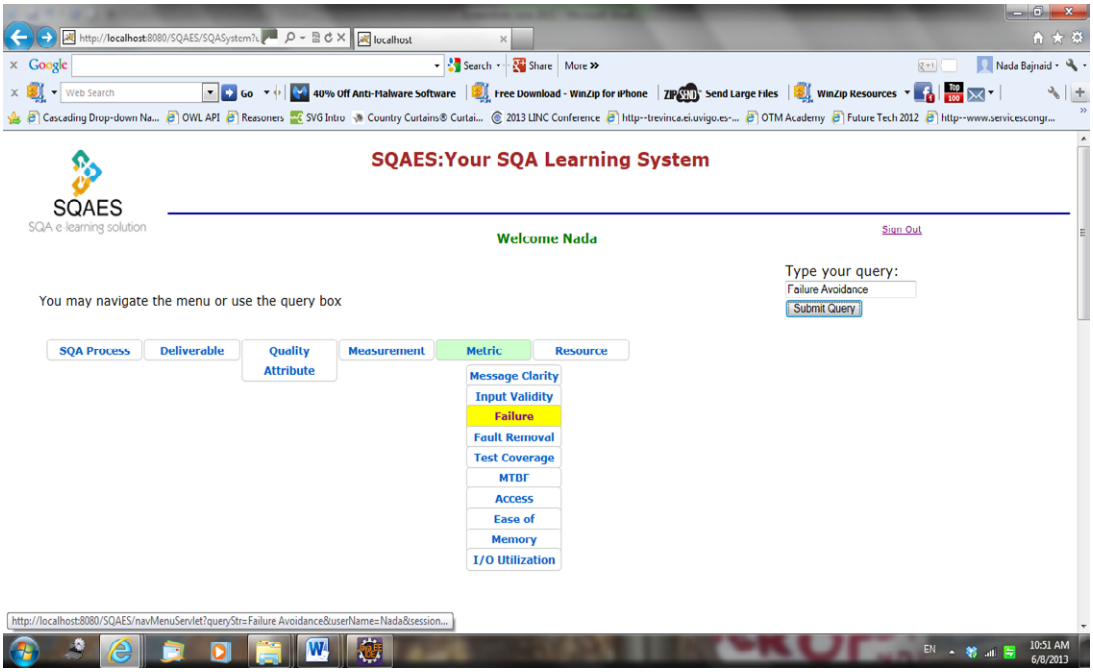

Figure 9a: The SQAES System.

Figures 9a-9d present the results from the system when the user queries "Software Failure Avoidance". The system initially provides all core LOs of the queried concept and a few related LOs generated when the reasoning system infers processes used to conduct the Failure Avoidance concept. These are related to software quality sub-characteristic that uses the Failure Avoidance measure and inputs to the Failure Avoidance measure. In this learning scenario (Figures 9a-9d), we show the system recommendations when the user viewed a LO related to the core knowledge of Failure Avoidance, and then, the learner further investigation about the SQA concept "Validation" where already consumed LOs are shown (Figures 9a-9d).

Agile software development scenario

Although agile methods produce software faster, they need to attain quality products. While quality software is the output of quality process, it is not clear how current agile practices and methods attain

