London Metropolitan University

Faculty of Life Sciences and Computing

School of Computing

LONDON
metropolitan
university

An Ontological Approach to Model Software Quality Assurance Knowledge Domain

By Nada O. Bajnaid

A Thesis submittedto London Metroploitan University in partial fulfilment of the requirements for the degree of Doctor of Philosophy

December 2013

Abstract

Software Quality Assurance (SQA) becomes one of the most important objectives of software development and maintenance activities and as a result within an area of Software Engineering (SE) there are developed standards related to the SQA. Despite the effort made to improve consistency and coherency among standards, still there is no single standard embraces the whole SQA knowledge area. To contribute to this effort, this thesis presents a framework of an ontological model to describe and define both domain and operational knowledge of the SQA domain. Ontology development methodologies were reviewed and analysed in order to adopt the hybrid methodology used to develop the SQA ontology. The framework alos includes evaluation of the developed ontology. International standards (SWEBOK, IEEE, and ISO) were the main sources of the terminology and semantic relations of the developed SQA conceptual model. A formal ontology was implemented using the semantic web open standard OWL language. To avoid contradictory information, the developed ontology was verified for consistency using the Protégé consistency checker plugin. Different approaches have been used to evaluate the developed SQA ontology. An assessment questionnaire has been distributed among domain specialist to validate the quality of the developed ontology from experts' point of view. Evaluation of the result of using the ontology in an application or Application-Based ontology evaluation is used to validate the ontology consicness where an e-learning prototype is developed to provide learning recommendations to students (traditional learning scenario) or software developer (e-learning in the workplace). Ontology axioms were added to the developed SQA ontology to avoid unneccessarly and overwhelmed information. The e-learning prototype is developed using free open source software tools such as Apache tomcat as a server software; the Jena, a Semantic Web framework for ontology manipulation; the SWRL tab of Protégé to build the ontology reasoning rules; where the RacerPro reasoner is used for manipulating the ontology and the SWRL rules. Based on the results and findings of the ontology evaluation process, an enhanced version of the SQA ontology was developed based on the latest quality standards. The ultimate goal was to develop an ontology that faithfully models the SQA discipline as practiced in the software development life cycle.

To the Soul of my Father

Acknowledgment

This work would not have been possible without the guidance and support of my supervision team. This work is dedicated to the memory of Dr. Boris Cogan, his understanding and encouraging has provided a firm basis for this thesis. His wide knowledge and logic was of a great value for me. I'm sincerely grateful to my co-supervisor Prof. Rachid Benlamri at Lakehead University for providing guidance and encouragement that helped me refine my standpoint. I benfited from his wide experience in the field and learned a lot from his guidance in writing scientific papers. He was always there answering questions and clarifying difficulties. I'd also like to express my sincere appreciaition to Prof. Algirdas Pakstas and Dr. Shahram Salekzamankhani for their guidance, motivation and support towards the completeion of this dissertation and the challenging research behind it. I'am thankful for their aid and insight. I'm really proud to be supervised by them. Again, thanks Dr. Boris, Prof. Rachid, Prof. Algirdas and Dr. Shahram.

I also would like to thank my local supervisors at King Abdulaziz University: Dr. Hana Alnuaim for her support and guidance at the early stage of this work and Dr. Mai Fadel for her comments and support through this study. I'd like to thank Mrs. Tamara Matveeva for her insightful review, suggestions and constant encouragement.

I would like to specially thank my mother *Fatma Bajunaid* for her support and encouragement throughout the difficult times of my study.

Special "Thank You" goes to my husband *Ayman* and kids *Maan, Ziyad* and *Alya* for their understanding and unconditional love and support. It is impposible for me to express in words the full extent of my appreciation for their patience.

I would like to thank King Abdulaziz University, the Faculty of Computing and Information Technology for their support to undertake this work. "Thank you" to Dr. Ammar Amin and all members of the Joint Supervision Program for their continuous help and support.

Table of Contents

List of Figures

List of Tables

Chapter1: Introduction

This Chapter presents the motivation, objectives and scope of the research project. The Chapter then presents the structure of the remaining Chapters of the thesis.

1.1 Motivation

Many areas of human activities such as communication, transportation, health, finances, and education are highly dependent on software applications that range from simple to highly complex life critical systems. This requires software of high quality. According to the ISO 9000 (1992) standard, quality is defined as "the totality of characteristics of entity that bear on its ability to satisfy stated or implied needs". Software Quality is "the degree to which a system, component, or process meets customer or user needs and expectations" (IEEE 610.12, 1990).

Studies show that software companies can make more money through increased customer satisfaction and improved product quality (Boehm et al., 2009). Therefore, Software Quality Assurance (SQA) becomes one of the most important objectives of software development and maintenance activities and as a result within an area of Software Engineering (SE) there are developed standards related to the SQA.

Standardization plays an important role in software engineering by providing organizations with agreed and well organized practices that assist the users of software development methods in their work. Despite the efforts in research and international standardization, inconsistency and terminology conflicts appear between standards even within the same organization.

Although Software Quality Assurance (SQA) becomes one of the most important objectives of software development and maintenance activities, yet there is no consensus among the SQA community of most of the domain terminology and concepts.

A well-defined, complete and disciplined SQA process can be helpful to improve communication and collaboration among project participants and can serve as a standard when there is a disagreement.

Software quality is a rather complex concept; some authors have defined the entire discipline of SE as the production of quality software (Mankandla and Dwolatzky, 2006). Therefore, adopting software management and SQA standards, as well as training highly qualified software engineers became critical for developing high quality software.

Ontologies provide a common understanding and sharing of knowledge by using a general agreement on terminology among all interested people. In addition, ontologies can be very useful in improving keyword-based information retrieval techniques given that ontological representation of knowledge can provide better and more relevant answer to user queries in what is called concept-based information retrieval (Andreasen and Bulskov, 2007).

SE domain ontologies are very useful in developing high quality, reusable software by providing an unambiguous terminology that can be shared through the development processes. Ontologies also help in eliminating ambiguity, increasing consistency and integrating distinct user viewpoints (Uschold and Gruninger, 1996; Perez and Benjamins, 1999; Spyns, 2002; Zhao et al., 2009).

Using ontology to model the SE knowledge shortens the development time, improves productivity, decreases cost, and increases product quality. Ontologies provide better understanding of the required changes and the system to be maintained (Calero et al., 2006).

In addition, SE ontologies can be used as a mean for translation between different human languages when different users need to exchange information. Software developers with different backgrounds and viewpoints working on the same project can be supported by ontologies in the requirement specification process by offering a declarative specification of the system, its components and the relationship between the components (Calero et al., 2006).

There was an effort by different bodies to develop Software Engineering standards followed by the forming of the ISO/IEC Joint Technical Committee 1 (JTC1) workgroup in order to guarantee consistency and coherency among standards. The IEEE Computer Society and the ISOJTC1-SC7 agreed to harmonize terminology among their standards. Despite the efforts in research and international standardization, still there is no single standard which embraces the whole Software Quality Assurance (SQA) knowledge.

This work is motivated by the need for having a consistant terminology and agreed upon concepts among existing taxonomies of the SQA domain, where these taxonomies are mainly found in standard documents. The aim is to investigate available SQA knowledge resources, design and evaluate an ontological model of the SQA area that would facilitate concept-based retrieval of the SQA domain. For the development of the SQA ontology, 1) conceptual model of the SQA knowledge area should be defined then 2) machine-readable SQA formal ontology based on the conceptual model is to be implemented, and 3) finaly the developed SQA ontology is to be evaluated.

According to PMBOK (2008, p.4), "Generally recognized" means that the described knowledge and practices are applicable to most projects most of the time, and that there is widespread consensus about their value and usefulness. Carful analysis is done to identify knowledge that is up to the described level to ensure the resulting SQA ontology represents the SQA knowledge that is generally recognized.

1.2 Research Scope

The work presented in this research thesis combines theory and techniques from SE, SQA in particular and the ontologies. A macro view of the scope of this work is illustrated in Fig. 1.1.

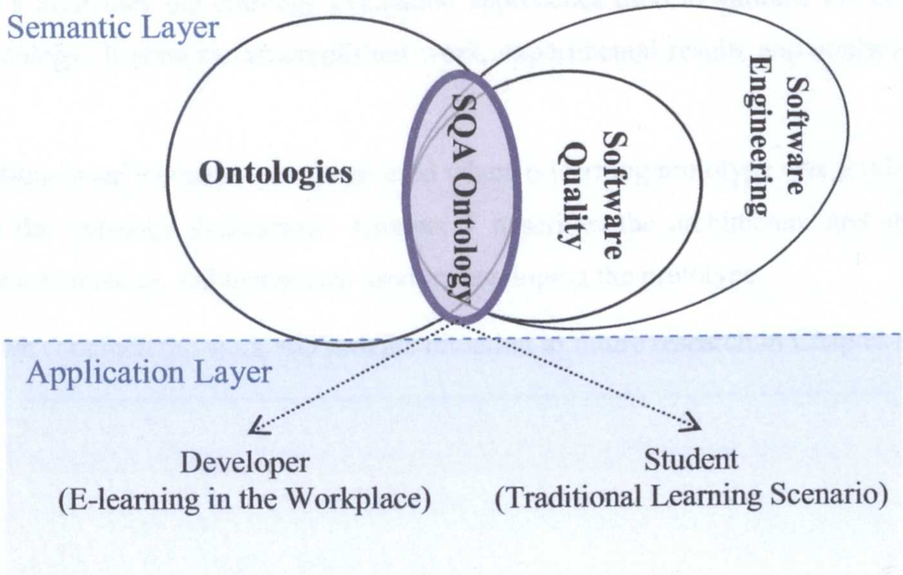


Figure 1.1: Research Scope

The primary source for development of the SQA ontology is the SWEBOK guide (2004), in addition to that, ISO and IEEE standards (ISO 9126, IEEE 12207, IEEE 610.12, IEEE 00100) and other documents such as PMBOK (2008).

1.4 Conclusion

This Chapter presented the motivation, aim and scope of the research. Ontology definition, components, development tools and languages are presented in chapter 2 with some examples of existing works in using ontologies in e-learning applications.

Chapter 3 introduces some basic background of two main relevant knowledge areas: SE and software quality in the context of SE. In addition, the Chapter presented exisiting ontologies for SE knowledge domain.

Chapter 4 presents review and analysis of ontology development methodologies and the approach used to model the SQA knowledge area.

The SQA conceptual model with detailed description of the vocabularies and relationships extraction process is presented in Chapter 5.

Chapter 6 addresses the ontology evaluation approaches used to validate the developed SQA ontology. It presents accomplished work, experimental results and analysis of the results.

Application-based evaluation has been used where e-learning prototype was implemted to validate the ontology deployment. Chapter 7 describes the architecture and the main software components and techniques used in developing the prototype.

Finally we conclude the work and provide direction to future research in Chapter 8.

Chapter2: Ontologies as Models of Knowledge

For the purpose of development of conceptual model of the SQA domain, it is necessary to define concise set of terms and their realationships for which is usually used ontologies. This chapter begins with defining ontology in different domains ranging from philosophy to computing and information technology domains. Then, it reviews some ontology aspects including (languages, components, tools, etc.). Finally, Section 2.7 represnts existing work that is related to using ontologies in education. Review of methodologies used to develop domain ontologies for the field of engineering and information technology are presented in Chapter 4.

2.1 Ontology Definition

"The Latin word ontologia *was created in 1606 by Lorhard and the first occurrence of "ontology" in English can be found in a work by Gideon Harvey of 1663". (Corazzon, 2013)*

Corazzon distinguishes two types of ontologies: pure philosphical ontology and applied scientific ontology. According to the Oxford English Dictionary (OED) the first appearance of the word "Ontology" was in 1721 in Nathan Bailey's dictionary which defined ontology as "an account of being in the abstract". The Webster's third new international dictionary defines ontology as: "a *science or study of being: specifically, a branch of metaphysics relating to the nature and relations of being; a particular system according to which problems of the nature of being are investigated"*. The term "Ontology" in philosophy is concerned with the study of being or the theory of the nature of existence (Gruber, 2008). Ontology in philosophy is also defined as the science of what is, of the structure of objects, events, processes and relationships among them (Smith, 2003).

Later the term ontology has been adopted by Artificial Intelligence (AI) researchers who established the idea of creating ontologies as computational models that enable automated reasoning (Gruber, 2008). According to (Neches et al., 1991) ontology *"defines the basic* *terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary".*

The most commonly used definition is: *"Ontology is an explicit specification of a conceptualization"* (Gruber, 1993). Conceptualization is an abstract, simplified view of concepts, objects and all other entities of domain knowledge and the relationships among them.

Based on Gruber's definition many other definitions were proposed. Borst added two requirements to the definition of Ontology:1) *formal* that means the ontology is machine processable, and 2) *sharable* which means having a consensus on the knowledge acquired by the community of experts. Borst's definition states that: *"Ontologies are formal specification of a shared conceptualization"* (Borst 1997 cited in Goōmez-Pérez et al. 2004 p.6).

A general definition (Uschold and Jasper 1999 cited in Goōmez-Pérez et al., 2004 p. 8) states that: *"ontology may take a variety of forms, but will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and indications of how concepts are interrelated which collectively impose a structure on the domain and constrain the possible interpretations of terms"* (Uschold and Jasper 1999 cited in Goōmez-Pérez et al., 2004 p. 8).

In relation to computer science, *"ontology refers to computer-based resources that represent agreed domain semantics. Ontology consists of relatively generic knowledge that can be reused by different type of applications or tasks"* (Spyns et al., 2002).

More definitions can be found in the literature, in particular in (Goōmez-Pérez et al., 2004) and (Calero et al., 2006).

Ontologies can be classified based on their contents (general ontology, domain ontology, and task ontology), the subject of the conceptualization, the level of dependence on a particular task, the richness of its internal structure, the purpose, degree of formality, and the benefits of the ontology (Goōmez-Pérez et al., 2004). Moreover, the ontology community differentiates between taxonomic ontologies and those that model the domain in a deeper way with more restricted semantics of the ontology (i.e. ontology axioms).

The ontology community also differentiates *lightweight* ontologies that include individual concepts, relationship between concepts, concepts taxonomies, and properties that describe the concepts and *heavyweight* ontologies that add constraints and axioms to the *lightweight* ontologies (Alyahya, 2006).

2.2 Upper-level and Domain Ontologies

Recall that the term ontology in philosophy characterises existence, this conceptualization of the world is called a *World* ontology that includes all existence concepts. Usually this ontology contains upper (top-level) ontologies and domain ontologies (Calero et al., 2006). Upper-level ontologies provide basic and very general concepts across domains and give general notations to which all terms in domain ontologies can be linked (Goómez-Pérez et al., 2004). Sometimes domain ontologies inherit from upper-level ones, but often domain ontologies are built then linked to upper-level ontologies. *Cyc*, an ontology of huge amount of common sense knowledge (Lenat and Guha, 1990), and the Standard Upper Ontology *SUO*, a large general-purpose formal ontology (Pease and Niles, 2002 cited in Goómez-Pérez et al., 2004), are examples of upper-level ontologies.

2.3 Ontology Components

Different knowledge representation languages exist for ontology implementation. Each of them provides different components that can be used in building ontologies. However, the following minimal set of components is shared among ontology representation languages (Calero et al., 2006):

[redacted]
set of objects. Classes in ontology are usually organized in hierarchal taxonomies through which inheritance mechanism can be applied. Some examples of classes are:
(cities, villages, etc.); (Ford, BMW, etc.), and [redacted]
Classes can contain individuals, other classes (sub-classes), or combination of both. Ontologies vary on whether they contain a universal class (a class that contains everything) or not. OWL ontologies have the [redacted]

Relations(Properties) represent a type of association between concepts of the domain. Ontologies usually contain ordered binary relations where the first argument represents the domain of the relations and the second argument represents the range. For example the binary relation `drives` has the concept `Person` as a domain and the concept `Car` as the range.

Binary relations are sometimes used to express concept attributes. Attributes are usually having their range as a datatype such as string, number, etc. in OWL relations are named *ObjectProperties* while attributes are named *DatatypeProperties*.

Instances are used to represents elements or individuals in an ontology. Instances (or individuals) are the basic, "ground level" components of an ontology. For instance `Tom` is an instance of the class `Person`.

Formal Axioms model sentences that are always true. Formal axioms are used to verify the consiciness of the ontology and to infer new knowledge (Gruber, 1993). An axiom in the traveling domain could be that it is not possible to travel from North America to Europe by train.

2.4 Ontology Representation Languages

There are many languages available for ontology representation. In 1990s, ontologies were built using mainly Artificial Intelligence (AI) modelling techniques. Such languages were based on:

- first order logic such as KIF (Genesereth and Fikes, 1992);
- frames combined with first order logic such as Cyc ontology (Lenat and Guha, 1990) and Ontolingua (Farquhar et al. 1997 cited in Goȏmez-Pérez et al., 2004);
- description logic such as LOOM (MacGregor, 1991).

Later, the boom of the internet led to the creation of ontology languages that can take advantages of the features of the Web known as *Web-based ontology languages* or *ontology markup languages* (Çalero et al., 2006). The most important examples of these markup languages are: RDF(S) (Lassila, and Swick, 1999), DAML+OIL (Horrocks, and

Van Harmelen, 2001), and OWL (Antoniou, and Van Harmelen, 2003). From all of them, RDF and OWL are the ones that are being actively supported now. Even though, RDF is developed long before the Web, the serialized version of RDF(s) in XML makes its way to the Web since the Web is based on XML. A detailed classification and review of ontology representation languages can be found in (Goōmez-Pérez et al., 2004).

Among the available ontology representation languages, the Web Ontology Language OWL has been selected in this research. Recently OWL is the ontology language that is preimerly recommended by the W3C. The OWL knowledge representation capabilities that allow defining objects as classes, properties as either *ObjectProperty* (relation) or *DatatypeProperty* (attribute), and individuals (instances) of different classes. Furthermore, OWL provides the possibility to reason about classes and individuals. It provides three sub-languages: OWL Lite, OWL DL, and OWL Full ordered with increased expressiveness.

2.5 Ontology Development Tools

Implementing ontologies directly in an ontology language, without supporting tool, makes the ontology building process complex and time consuming. To ease the task and help developers with some ontology development activities, the first ontology development environment was created in 1990s. Few years later, the number of ontology tools has greatly increased. Goōmez-Pérez and Corcho (2002) distinguish the following ontology tools: ontology development tools, ontology evaluation tools, ontology merge and alignment tools, ontology learning tools, ontology querying and inference engines, and ontology-based annotation tools. Overview and analysis of ontology learning tools and techniques can be found in (Calero et al., 2006; Fernández-López and Gômez-Pérez, 2002).

The first ontology development (or editing) tool was the **Ontolingua Server** (Farquhar et al. 1997 cited in Goōmez-Pérez et al., 2004) available as a World Wide Web service. It has been developed by Knowledge Systems Laboratories in Stanford to ease the development of the Ontolingua ontologies. Ontolingua supports distributed and collaborative editing of ontologies. Ontologies can be created from scratch or by extending existing ones.

In 1997, **WebOnto** (Domingue 1998 cited in Goõmez-Pérez et al., 2004) was released. The main advantage of WebOnto was its strong support for collaborative ontology edition, which allowed synchronous and asynchronous discussions about the ontologies being built by groups of users.

Another extensible tool is the **WebODE** (Arpirez et al. 2001 cited in Goõmez-Pérez et al., 2004). This tool is based on HTML forms and Java applets. The core of WebODE is its ontology access service, which is used by all the services and applications plugged into the server.

A free open source standalone application with an extensible architecture is the **Protégé** tool (Noy and McGuinness, 2001). The core of Protégé is its ontology editor, which can be extended with plug-ins that adds more functions to the environment.

Based on plug-in architecture, the free, flexible and extensible environment **OntoEdit** (Sure et al. 2002 cited in Goõmez-Pérez et al., 2004) was created. It provides user-friendly graphical interface and supports ontology development and maintenance. Its ontology editor is a stand-alone application that exports and imports ontologies in different formats (XML, FLogic, RDF(S), and DAML+OIL).There are two versions of OntoEdit: OntoEdit Free (with limited capabilities) and OntoEdit Professional, each with a different set of functions.

As the aim is to develop the SQA ontology from scratch, the tools and techniques that use existing ontologies to build new ones have been excluded and Protégé was selected due to the following reasons:

- Protégé is a free open source ontology editing tool with a variety of plug-ins and widgets to support the system functionality and capability.
- It has a user friendly graphical interface with easy to use menu-command tool.
- It is supported with a clear user guide and supports the import and export of ontology from/to different ontology representation languages (such as RDF and OWL).

- Protégé has the ability to verify the ontology and to check consistency for conformance with the language rules.
- Moreover, the "protégé-discussion" mailing list provides technical supports for the users which save time and efforts.

2.6 Ontology Reasoning Techniques

Ontologies provide formal meaning of concepts in a domain knowledge leading to shared and common understanding that improves communication between people and software agents. Using ontologies to represent domain knowledge allows not only the definition of concepts and their interrelationships but also inferring implicit relationships using reasoning techniques.

Reasoning is important to ensure the quality of an ontology, for example to check concepts consistency and derive implied relations (Baader et al., 2005). Ontology reasoning approaches supports inference through various kinds of logic: description logic, first order logic, temporal logic to name a few (Shehzad and Ngo, 2004). There are many ontology reasoning languages such as: the Description Logic Programs (DLP) (Baader et al., 2005), the Rule Markup Language (RuleML) (Horroks et al., 2004), and the Semantic Web Rule Language (SWRL) (Horroks et al., 2004; Parsia et al., 2005).

SWRL is a logic language based on a combination of OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. Table 2.1 shows a subset of the reasoning rules that support OWL semantics (Wang et al., 2004). SWRL uses the following syntax in writing user defined rules:

antecedent (body) ==> consequent (head)

where both antecedent and consequent are conjunctions of atoms $a_1 \wedge \ldots \wedge a_n$. The atoms can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom (x,y); where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals, or OWL data values (Horroks et al., 2004). Using this syntax, the following SWRL rule asserting that the composition of parents and brother properties implies the uncle property:

```
Rule-1:
person (?x) ^ hasParent (?x, ?y) ^ hasBrother (?y, ?z)
==> hasUncle (?x, ?z)
```

Where the concept person has been captured using the OWL class Person, the parent, brother and uncle relationships are expressed using the OWL object properties hasParent, hasBrother, and hasUncle respectively.

Table 2.1: Some OWL Ontology Reasoning Rules

TransitiveProperty	`(?P rdf:type owl:TransitiveProperty) ^ (?A ?P ?B) ^ (?B ?P ?C) ==> (?A ?P ?C)`
subClassOf	`(?a rdf:subClassOf ?b) ^ (?b rdf:subClassOf ?c) ==> (?a rdf:subClassOf ?c)`
subPropertyOf	`(?a rdf:subPropertyOf ?b) ^ (?b rdf:subPropertyOf ?c) ==> (?a rdf:subPropertyOf ?c)`
disjointWith	`(?C rdf:disjointWith ?D) ^ (?X rdf:type ?C) ^ (?Y rdf:type ?D) ==> (?X owl:differentFrom ?Y)`
inverseOf	`(?P owl:inverseOf ?Q) ^ (?X ?P ?Y) ==> (?Y ?Q ?X)`

DLP is the intersection set of strings of Horn logic and OWL while SWRL is the union of them. In DLP, the resulting language has very unusual looking description logic and overall inexpressive language (Parsia et al., 2005).