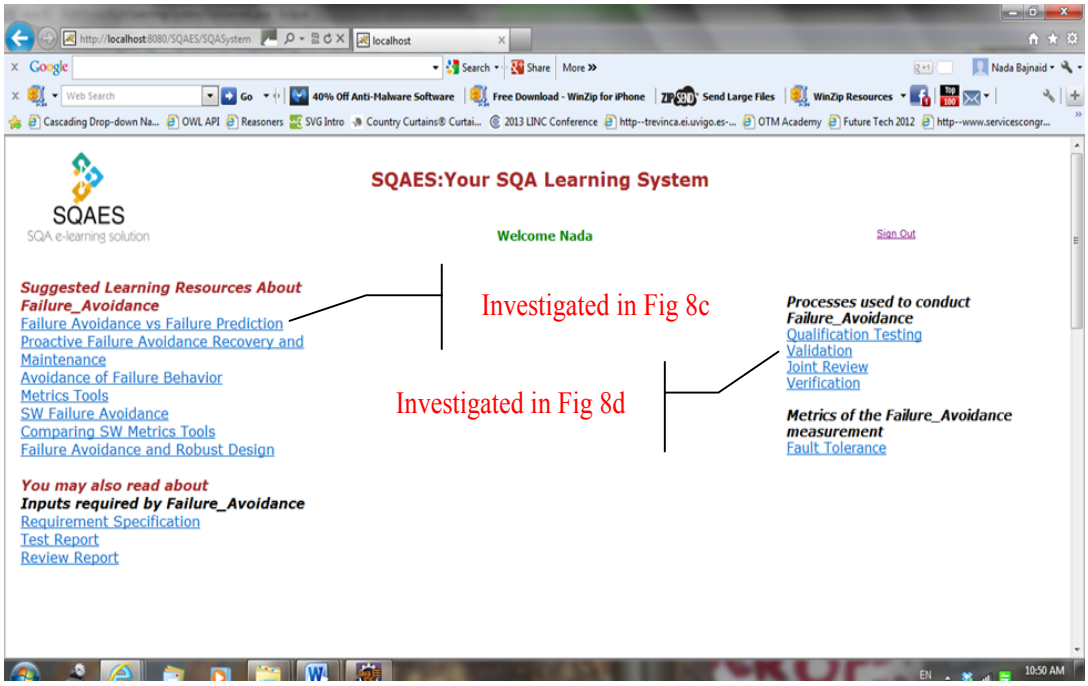

Figure 9b: The SQAES System.

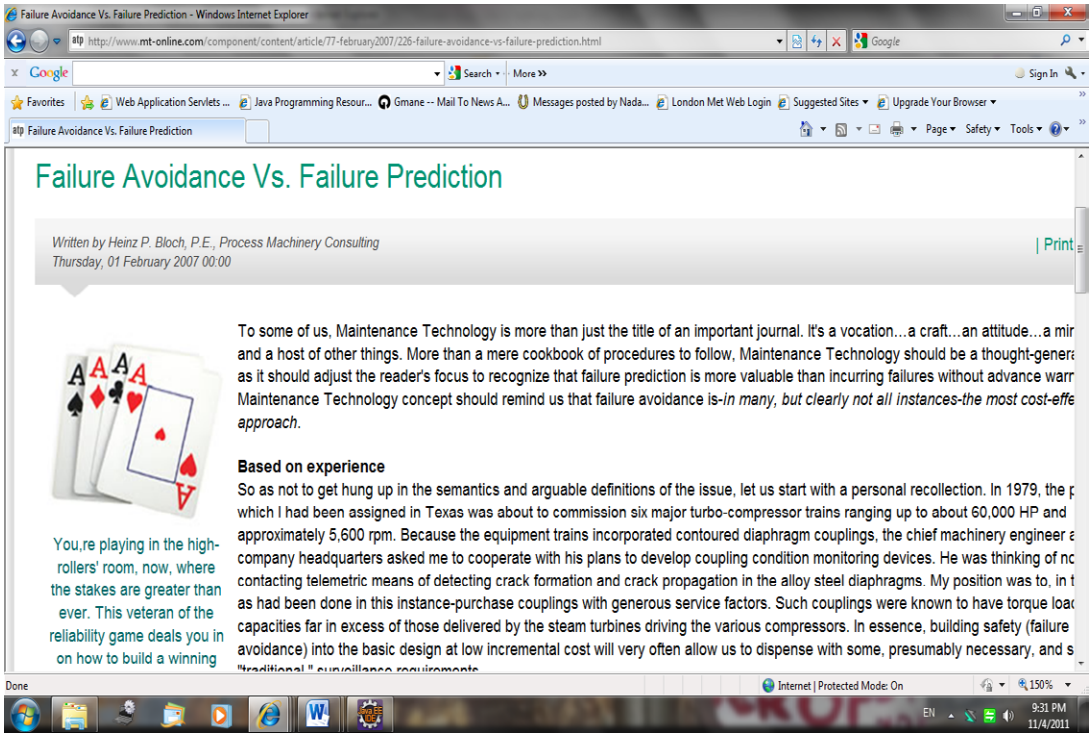

Figure 9c: The SQAES System.

quality under time pressure and in an unpredictable requirements environment. As an extension of the use of the prototype recommender system, the latter can be used to provide agile developers with, just-in-time and in a contextualized way, all necessary resources that deal with SQA related aspects of the software process at hand and hence improve quality in an agile software development environment. To support agility, which relies on individual's tacit knowledge that is very much based on usual work practices and methods, some agile software development resources [12,13] were used to encode related SQA concepts in the developed ontology. It should be noted that the