2.7 Ontologies in Education and e-Learning Applications

Ontology can be used as a tool for the representation of a specific domain knowledge which offers a consensual shared understanding of the domain knowledge to be exchanged and reused among people and organizations. In addition, the great expressiveness of the knowledge in the domain ontology supports the teaching and learning of the domain as it is machine-readable, and thus, can be used for e-learning purposes.

According to Stojanovic and colleagues (2001) ontologies in e-learning can be used to describe the content, the context, or the structure of the learning materials. For instance, when searching for a learning material, the content refers to what the learning material is about (the topic) and the context refers to the form in which this learning material is presented. However, structured ontologies breaks down learning materials into small bits of information (or chunk of knowledge) that can be connected to each other in order to build up a complete course. In this thesis, we adopted the approach based on the first and second category (i.e. the content and context ontologies)

Ontologies can support teachers in the course construction phase in the analysis and annotation of the learning objects where the course can be seen as a path over the ontology model of the course content. In addition, ontologies can support students to follow the suggested learning path or dynamically modify it according to their needs (Nicola et al., 2004).

Developing quality software requires well trained graduates with high SQA skills. Unfortunately, experience shows that most institutions are unable to graduate software engineers to meet manufactures expectations. This is mainly due to: (i) the fast changing discipline; (ii) inability to deal with large complex problems in a limited educational setup; and (iii) the variety of methods, techniques, and technological tools used in this field (Saiedian and Weide, 2005; Boehm et al., 2009).

One problem in teaching software engineering as a discipline is the use of textbooks, where the descipline is considered as a set of topics and subtopics that are studied sequentially. The discipline may be studied as separate modules/courses that may be not coordinated in terms of consistency and completeness. 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 term or concept. This often results in lack of communication between the same team members and ambiguity in understanding requirements and defining system specifications.

We, as educators, believe that we share part of the responsibilities for the gap between the software engineering graduates' knowledge and what is required in practice by software industry. Therefore, to improve the way of learning and teaching software quality, an ontological approach will be used to model SQA knowledge area (Bajnaid et al., 2010).

The following sub-sections present related works of: 1) developing domain ontologies for learning purposes and 2) using ontologies in context-aware personalised learning.

2.7.1 Domain Ontologies for Learning

In an attempt to create meaningful and effective learning strategies in teaching C programming, Sosnovsky and Gavrilova (2006) accumulate their experience in teaching several C-related programming courses to present an educational ontology that reflects their vision on what is important in studying C programming. In addition, they propose a stepwise algorithm to ontology development with a set of recommendations for ontology design. The proposed algorithm generalizes their experience in building different educational ontologies for e-learning in the field of AI and neurolinguistics.

Another educational ontology created by Jakkilinki and colleagues (2005) for their Multimedia Design and Planning Pyramid MUDPY model. They define MUDPY as a meta-design framework that facilitates successful creation of multimedia projects and supports teaching multimedia design and planning. MUDPY is being built to guide a novice learner through the multimedia design and planning process by answering queries on the MUDPY elements and their relationships. The MUDPY ontology support formalizing the processes of the multimedia design and planning which helps in teaching the same content for all learners. Dzcmydiene and Tankeleviciene (2008) proposed the framework for manual ontology development methodology used in building the "e-Learning Tools" domain ontology to enhance and improve the distance learning course "e-Learning Technologies".

For better software engineering education, a project-based collaborative learning environment was developed for learning software design patterns (DPs) (Jeremić et al., 2011). The environment integrates an existing learning management system, a software modelling tool, diverse collaboration tools and online repositories of DPs.

2.7.2 Ontology-Based Personalized Learning

Despite the development of many e-learning systems that enable flexible delivery of learning content, many research efforts are still needed to develop adaptable e-learning systems that take into account the changing context of the learner, or what is called context-aware e-learning systems. Context is defined as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object (Dey and Abowd, 2001).

For such systems, learning will become more integrated with work and will use more modular and just-in-time delivery system. To achieve these goals, new techniques are needed to model both explicit and tacit knowledge about the learner, including learner's goals, background, actual progress in the learning process, timing constraints if any, and current tasks and activities. Semantic Web represents a promising technology for developing such context-aware personalised e-learning system, and the use of ontology in particular supports expressive semantic representation of both explicit and tacit knowledge.

In the field of personalized learning, an approach for a dynamically generated personalized educational system powered by reasoning mechanisms has been proposed (Henze et al., 2004). The system uses three types of ontologies: a user ontology (describing user characteristics), an observation ontology (modelling different possible user interactions with the system), and a domain ontology (describing the concepts covered in the knowledge domain and the relationships among concepts). They show how rules can be enabled to reason over distributed information resources in order to dynamically derive semantic relations that can be used to adopt a learning path.

The Learning in Process project (LIP) (Schmidt and Winterhalter, 2004) aims to integrate e-learning and knowledge management technologies for a context aware learning object delivery. The system suggests personalized learning program based on a matching procedure between available learning material and user's current context.

Berri and Benlamri (2006) have developed context-aware e-learning system consists of a rule-based ontology and a search engine. Extracted knowledge from the source ontology is used to recommend a learning path by firing a set of rules based on the learner profile.

The **Learning Object Context Ontology** (LOCO) is an ontological framework that captures necessary information for personalization learning process (Jovanović et al., 2006).The central component of the framework is the LOCO-Cite ontology that serves an integration point of the other related ontologies (the user model ontology, the learning design ontology, and the content structure ontology). LOCO-Analyst, an educational tool built on top of the LOCO framework, aims to provide teachers with feedback on the learning process taking place in a web-based learning environment (Jovanović et al., 2007).

However, most of these personalized learning systems consider learner preferences and interests but fail to consider the difficulty level of the learning materials which may lead to the generation of poor quality learning paths.. In such cases learner could generate perceptive overload or fall into cognitive disorientation due to inappropriate curriculum sequencing during learning processes. In a way to solve this problem, Chen (2009) proposes a novel genetic-based personalized learning path generation schema to provide near-optimal learning path for individual learner. The schema based on an ontology-based concept map is able to simultaneously consider the course material difficulty and the relations between concepts of the prior and posterior knowledge between course materials in generation personalized learning paths.

In the same area, an infrastructure for context-aware e-learning services based on semantic knowledge representation, learning context processing and adaptive content recommendation has been developed (Yu et al., 2010).

Another similar context-aware e-learning system was developed by Das and colleagues (2010). This system uses standardized context parameters to build the context models, which in turn are used by a content management component to create learning resources that are dynamically composed into basic learning objects based on the leaner's context.

2.8 Conclusion

In this chapter it has been presented the most relevant definitions of the term **ontology**, other definitions can be found in Artificial Intelligence and Information Technologies literature. However, it can be noted that with all these definitions there is almost always a consensus of the usage of the term ontology among ontology developers and users. It can be concluded that ontologies are used to capture knowledge of a domain that can be shared and reused by group of people of software agents.

The chapter introduced examples of domain ontologies that have been developed for educational purposes (domain ontologies for learning, ontology-based personalized learning, and ontology-based context-aware learning). Eventhough these domain ontologies are developed for education, none of the ontologies is useful for this research as each of them represents a different domain and hence cannot be re-used in the development of the SQA ontology.

To our knowledge, there is no software quality ontology available for teaching and learning purposes. Having the opportunity to build operational ontology will provide a unique insight in teaching software quality in an e-learning environment.

The chapter then represents some related works that used ontological approaches for building context-aware personalized e-learning systems. Various context parameters are considered in existing e-learning system such as: learner personnel profile, expertise level, learning preferences, learning situation, network, device...etc. (Das et al., 2010). The e-learning prototype of this research work is presented in Chapter 7.

Chapter 3: Software Quality as Knowledge Domain

This chapter presents the background for the present research in several dimensions. The chapter starts by presenting a brief history of the SE domain and the SWEBOK guide. A brief history of the SQ as a SE area and quality issues in SWEBOK has been presented. References to related work in developing ontologies for the SE domain are made in this chapter.

3.1 Software Engineering as a Knowledge Domain

This section introduces *S*oftware *E*ngineering (SE) as a knowledge domain giving a brief history of the domain followed by a brief presentation of various versions of the *S*oft*W*are *E*ngineering *B*ody of *K*nowledge (SWEBOK) guides.

3.1.1 A Brief History of Software Engineering

In 1968, the North Atlantic Treaty Organization (NATO) Science Committee sponsored a conference to discuss all aspects of software including design, implementation, distribution, and services of software. The term "*S*oftware *E*ngineering" was known after the conference (Simons et al., 2003). There was a general agreement in the conference discussion that comparing to other engineering discipline, software engineering was in a very elementary stage of development. As an engineering branch, software engineering has some aspects (such as design life cycle) that are generally similar to other engineering branches while other aspects (such as problem analysis) were dissimilar due to the abstract nature of software.

The chosen term "Software Engineering" implies the need for software manufacturer to be based on theoretical foundations and practical disciplines as other engineering branches (Mahoney, 2004).

Glass (1997) divides the software engineering era into three periods:

1. The Pioneering Era (1955-1965)

Software people need to rewrite their programs to run in new computers coming up almost every year or two. New high-level languages like COBOL and FORTRAN were developed to translate old software to meet the needs of the new machines. No computer science principles had been taught yet.

2. The Stabilizing Era (1965-1980)

The IBM 360 came to sign the beginning of the stabilizing era and put the end of the era of emerging a new computer every year or two. Finally software people started writing new codes instead of rewriting the old ones. The beginning of the notion of time-sharing emerged. The value of software became huge as the software field stabilized. Structured programming appeared in the middle of this era. In addition, disciplines such as *A*rtificial *I*ntelligence (AI) came into existence. With the raising of *J*ob *C*ontrol *L*anguage (JCL), programmers needed to write the whole program in a new language to tell the operating system and computer what to do.

3. The Micro Era (1980- present)

Computer prices dropped dramatically. Every programmer had a desktop machine. The user-friendly GUI replaced the JCL. The most-used programming languages were 15 to 40 years old. There was an increasing need of more and better research in the software field.

Osterweil (2007) claims that, the history of software engineering clarifies the dual nature of today's software engineering. It has two activity types: the development of supportive tools and technologies to address the practical problems; and the search for better and deeper understandings as basis for those tools and technologies.

Due to the successful collaboration between software engineering practitioner community and research community, software development is now viewed as an industry that supplements economies of countries, nations, and even individuals. The continuous flow of problems from practicing software development opens new area of research and investigations.

Even though software engineering knowledge is more stable today, software engineering terms are inconsistent and may have different meanings in different contexts. The need for international agreements among standards, practitioners, researchers, organizations, and any related software engineering communities cannot be ignored. This was the main purpose to develop the SoftWare Engineering Body of Knowledge (SWEBOK) guide (2004).

3.1.2 The SWEBOK Guide

In the field of software engineering, communication is a key activity in developing software and the lack of communication leads to difficulties in identifying software requirements and specification. The ambiguity of the natural language of the participants leads to mistakes and non-productive efforts and limits the potential of reuse and sharing of knowledge.

Software engineering researchers face the challenge of knowledge integration that implies wasting time and efforts due to the lack of shared knowledge among members in the group project or with other groups or stakeholders.

In 1990, the planning for an international standard with a general view on the software engineering knowledge began. Five years later, the ISO/IEC 12207 was completed and published. This standard considered as a starting point to capture the software engineering body of knowledge.

In 1998, thirty years after the first use of the term "Software Engineering" in the 1968's NATO conference, the SWEBOK project was initiated with the following objectives (Mendes and Abran, 2004):

- To characterize the contents of the software engineering discipline;
- To provide topical access to the software engineering body of knowledge;
- To promote a consistent view of software engineering worldwide;
- To clarify the place – and set of boundaries – of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics; and

- To provide a foundation for curriculum development and individual certification material.

In December 2001, the first trial version of the SWEBOK guide was published. More than 500 reviewers from over 40 countries were involved in the project to develop the SWEBOK guide by the IEEE/ACM working group (Dupuis et al., 2003). The main purpose of this effort was to characterize the bounds of the software engineering discipline and provide access to the literature that describe the generally accepted knowledge of the discipline.

The trial version of the guide was released for general trial usage and applications. Review and comments of over 120 reviewers were used in developing the improved version in 2003, leading to the 2004 version. For example the software quality knowledge area was a mix of product quality and process quality; this was rewritten to consider product quality only (SWEBOK, 2004).

Transparency and consensus are principles developed by the project team to guide the project. Transparency means that all processes are documented and published so that participants are aware of project decisions and status. While consensus ensures that any statement is agreed by all significant parties (SWEBOK, 2004).

The resulting project is not the software engineering body of knowledge itself but a guide to the knowledge that hierarchically structured the field of software engineering into ten knowledge areas (KAs). For each subject, the reader is referred to book chapters or paper that describes the knowledge in that subject briefly. Each knowledge area is treated as a chapter in the guide plus a chapter gathers disciplines that are strongly related to the software engineering domain. According to SWEBOK, the software engineering is organized into the following ten knowledge areas:

1. **Software Requirements.** The guide defined a requirement as "a property that must be exhibited in order to solve some real-world problem".

2. **Software Design.** The guide adopted the IEEE definition of software design as "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of (that) process".

3. Software Construction. According to the guide, software construction refers to the detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing and debugging. The guide shows the links of software construction to other KAs strongly to software design and software testing.

4. Software Testing. Testing is the activity of identifying defects and problem in order to evaluate and improve a product quality.

5. Software Maintenance. Software need to be maintained to recover anomalies, be adapted to environmental changes or new user requirements.

6. Software Configuration Management. The term Configuration Management (CM) applied to all items to be controlled (software and hardware). It benefits project management, development, maintenance, assurance activities, and customers and end users. The guide clearly shows the close relation between SCM and the software quality KA.

7. Software Engineering Management. The guide adopted the IEEE definition of software engineering management as the application of management activities - planning, coordinating, measurement, monitoring, controlling, and reporting – to ensure that the development and maintenance of software is systematic, disciplined, and quantified. The guide uses the Project Management Body Of Knowledge (PMBOK) as a source of knowledge in the software engineering management KA.

8. Software Engineering Process. The guide deals with software engineering process KA at two levels. First: the technical and managerial activities within the software life cycle processes during software acquisition, development, maintenance, and retirement. While the second is the meta-level concerns with the definition, implementation, assessment, measurement, management, change, and improvement of the project life cycle processes.

9. Software Engineering Tools and Methods. This Chapter presents the methods and computer-based tools the assist the software life cycle processes.

10. Software Quality. What is software quality and what its importance as a software engineering knowledge area are questions answered by the software quality chapter in the

SWEBOK guide. The following section considers the software quality in the context of the software engineering domain in detail.

Each knowledge area includes a matrix to related references (book chapters, referred paper ...etc.) to each topic. The organization of the ten knowledge areas is not sequential. Links between KAs are not of input-output base and are given within text whenever needed (SWEBOK, 2004).

Public and private organizations can benefit from the SWEBOK guide in defining their education and training requirements, develop performance evaluation policies, classify jobs, and making public policy regarding professional licensing and guidelines. In addition, universities and learning institutes will benefit from the SWEBOK guide in defining certification rules, accreditation policies, curricula, and course contents (Dupuis et al., 2003).

3.2 Software Quality Knowledge Area in the Context of Software Engineering Domain

Over the past decades, changes in hardware have been absolutely remarkable and even changes in software and the ability to build large and complex software improved dramatically. The following section presents a brief history of the software quality knowledge area followed by a presentation of the software quality domain issues in the context of the SWEBOK guide.

3.2.1 A Brief History of Software Quality Issues

Practically achieving quality is a difficult process 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 related disciplines and experience of experts with different backgrounds (Kusters et al., 1999).

Software industry today pays more attention to the customer's requirement of better quality software. Industrial data shows that 50% of the project budget is spent on activities toward increasing quality such as testing. Industry leaders show that half of the testing costs can be reduced by applying practices and techniques to control quality throughout the software development life-cycle (Hilburn and Towhidnejad, 2002). A study by the Jet Propulsion Laboratory (JPL) shows that the ratio of defect detection and correction costs is 1:10:100:1000 through requirement: design: implementation: release. This means that fixing defects at the release phase costs 1000 times more than at the requirement phase (Rothman, 2002).

Over the past decades much effort has been put in software quality issues. Research papers and books on software quality have been published, and new standards were developed. The ISO-9000 (1992) series in particular become the most widely used by organizations to manage quality. Different standards interpret different definitions to software quality or quality in general. Let's consider for example the following ones:

- In the standard IEEE-610 (1990) Quality is defined as:

  1. the degree to which a system, component, or process meets specified requirement, and
  2. the degree to which a system, component, or process meets customer or user needs and expectations".

- In one of the popular textbooks (Pressman, 2005) Software Quality is defined as "conformance to explicitly stated functional and performance requirements, explicitly documented standards, and implicit characteristics that are expected of all professionally developed software"

However, to fully understand the different practitioners' view of quality, how to develop and achieve quality in software product, and how to measure and improve software quality, more research and studies are required in the field.

To achieve the required level of quality, organizations spend more efforts and resources through the development process. This includes technical development, process guidance and control, and some management activities to ensure what should be done, the way and time to do it and what should be not done.

Software product could not be highly qualified just by accident; quality processes lead to quality products. The effectiveness of the software development process can be measured by comparing the used processes by the widely accepted best practices (Thomas et al., 1996). It is difficult to say that a product quality is better than the quality of the process used to develop that product.

3.2.2 Software Quality Issues in SWEBOK 2004

Quality cannot be added to some steps of development or after completion. Quality implies in every action and step of the total development process from requirement definition to post-delivery evolution. For this reason, quality issues penetrate and cover all other knowledge areas of the SWEBOK guide.

Wille and colleagues (2003) analysed how the term "quality" and its related concepts are used in the context of the SWEBOK guide (2001 trial version). Table 3.1 summarizes their findings of the inclusion of the term "quality" into the other KAs in SWEBOK. The table illustrates how software quality issues penetrate into other software engineering knowledge areas.

Table 3.1: "Quality" in the ten SWEBOK KAs (adopted from Wille et al., 2003)

Knowledge Area	**The number of times "quality" is mentioned**
Software Requirement	60
Software Design	21
Software Construction	9
Software Testing	16
Software Maintenance	22
Software Configuration Management	19
Software Engineering Management	32
Software Engineering Process	16
Software Engineering Tools and Methods	4
Software Quality	187
Total	386

3.2.3 Bloom's Taxonomy for SQA Ontology Concepts

Bloom's taxonomy levels (Bloom, 1956) contains six levels of educational objectives: 1) knowledge (remembering, recalling); 2) comprehension (understanding); 3) application; 4) analysis; 5) synthesis (creating); and 6) evaluation.

The SWEBOK guide (2004) maps all knowledge areas to Bloom's taxonomy levels for one software engineer profile: a graduate with four years of experience.

Bourque and colleagues (2004) proposed Bloom's levels for two other profiles: new graduate and experienced engineer working a software engineering process group. They defined the levels for four knowledge areas of SWEBOK including software quality. In their approach no topic of the Software Engineering Education Knowledge, a body of knowledge developed for the purpose of designing software engineering curriculum in university, is assigned a rating higher than the application level for a new graduate profile. Their approach is applicable for undergraduate students too. In this research, Bourque's approach to identify the level of Bloom's learning objectives for the concepts of the developed SQA ontology has been followed. Table 3.2 presents software quality topics extracted by the author from SWEBOK and standards using Bloom's taxonomy and Bourque's classification.

Table 3.2: Bloom's Taxonomy for the some SQA Ontology Concepts

SQA topics according to SWEBOK and standards	Bloom's Taxonomy Level
Software engineering process quality	Application
Software engineering product quality	Application
Software Quality Assurance	Comprehension
Verification and Validation	Application
Management Review	Comprehension
Technical Review	Comprehension
Inspection	Application
Walkthrough	Application
Audit	Comprehension
Technique	Application
Testing	Application
Quality Measurement	Application

3.2.4 Software Quality Knowledge Area in the Context of Software Engineering Graduate Courses

Experience shows that traditional computer science departments were unable to graduate software engineers to meet manufactures expectations. Today's software engineers should be taught main software engineering concepts in addition to technical concepts and software engineering technologies. One problem educators face from the knowledge area side is the content or what specific ideas to be taught? Another one from the pedagogy side is what are the best ways of teaching those ideas? (Saiedian and Weide, 2005).

Experts from different universities, industry, and professional societies helped to create the first volume of the Graduate Software Engineering Reference Curriculum GSwERC that provides a set of recommendations for university educators to use when developing and improving curricula for a software engineering course at a master's degree level. GSwERC concentrates on the knowledge and pedagogy related to the software engineering curriculum and based on recognized bodies of knowledge such as the SWEBOK Guide (Klapholtz et al., 2009).

A result of the GSwERC is the Core Body Of Knowledge CBOK that is expected to be learned by all graduates in every university. It includes knowledge units that mostly based on the SWEBOK taking into account the expected level of the Bloom's taxonomy of educational objectives (Bloom, 1956). Fig. 3.1 illustrates the percentage devoted to CBOK areas while Table 3.3 contains areas of the CBOK with crosscutting topics that are associated with the software quality knowledge.

The proposed structure of the software engineering areas and topics and the associated percentage to each area in the CBOK shows that 8% of the core body of knowledge is pure software quality while each other knowledge area includes software quality related concepts and issues as part of its knowledge. Consider for example requirement engineering where quality is involved in the requirement validation subtopic. Also according to the table testing – a quality technique – makes up 10% of the CBOK. This in turn shows that software quality related topics can make up 30-35% of the recommended subjects by GSwERC for any software engineering master level degree. Analysis of the inclusion of software quality into other SWEBOK KAs that shows how software quality involves much more material than what is presented in current courses is presented later.

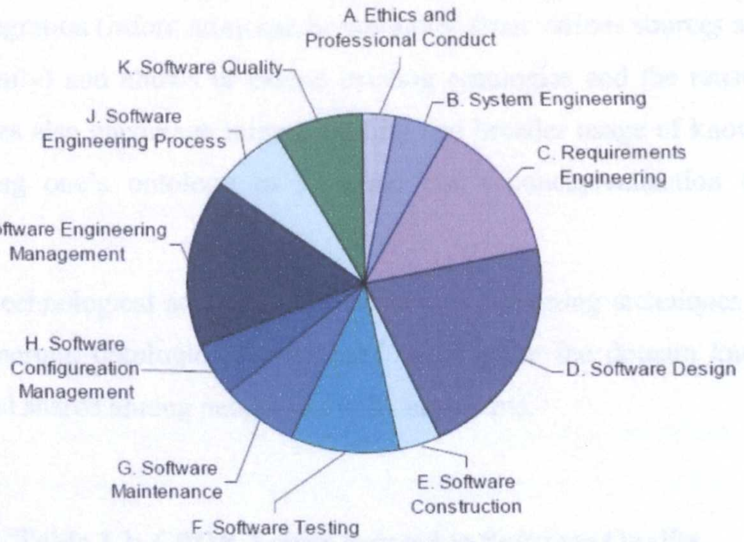


Figure 3.1: Percentage Devoted to CBOK Areas (Klapholtz et al., 2009)

Although software quality makes up to 30-35% of the CBOK, it is rarely to find a computer science curriculum with a dedicated software quality course. This means graduates with lack of software quality knowledge and experience and in turn more complaints from organizations about the new employees' level of knowledge in the field.