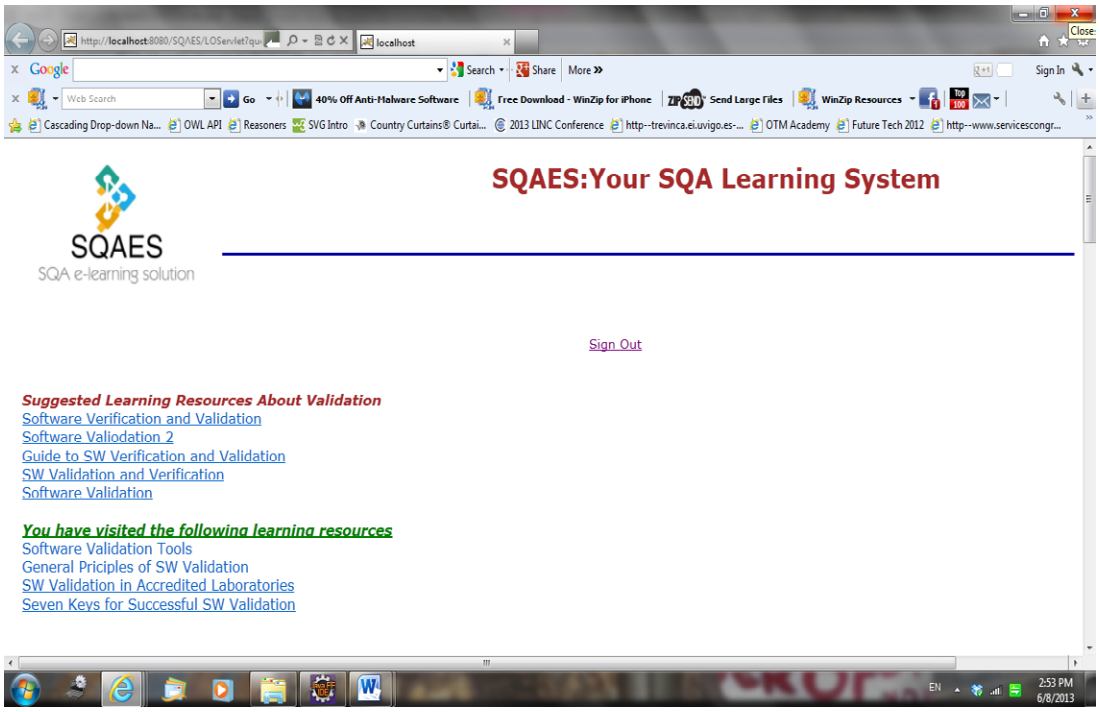

Figure 9d: The SQAES System.

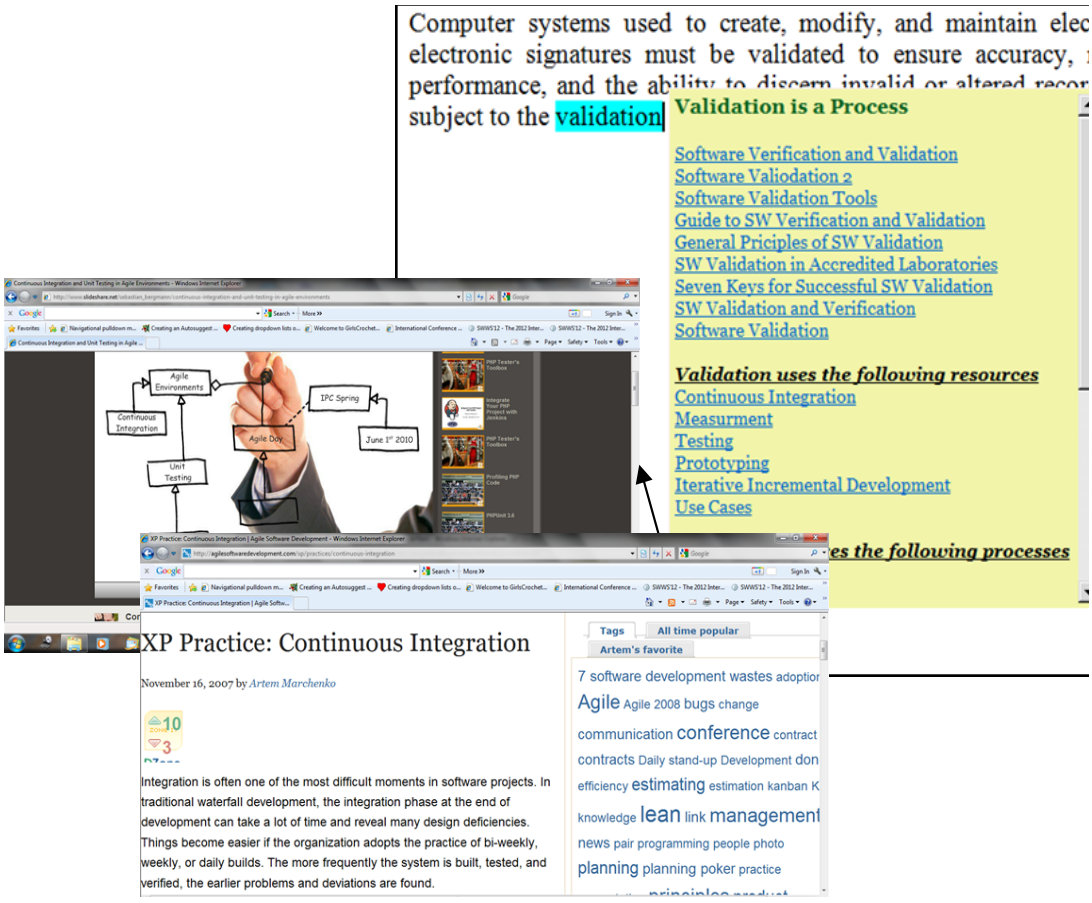

Figure 10: Combined view of the SQAES System for Agile SW Development.

inclusion of the agile terminology into the SQA ontology did not affect the concepts and relationships of the original ontology and thus confirms the expandability of the ontology.

To use the system in an agile development environment, the ontology is first used to annotate software development related keywords. Once a keyword is annotated, the system triggers a drop-down menu with all possible queries that can be generated from the ontology concept that is related to that keyword as shown in Figure 10. The example in Figure 10 shows a combined view with the drop-down menu displaying learning resources related to Validation and its SQA related concepts (invoked processes, produced deliverables, required inputs and used resources). In this case, the user would like to know more about Continuous Integration techniques that are used by the Validation process (Figure 10).

Related Work

In this section we survey recent work in the area of developing ontologies for SE knowledge representation with special focus on those developed for the SQA domain. Software Engineering Body Of Knowledge - SWEBOK-guide provides an international recognized consensus in software engineering terminology. SWEBOK has been used by many researchers to develop partial or sub-domain ontologies tailored to different purposes. However, comprehensive domain ontology in SE does not yet exist. Wille et al. [22] were the first to present a formal approach for designing ontology for SWEBOK. Their work was limited to modeling the taxonomy of software engineering as defined by SWEBOK knowledge areas. Also, their ontology is tightly designed to the SWEBOK naming space, which makes it difficult for mapping with externally defined concepts. Calero et al. [23] have developed a Software Measurement Ontology (SMO) to provide a coherent terminology among different software measurement proposals and standards. Unlike the ontology developed by Wille [22], the SMO ontology includes detailed knowledge about the measurement process, their attributes and results, while it does not link them to their SQA metrics and standards.

In the area of software testing, Barbosa et al. [24] have used the ISO/IEC 12207 [25] standard to develop Onto Test, an ontology based on a common well-defined vocabulary for software testing that can be useful to develop supporting tools and to increase interoperability among software testing tools. In our paper, we have borrowed few aspects of the Onto Test ontology, especially those related to testing processes, resources, and procedures [24]. In another related area, Kassab [26] proposed an ontological representation of the software Non-Functional Requirements (NFRs), their refinements, and their interdependencies. The ontology focuses mainly on the SQA measurement process, highlighting the mechanisms for measurable NFRs. For a complete classification of developed ontologies for software engineering [27].

Conclusion

In this paper we presented an ontological approach for developing a process-driven context-aware recommender to support agile software development. The SQA ontology developed in this study embeds both domain and operation knowledge about SQA processes and their requirements, including SQA quality attributes, metrics and SQA measurements based on the ISO/IEC 25010 and ISO/IEC 25023 standards. The system provides users with tailored SQA resources to support them developing the software process in hand in a timely manner. Context-awareness is achieved through a set of reasoning tools that take into account user's profile and learning history to recommend SQA resources needed for the task in hand. Also, reasoning axioms are dynamically added to the ontology for refining the list of recommended LOs. An evaluation study was performed to check the developed SQA ontology in terms of consistency, clarity, and completeness and the results were very promising. Future research is directed towards developing an excessive ontology assessment from experts' point of view. An ontology evaluation questionnaire has been developed and the ontology is currently being extended based on suggestions provided by SQA specialists.