Moreover, software engineering teachers have different background, languages, and using different jargon which leads to a variety of understanding and overlapping of the meaning of the same software engineering term or concept and results in lack of communication which in turn leads to difficulties in identifying requirements and defining system specifications. The ambiguity of natural languages of participants leads to mistakes and non-productive effort and limits the potential of reuse and sharing of knowledge.

To reduce and eliminate this conceptual confusion, we need a common understanding and sharing of knowledge of the problem domain and using a general agreement on terminology among all interested people. "Without such a consensus, no licensing examination can be validated, no curriculum can prepare an individual for an examination and no criteria can be formulated for accrediting a curriculum" (Wille et al., 2004). A shared taxonomy of entities called ontology may provide a significant solution to the incompatibility of terms problem. In addition, the flexibility of ontologies eases information integration (information can be combined from various sources and new facts can be infer easily) and allows to extend existing ontologies and the reuse of existing work. Ontologies also encourage interoperability and broader usage of knowledge when allowing relating one's ontology to someone else's conceptualization (Happel and Seedorf, 2006).

With the new technological advances and the use of e-learning techniques for teaching software engineering, ontologies can be used to structure the domain knowledge and make it used and shared among people and software agents.

Table 3.3: CBOK Topics Related to Software Quality

Knowledge Area	Approximate % of the Core
System Engineering • *Verification and Validation*	5%
Requirement Engineering • *Requirement Validation*	14%
Software Design • *Software Design Quality Analysis and Evaluation*	21%
Testing	10%
Software Engineering Management • *Review and Evaluation*	16%
Software Engineering Process • *Process Assessment* • *Product and Project Measurement*	7%
Software Quality	8%

3.3 Existing Ontologies for SE Knowledge Domain

Software engineering projects require high level of communication and exchange of information among projects participants. Having different knowledge background and speaking different languages, makes this type of communication problematic in the field of software engineering. Using ontologies could eliminate this problem. This in turn encourages researchers to propose ontologies in their tools and projects. Classification of ontologies used for semantic-web based software engineering can be found in (Zhao et al., 2009).

3.3.1 The SWEBOK Ontology

The SWEBOK guide provides an international recognized consensus in software engineering terminology. Software engineering domain ontology if one exists will ease the share and reuse of the knowledge accumulated in the software engineering field, and will allow automatic interpretation of this knowledge.

Wille et al. (2003) presented a candidate approach for the design of ontology for SWEBOK. The proposed ontology would include all important concepts of software engineering knowledge supported by definitions and relationships among concepts and arranged in a taxonomic hierarchy. In their proposed approach, Wille and colleagues claimed that the ontology should include all important concepts and sub-concepts of the software engineering knowledge area where SWEBOK represents the super-class and the ten knowledge areas are subclasses of the super-class represented by a structured set of concepts and corresponding definitions. The suggested structure of the ontology includes bidirectional links to internal and external references to allow fast user access to either concept or reference by means of the SWEBOK ontology (Fig. 3.2). The design approach was proposed but the ontology has not been developed yet.

Mendes and Abran (2004) develop a proto-ontology as a starting point to develop a comprehensive ontology for the software engineering knowledge area. This initial ontology was presented in the Web Ontology Language OWL (Antoniou, and Van Harmelen, 2003; Smith et al., 2004) where it defines the concept SWEBOK as the root class of the ontology (which is in-turn a subclass of the *owl:Thing*, a class that contains all classes). The ten knowledge areas were defined as the main classes linked to the root class by the *hasParts* property. Each knowledge area can be successively expanding, revealing new classes with more details.

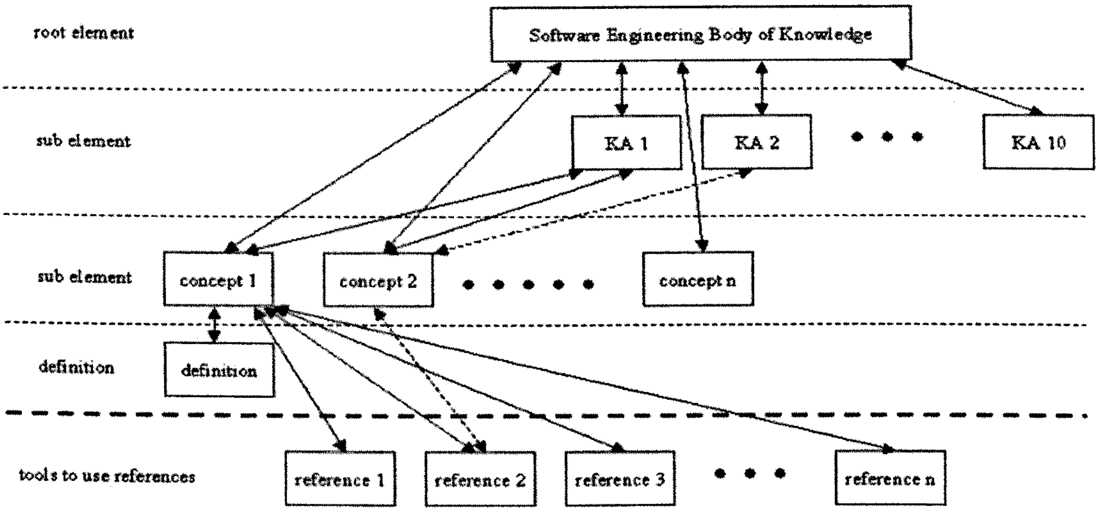


Figure 3.2:Design and Structure of the SE Ontology(Wille et al., 2003)

The ontology classes (super-class and subclasses) are structured in a taxonomic hierarchy using generalization/specialization links. Other types of relations or links used are: *contains, defines, isTopicOf, isDefinitionOf*, etc.

Wille et al. (2003) 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.

To relate the SQA knowledge with other knowledge area of the SE domain, the informal SWEBOK ontology (Wille et al., 2003) was more significant. Their inventory of the term 'quality' in some SWEBOK chapters will be used in the conceptualization phase of the development of the SQA ontology proposed in this thesis.

Although comprehensive domain ontology in software engineering does not exist yet, there are some efforts to develop partial or sub-domain ontologies.

3.3.2 Software Measurement Ontology

García and colleagues (García et al., 2006; 2009) analysed selected existing international standards and research proposals that deals with the software measurements terminology. Commonalities, gaps, and terminology conflicts are identified in order to unify a consistent terminology for software measurement. The proposed Software Measurement Ontology SMO provides a coherent terminology among different software measurement proposals and standards. Fig. 3.3 shows the SMO ontology as illustrated in (García et al., 2006).

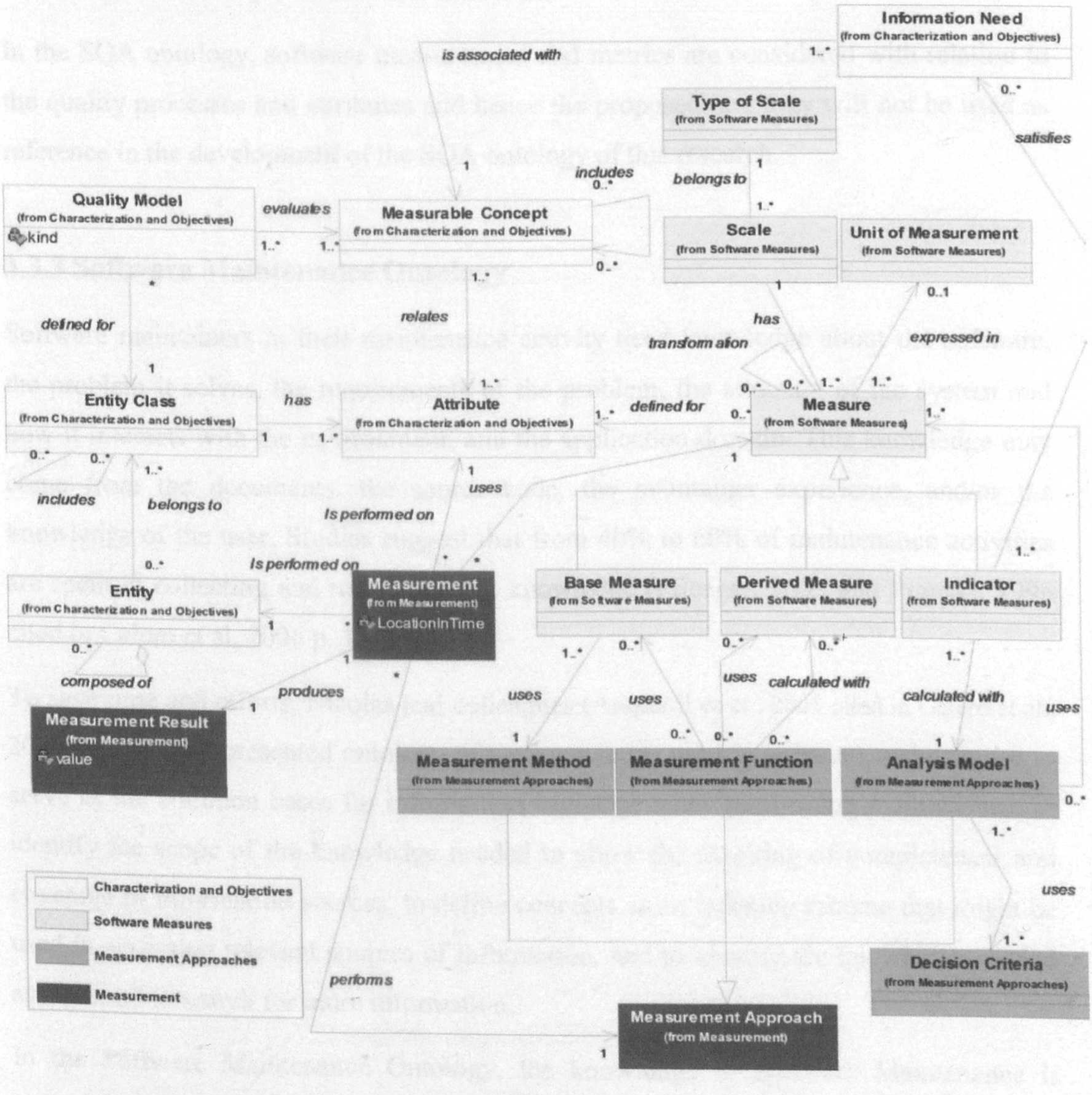


Figure 3.3: UML class diagram of the SMO Ontology (García et al., 2006)

The development of the SMO ontology provides:

- a basis for comparative analysis of software measurement terminology;
- organizations with a set of coherent concepts for carrying the measurement processes and storing their results in a consistent way;
- an important communication medium among organizations;
- a basis for software measurement community to start their future agreement.

Unlike the ontology developed by Wille (2003), the SMO ontology includes detailed knowledge about the measurement process, their attributes and results, while it does not relate them to their SQA metrics and standards.

In the SQA ontology, software measurement and metrics are considered with relation to the quality processes and attributes and hence the proposed ontology will not be used as reference in the development of the SQA ontology of this research.

3.3.3 Software Maintenance Ontology

Software maintainers in their maintenance activity need knowledge about the software, the problem it solves, the requirements of the problem, the structure of the system and how it interacts with the environment, and the application domain. This knowledge may come from the documents, the source code, the maintainer experience, and/or the knowledge of the user. Studies suggest that from 40% to 60% of maintenance activities are spent in collecting and recreating this knowledge (Pfleeger, 2002 and Pigoski, 1996 cited in Calero et al. 2006 p. 156).