Acknowledgment

The authors would like to thanks reviewers for their assessments, comments, and suggestions that helped improve the current work.

References

1. Boehm B, Chulani S, Verner J, Wong B (2009) Seventh workshop on Software Quality. International Conference on Software Engineering, ICSE-Companion, Vancouver, Canada 449-450.
2. Saiedian H, Weide B (2005) The New Context for Software Engineering Education and Training. The Journal of Systems and Software 74: 109-111.
3. Kalermo J, Rissanen J (2002) Agile Software Development in Theory and Practice. Software Business program, Masters Thesis.
4. Huo M, Verner J, Babar AM, Zhu L (2004) How Does Agility Ensure Quality? Proc. 2nd Workshop on Software Quality, Scotland.
5. Wang M, Jia H, Sugumaran V, Ran W, Liao J (2011) A Web-based Learning System for Software Test Professionals. IEEE Transactions on Education 54: 263-272.
6. Bourque P, Dupuis R (2004) Guide to the Software Engineering Body of Knowledge. SWEBOK, Computer Society Press.
7. Kusters RJ, Van Solingen R, Trienekens JJM (1999) Strategies for the Identification and Specification of Embedded Software Quality. Proc. STEP'99: Software Technology and Engineering Practice 33-39.
8. Bajnaid N, Benlamri R, Cogan B (2011) Context-Aware SQA E-learning System. Proc. of the Sixth International Conference on Digital Information Management ICDIM 2011, Melbourne, Australia.
9. (2011) Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and software quality models ISO/IEC 25010.
10. (2011) Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality ISO/IEC 25023.
11. Pressman RS (2005) Software Engineering: a Practitioner's Approach, (6thedn). McGraw-Hill Inc.
12. Bishop R, Lehman MM (1991) A View of Software Quality. IEEE Col. on Designing Quality into Software Based Systems. London.
13. Mnkandla E, Dwolatzky B (2006) Defining Agile Quality Assurance. Proc. ICSEA-International Conference on Software Engineering Advances, Tahiti.
14. Abrahamsson P, Salo O, Ronkainen J, Warsta J (2002) Agile Software Development Methods: Review and Analysis. (VTT Publication 478). Technical Research Centre of Finland, Espoo, Finland.
15. Vrandečić D (2009) Ontology Evaluation. Handbook on Ontologies. International Handbooks in Information Systems, (2ndedn), Springer, Heidelberg 293-313.
16. Gruber T (1995) Towards principles for the design of ontologies used for knowledge sharing. Int. Journal of Human-Computer Studies 43: 907-928.
17. Pérez G, Lopez F, Corcho O (2004) Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic Web. Springer-Verlag, New York, London.
18. Brank J, Grobelnik M, Mladenic D (2005) A survey of ontology evaluation techniques. Proc. of 8th Int. multi-conf. Information Society, Ljubljana, Slovenia.
19. Haarslev V, Hidde K, Möller R, Wessel M (2011) The RacerPro Knowledge Representation and Reasoning System. Semantic Web 1: 1-5.
20. Obrst L, Ceusters W, Mani I, Ray S, Smith B (2007) The Evaluation of Ontologies: Toward Improved Semantic Interoperability. Chapter in: Semantic

Web 139-158.

21. (2003) Software Engineering – Product Quality, Part1: Internal Metrics ISO/IEC 9126-9133.
22. Wille C, Dumke RR, Abran A, Desharnais E (2004) E-Learning Infrastructure for Software Engineering Education: Steps on Ontology Modeling for SWEBOK. Proceedings of the IASTED International Conference on Software Engineering 520-525.
23. Calero C, Ruiz F, Piattini M (2006) Ontolgies in Software Engineering and Software Technology. Springer.
24. Barbosa EF, Nakagawa EY, Maldonado JC (2006) Towards the establishment of an ontology of software testing. 18th Int. Conf. on Soft. Engineering and Knowledge Engineering (SEKE'06) , San Francisco.
25. (2008) System and Software Engineering – Software Life Cycle Processes. JTC 1 Information technology ISO/IEC 12207.
26. Kassab M (2009) Formal and quantitative approach to non-functional requirements modeling and assessment in software engineering. PhD thesis, Concordia University, Canadas.
27. Zhao Y, Dong J, Peng T (2009) Ontology Classification for Semantic-Web-Based Software Engineering. IEEE Transactions on Services Computing 2: 303-317.