To save time and efforts, Nicolas and colleagues (Anquetil et al., 2005 cited in Calero et al., 2006, p. 153-174) presented ontology of the knowledge used in software maintenance to serve as the common bases for information exchange when performing maintenance, to identify the scope of the knowledge needed to allow the checking of completeness and coverage of information sources, to define concepts as an indexing scheme that might be used in accessing relevant sources of information, and to identify the knowledge needed as a ground to search for more information.

In the Software Maintenance Ontology, the knowledge of Software Maintenance is organized into five different aspects: knowledge about the Software System, knowledge about the Maintainer's Skills, knowledge about the Maintenance Process, knowledge about the Organizational Structure, and knowledge about the Application Domain. Competency questions are used to clearly identify the ontology purpose and its intended use. Fig. 3.4 shows how the five sub-ontologies combine in the general ontology.

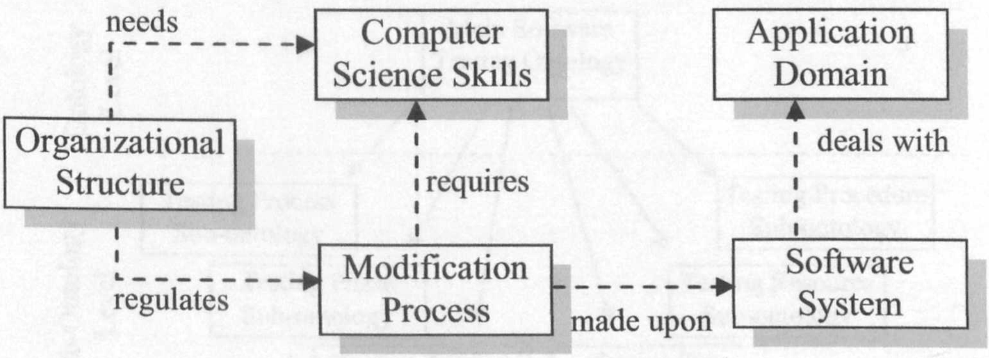


Figure 3.4: Software Maintenance Ontology Overview (Calero et al., 2006)

The developed software maintenance ontology can be used as a classification scheme to categorize information one may need or gather to exchange information (Calero et al., 2006).

As the software maintenance is out of the scope of this research, the proposed software maintenance ontology will not be considered as a reference in developing the SQA ontology.

3.3.4 The OntoTest Ontology

Based on the ISO/IEC 12207 standard, the OntoTest ontology (Barbosa et al., 2006) has been developed to define a common well-defined vocabulary for software testing that can be useful to develop supporting tools and to increase interoperability among tools. OntoTest supports acquisition, organization, sharing, and reuse of the software testing knowledge. OntoTest intends to explore the different aspects involved in the testing activity, techniques and criteria, human and organizational resource, and automated tools.

Fig. 3.5 shows the main concepts of the OntoTest ontology (Testing Process, Testing Phase, Testing Artifact, Testing Step, Testing Procedure, and Testing Resource). The structure of OntoTest makes it flexible to reuse and integrate, depending on the application, as a whole or some of its sub-ontologies.

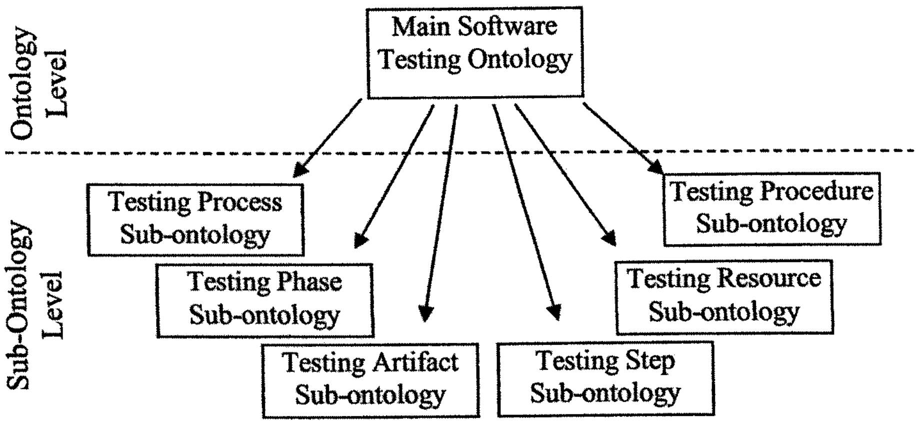


Figure 3.5: OntoTest (Barbsoa, 2006)

Even though, Software Testing as an SQA process is considered in the ontology proposed in this thesis, the detailed tasks of the Software Testing process is out of the scope of our SQA ontology. In our thesis, we have borrowed few aspects of the OntoTest ontology, especially those related to testing processes, and resources, proposed by the *Process* and *Resource* concepts, and the *uses* and *invokes* object properties. In our SQA ontology testing is considered as an SQA process while detailed testing procedures, steps and phases are out of the scope of this research.

3.3.5 Non-Functional Requirements Ontology

In software market, Non-Functional Requirements (NFRs) become more important in distinguishing between competing software products. However, in practice, NFRs receive little attention relative to Functional Requirements (FRs). In his PhD project, Kassab (2009) proposed an ontological representation of the software requirements (FRs and NFRs), their refinements, and their interdependencies. The work identified three views of the NFRs ontology: the first view relates the NFRs with the other entities of the software system being developed. The second view structures the NFRs using classes and properties. The third view represents the measurement process and contains the concepts used to produce measures to measurable NFRs.

Although the first view of the NFRs ontology (Kassab, 2009) gives an impression that the work might be related to the SQA, the structure and view of the NFRs ontology is not related and cannot be beneficial in building the SQA ontology of the current research work.

3.3.6 Ontology for Software Product Quality Attributes

Towards the development of ontology for software product quality attributes (SWPQAs) (Samhan, 2008; Kayed et al., 2009), the most common SWPQAs concepts and terminology were evaluated and extracted from many documents, reports and proposals. General relationships among the suggested concepts are also extracted. TextToOnto, an ontology engineering tool based on text mining techniques and natural language processing algorithms, was used to extract the ontology concepts from 34 related documents. By applying elimination process with the aid of experts in the field the extracted 292 concepts were reduced to 100 and finally 66 SWPQAs concepts based on concepts' frequencies. After using ontology evaluation technique, 125 SWQPAs concepts were agreed.

Believing that reaching coherent ontology concepts accomplishes 70% of the ontology building process, Kayed and colleagues (2009) proposed a framework that aims to identify some important SWPQA attributes concepts that are heavily used at different definitions. As no formal model was proposed, the suggested concepts can be used to evaluate our extraction of the quality attributes concepts as part of the SQA ontology.

3.4 Conclusion

The Chapter presented a historical background of the SE and the SQA knowledge areas. It showed the initiation and objectives of the SWEBOK guide to capture the SE knowledge and establish an agreement on its structure and terminological treatment among the SE community. The SWEBOK guide organizes the SE material into ten knowledge areas. The agreed structure of the SWEBOK guide was presented with the focus on the SQA knowledge area in the context of the SWEBOK guide. Our findings of the inclusion of software quality into other SWEBOK knowledge areas are presented in Chapter 5. SQA in the context of SE graduate courses was reviewed based on the CBOK that is expected to be learned by all graduates in universities.

A research of existing domain ontologies in the field of software engineering have been carried out with the aim to reuse knowledge. Analysis of the pevious works and relations to the current research work also has been proposed. The most related efforts to build ontologies in the field of SE are:

- The informal SWEBOK ontology (Wille et al., 2003) where no ontology was developed and only taxonomy of the SE as presented in SWEBOK. Their inventory of the term 'quality' in some SWEBOK chapters will be used in the conceptualization phase of the development of the SQA ontology proposed in this thesis;
- The software maintenance ontology (Anquetil et al., 2005 cited in Calero et al., 2006, p. 153-174) is out of the scope of this research;
- The SMO (García et al., 2006) does not relate software measurements and metrics to the SQA processes and attributes and hence will not be used in the development of the SQA ontology of the current research;
- The OntoTest ontology (Barbosa et al., 2006). Some aspects related to the testing processes and resources are considered in development of the SQA ontology with relation to other SQA processes, attributes, measurements and metrics;
- The NFRs ontology (Kassab, 2009) is not related and cannot be beneficial in building the SQA ontology;
- The suggested concepts of the SWPQA (Kayad et al., 2009) will be used to evaluate our extraction of the quality attributes concepts as part of the SQA ontology.

The next chapter propose and analyse available ontology development methodologies in order to adopt a methodology to develop the SQA ontology.

Chapter4: Defining SQA Ontology Development Methodology

Ontology is a skeleton of shared structured terms to represent knowledge. Ontology construction is a challenging and expensive process. This chapter starts reviewing the most known ontology development methodologies. A methodology to develop SQA ontology will be adopted using applicable activities from existing ones. The chapter concludes by declaring the requirements for developing software quality ontology for teaching.

4.1 Review and Analysis of Ontology Development Methodologies

Ontology construction is a challenge. Several approaches and methodologies have been reported for developing ontologies: Cyc, Uschold and king, Gruninger and Fox, KACTUS, Sensus, METHONTOLOGY, UPON, and O4IS. Some of these methodologies are concerned with building ontologies from scratch while others reuse and integrate existing ontologies to build new ones (Gomez-Pére et al., 2004).

In this section we survey some of the well-known ontology development methodologies. As no software quality ontology exists, methodologies for building ontologies from existing ones like the KACTUS and Sensus methodologies will not be considered in our survey. Detailed description and analysis of the methodologies can be found in (Fernandez-Lopez and Gomez-Pérez, 2002), (Gomez-Pérez et al., 2004), and (Calero et al., 2006).

Conceptualization is an abstract, simplified view of concepts, objects, and all other entities of domain knowledge and the relationships among them. A conceptual model, the output of the conceptualization process, is defined as an abstract (mental) model of some part of reality (Kabilan, 2007). Conceptual model supports clarity where the graphical representation is easier to understand and use. The conceptual model is easy to understand, modify and maintain. It supports reusability as it can be transformed into different ontology representation languages. In this work, reviewed methodologies will be classified according to their usage of a conceptual model.

4.1.1 Methodologies without Conceptual Model

The ***Cyc*** methodology presented by Lenat and Guha (1990) to build the Cyc, a huge Knowledge Base (KB) of common sense knowledge. Building the Cyc ontology went through three phases: The first phase handles the manual coding of the explicit and implicit pieces of knowledge, in which common sense knowledge is extracted by hand from different sources. The second phase is knowledge coding aided by tools using the knowledge already stored in the Cyc KB. The third phase is also knowledge coding that is mainly performed by tools. The Cyc methodology provides very general approach; no requirement or design processes are specified.

The methodology of ***Uschold and King*** is based on the experience of building the Enterprise Ontology (Uschold and King, 1995) and proposes the first more formal method for building ontologies which was extended in (Uschold and Gruninger, 1996). This methodology consists of the following phases: 1. identify the purpose of the ontology; 2. building the ontology (consists of: capturing the knowledge, coding it, and integrating existing ontology); 3. evaluating the ontology; and 4. documenting the ontology. Three strategies for identifying concepts of the ontology are proposed: a *top-down* approach, in which the concepts are identified from the most abstract to the most specific; a *bottom-up* approach, in which the most specific concepts are identified first then the more abstract ones; and a *middle-out* approach, in which the most relevant concepts are identified first then specialized or generalized into other concepts. A drawback of this method is the direct implementation of the ontology with lack of the conceptual model. According to Gruninger and Fox (1995) these phases are not enough to be considered a methodology as there are no techniques, methods or principle for each of the above stages. Also there is no relationship or recommended order among the stages.

Gruninger and Fox (1995) proposed a very formal methodology based on their experience in building the TOronto Virtual Enterprise (TOVE) project ontology using first-order logic. The TOVE is a set of integrated ontologies for the modelling of business enterprises like the Resource Ontology, the ISO 9000 Quality Ontology, etc. This methodology proposed the first use of the competency questions (a set of natural language questions used to determine the scope of the ontology) in building ontologies. These questions are also used to capture the main concepts, relations, proprieties and axioms of the ontology. The main processes identified for this methodology are:

1. identify motivation scenarios;
2. elaborate informal (natural Language) competency questions;
3. specify the terminology using first order logic;
4. formalize the competency questions;
5. specify axioms using first order logic;
6. specify completeness theorems (conditions under which the solutions of the competency questions are complete).

The Gruninger and Fox methodology is based on building ontologies for the business domain. Due to its high degree of formality, this approach requires the ontology designer to be well familiar with formal logic languages. This high degree of formality may not be required in information systems applications like the one presented in this research. Even though this methodology is logical for ontology building and evaluation, some management and support activities are absent.

4.1.2 Methodologies with Conceptual Model

METHONTOLOGY is a methodology developed in the Artificial Intelligence Laboratory at the Polytechnic University of Madrid (UPM) for building ontologies from scratch, reusing existing ontologies as they are, or by reengineering existing ontologies (Fernandez-Lopez, et al., 1999). METHONTOLOGY is built taking into account the main activities identified by the software development process (IEEE 1074, 1996). The METHONTOLOGY life cycle is based on evolving prototypes where terms can be added, changed, or removed with every new version. The METHONTOLOGY activities are divided into three categories:

- The management activities that include the scheduling, the control, and the quality assurance activities.
- The development-oriented activities that include the specification, the conceptualization, the formalization, and the maintenance activities.
- At the same time with the development activities, the support activities are performed. They include knowledge acquisition, evaluation, integration, documentation, and configuration management. Integration activity is required when building ontology by reusing existing ones.

Due to the existence of translators, formalization is no more a mandatory activity in the building process as the conceptual model can be translated to the implementation model. Among all other reviewed methodologies, METHONTOLOGY is the first one to recommend its notable *Conceptualization* activity that structures the conceptual model of the domain knowledge on tabular and graph notations. Recall that it might sound easier to directly code the ontology into formal language, the conceptual model is easy to understand, modify and maintain.

Nicola and colleagues (2005) propose an incremental ontology development method ***UPON*** (Unified Process for Ontology building). UPON is derived from the Unified Software Development Process and uses the Unified Modelling Language (UML).

What distinguish UPON from other methodologies is its use-case driven nature that aims at building ontologies that serve its users, both humans and software agents. The nature of UPON is iterative as each phase is repeated through the ontology development, and as at each phase the ontology is further extended the UPON is an incremental method.

Each cycle of UPON results in a new version of the ontology and consists of four phases (*inception, elaboration, construction, and transition*). Each phase is also divided into iterations with five workflows that take place in the iteration (*requirements, analysis, design, implementation, and finally test*). UPON identifies the roles of domain experts and information system designers in the ontology development process. UPON also proposed a storyboard mechanism to extract the terminology of the domain expert.

The Ontology 4 Information Systems ***O4IS*** methodology (Kabilan, 2007) adds some recommendations, algorithms and tools to different steps of existing methods. The O4IS proposes a multi-tiered architecture for logical demarcation of the domain of interest, among the reviewed methods O4IS introduces the use of a dual conceptual representation of the target ontology, and it also proposes a series of conceptual analysis patterns (the Semantic Analysis Relationships SARs) that aid in the analysis and conceptualization of the implicit knowledge on the targeted domain. The dual conceptual representation includes: 1) semi-formal representation where the domain knowledge is captured and represented in a reusable conceptual model; and 2) formal representation where the conceptual model is transformed into machine-readable formalism like OWL, RDF or any other ontology representation language.

Kabilan reuses and combines available techniques and methods and links them together to present the O4IS skeletal design methodology.

Except for the METHONTOLOGY and UPON, none of the presented methodologies propose project management processes or post-development processes as most of the methodologies are focusing on the ontology development activities (conceptualization, coding, etc.). Among the previous methods, METHONTOLOGY, UPON and O4IS are the most mature ones. Diagramming, documenting, and versioning aided by specialized tools for UML, are special advantages of UPON over other methodologies. Although it does not consider management, pre/post development activities, the use-case driven, the incremental, and iterative nature distinguish UPON from other methodologies.

To compare the previous methodologies Fernandez-Lopez and Gômez-Pérez (2002) propose a framework based on the comparison with respect to the IEEE standard for developing a project life cycle process IEEE1074:1995. We adopt this framework based on the new version of the IEEE1074-2006. In their comparison framework they analysed the first four methodologies. Two additional methodologies, UPON and O4IS,have been added and assessed with respect to other methodologies as shown in Table 4.1.

In general, the methodologies are not unified; some approaches are completely different from the others. No single methodology meets all the requirements for designing and developing domain ontologies (simplicity, adaptability, understand-ability, reusability…etc.). A comparison framework is illustrated in Table 4.2.

Table 4.1: Comparison of the Methodologies According to IEEE1074:2006

Methodology	Management Processes (initiate, monitor and control)	Development Oriented Processes						Support Processes (Evaluation, configuration management, documentation, and training)
		Pre-development (statement of needs and system allocation)	Development processes			Post-development (installation, operation and support, maintenance)	
			Requirement	Design	implementation		
Cyc	Not proposed	Not proposed	Not proposed	Not proposed	Proposed	Not proposed	No activities identified for configuration management, evaluation and training
Uschold and King	Not proposed	Not proposed	Proposed	Not proposed	Proposed	Not proposed	No activities identified for configuration management and training
Grunninger and Fox	Not proposed	Not proposed	Proposed	Proposed	Proposed	Not proposed	No activities identified for configuration management and training
METHON-TOLOGY	No activities identified for initiating the project	Not proposed	Proposed	Proposed	Proposed	No activities identified for installation, operation and support	No activity identified for training
UPON (2005)	Partially proposed	Not proposed	Proposed	Proposed	Proposed	Not proposed	No activity identified for training
O4IS	Not proposed	No activities identified for system allocation	Proposed	Proposed	Proposed	Proposed	No activities identified for training
Prpoposed Hybrid Methodology	**Proposed**	**No activities identified for system allocation**	**Proposed**	**Proposed**	**Proposed**	**Proposed**	**No activities identified for training**

Table 4.2: Comparison of the Reviewed Methodologies

Methodology	Formality	Understandable	Easy to Follow	Existence of Conceptual Model	Evolving Prototype
Cyc	Low	Yes	Yes	No	No
Uschold and King	Medium	Yes	Yes	No	No
Grunninger and Fox	High	Yes	No	No	No
METHON-TOLOGY	Medium	Yes	Borderline	Yes	Yes
UPON (2005)	Medium	Yes	No	Use UML	Yes
O4IS	Medium	Yes	Yes	Yes	No
Proposed Bybrid Methodology	**Medium**	**Yes**	**Yes**	**Yes**	**Yes**

The method adapted to develop our software quality ontology is based on a combination of guidelines presented in (Gómez-Pérez et al., 2004), some activities of the METHONTOLOGY and the O4IS methodologies, in addition to the project management activities from UPON. Fig.4.1 illustrates the Life Cycle Model (LCM) for developing the Software Quality Assurance Ontology. The idea of O4IS to specify requirements of the designed ontology will be taken into account and detailed activities of the METHONTOLOGY conceptualization phase will be used.

It might sound simpler and faster to implement the formal ontology directly but the conceptual model supports clarity where the graphical representation is easier to understand and use. Moreover, with the semi-formal conceptual model, domain experts can easily validate wither the model matches the purpose it was built for.

4.2 The SQA Ontology Development Methodology

We follow the PMBOK (2004) model of the project life cycle shown in Fig. 4.1. As illustrated in Figure 4.2, the SQA ontology development process consists of four sequential stages (phases): scoping, conceptualization, implementation, and evaluation.

Comparing the two models (Figs. 4.1 and 4.2) we can see that scoping corresponds to the initial phase of the LCM while the intermediate phase consists of the conceptulaization and implementation phases of the current project LCM, and the finally comes the evaluation phase where the developed SQA ontology is evaluated and an approved version is reached.

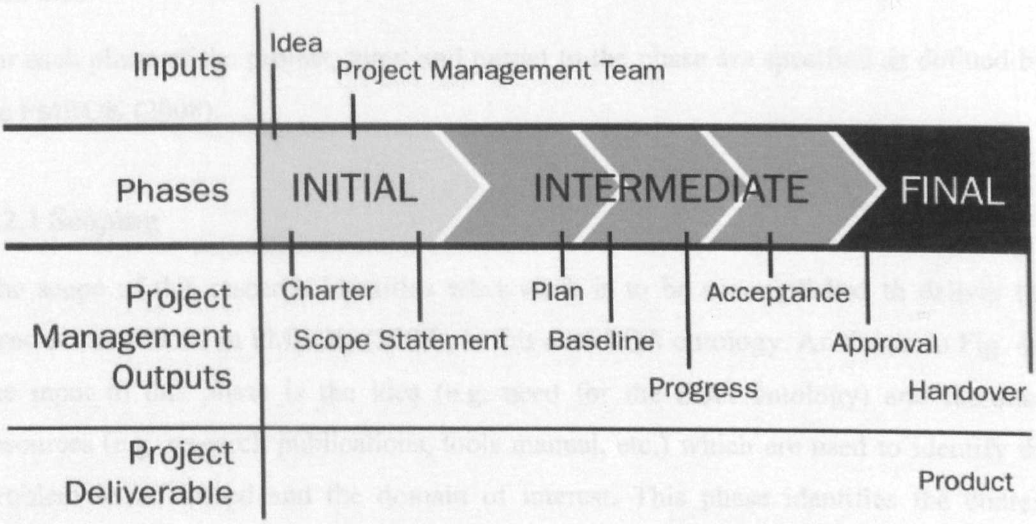


Figure 4.1: Typical Sequence of Phases in Project Life Cycle (PMBOK, 2004)

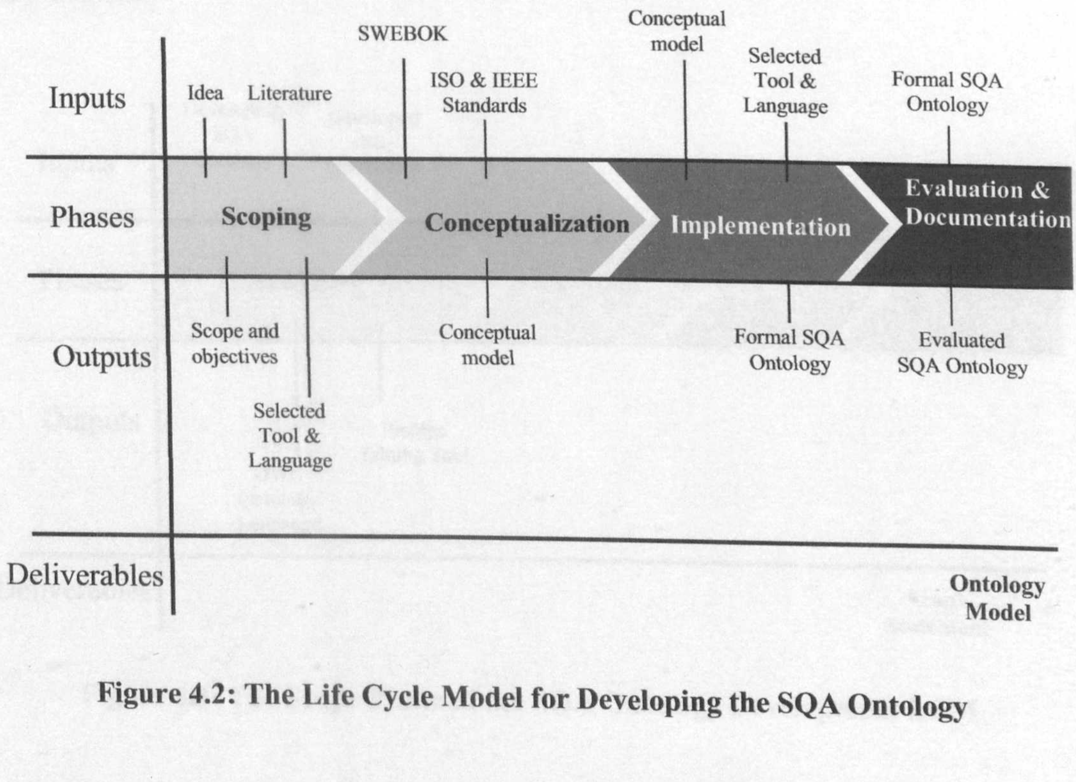


Figure 4.2: The Life Cycle Model for Developing the SQA Ontology

It should be noted that the four phases might be overlapped. For example, the conceptualization stage might starts in parallel while selecting the ontology representation language and tool. Deliverables of particular phase are reviewed and approved before work starts on the next phase. It is important to point out that it is an evolving life cycle where a preliminary ontology prototype is built and then polished with time.

For each phase of the project, input and output to the phase are specified as defined by the PMBOK (2008).

4.2.1 Scoping

The scope of this research identifies what work is to be accomplished to deliver the **product** as defined in PMBOK (2008), in this case SQA ontology. As shown in Fig. 4.3 the input to this phase is the idea (e.g. need for the SQA ontology) and literature resources (e.g. research publications, tools manual, etc.) which are used to identify the problem to be solved and the domain of interest. This phase identifies the context specificity of the ontology, the main features of the domain and how it may relates to other domains.

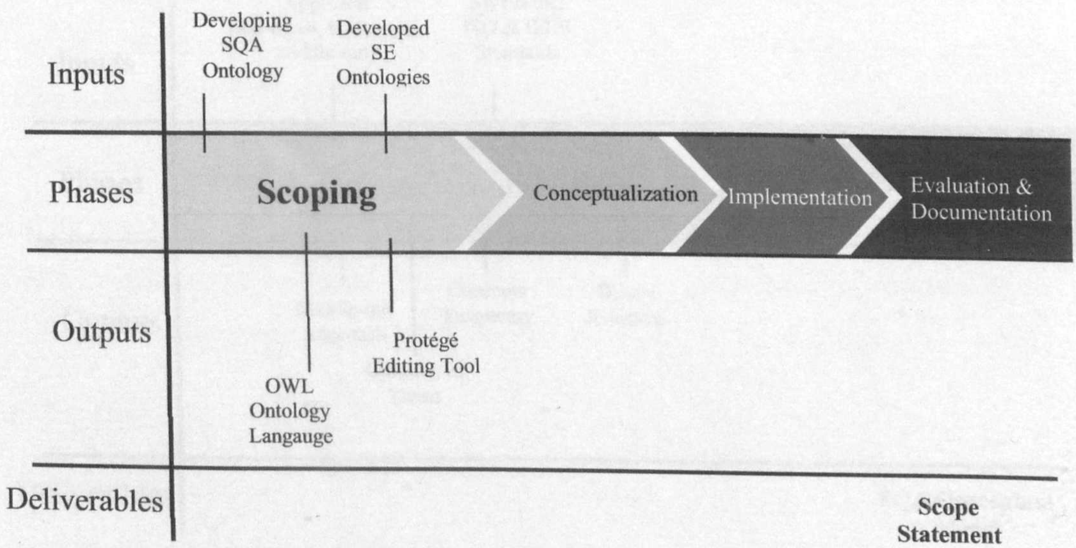


Figure 4.3: The Scope Phase of the SQA Ontology Development LCM

The output is the Project Scope Statement which includes the following elements:

- *Project Objective:* Software Quality Assurance Ontology;
- *Product Characteristics:* the context specificity of the ontology under construction, Section 5.1.1;
- *Project Constraints:* see Section 5.1.2;
- *Selected Language:* see Section 5.1.3;
- *Selected Tool:* see Section 5.1.4.

4.2.2 Conceptualization

Conceptualization is the key phase that affects the rest of the development processes. Conceptualization observes most of the ontology construction time. It starts with the knowledge acquisition process where a description of the domain ontology is developed. Then the acquired knowledge is organized and structured in a conceptual model. Kabilan (2007) defined the conceptual model as an abstract (mental) model of some part of reality that describes the key concepts and relationships. The conceptualization phase is illustrated in Fig. 4.4 and detailed in the following subsections.

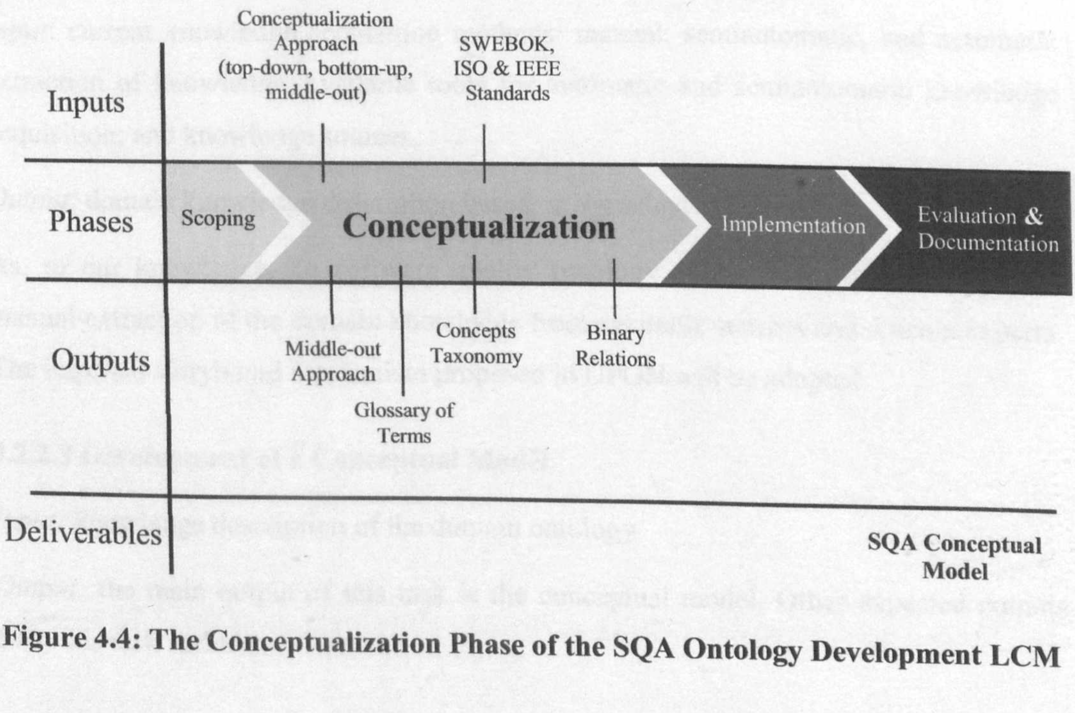


Figure 4.4: The Conceptualization Phase of the SQA Ontology Development LCM

4.2.2.1 The Conceptualization Approach

Input: available approaches: top-down, bottom-up, and middle-out (Gruninger and Fox, 1995).

Output: one of the previous approaches (top-down, bottom-up, and middle-out) is used based on the designer convenience.

Among the available approaches, the designer needs to decide which approach to choose to identify the concepts in the ontology.

If no such ontology exists in the domain, the researcher suggests the middle-out approach where the core concepts are identified first then specifying and generalizing them as required. Uschold and Gruninger (1996) claim that this approach provides a balance level of detail where detail arises as necessary by specialization of the basic concepts which in turn reduce effort. Once the core concepts are derived, other related concepts can be derived from this.

4.2.2.2 Knowledge Acquisition

Selection of the method on how the domain knowledge is to be collected is the first step in this process.

Input: current knowledge acquisition methods: manual, semiautomatic, and automatic extraction of knowledge; available tools for automatic and semiautomatic knowledge acquisition; and knowledge sources.

Output: domain knowledge description based on the selected method.

As, to our knowledge, no software quality ontology exists, the researcher will use manual extraction of the domain knowledge from available sources and domain experts. The informal storyboard mechanism proposed in UPON will be adopted.

4.2.2.3 Development of a Conceptual Model

Input: knowledge description of the domain ontology

Output: the main output of this task is the conceptual model. Other expected outputs from this task include:

- Glossary of terms that identifies relevant terms of the domain with their agreed natural language definitions.
- One or more concept taxonomies to classify the concepts into taxonomic hierarchy (super-class and sub-class relations).
- Ad hoc binary relation diagrams to define the relations between the ontology classes.

The dual conceptual representation of the O4IS method is used where informal knowledge description of the domain from the previous step is transformed into a semiformal representation of the domain or the conceptual model. The researcher will adopt graph and tabular notations as they are more understandable by developers and domain experts.

4.2.3 Implementation

Input: the set of conceptual models from previous phase, ontology development tool

Output: the formal ontology model

As illustrated in Fig. 4.5 at this phase of the project, the conceptual model from previous work is used to specify the ontology components (classes, instances, relations…) in a machine-readable computational model or the implemented ontology.

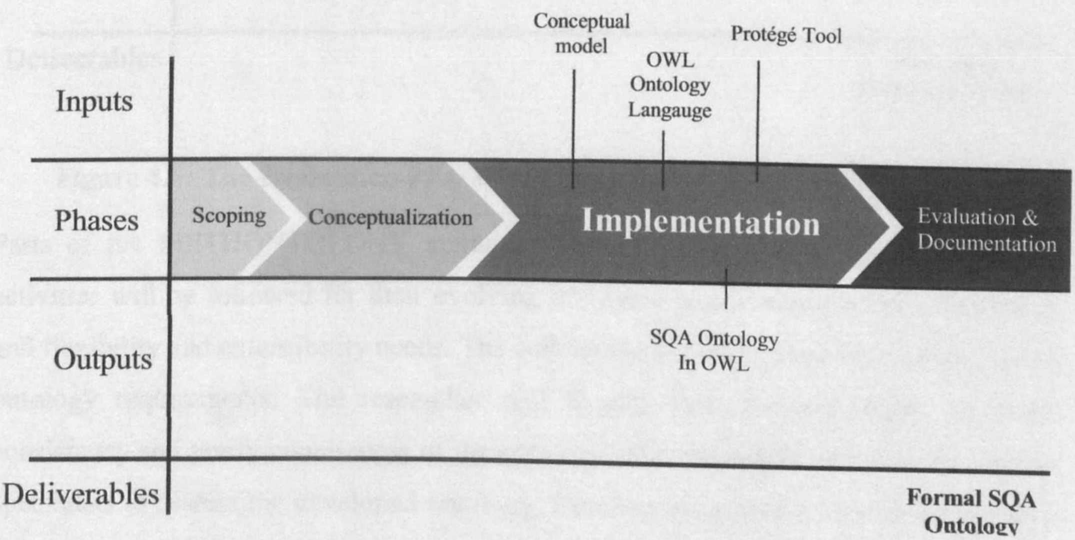


Figure 4.5: The Implementation Phase of the SQA Ontology Development LCM

It also includes writing the code in the selected ontology representation language. Translators of the development tools allow automatic implementation of the conceptual model into several ontology representation languages.

4.2.4 Evaluation and Documentation

Input: the formal ontology model, the ontology requirements, and domain experts

Output: evaluated and verified ontology model. Documents of work accomplished.

A key step is to verify and document the developed ontology model (Fig. 4.6). This is performed at the same time of the previous phases. Technical verification and judgment of the ontology is held and each and every phase of the development process is documented to provide a frame of reference.

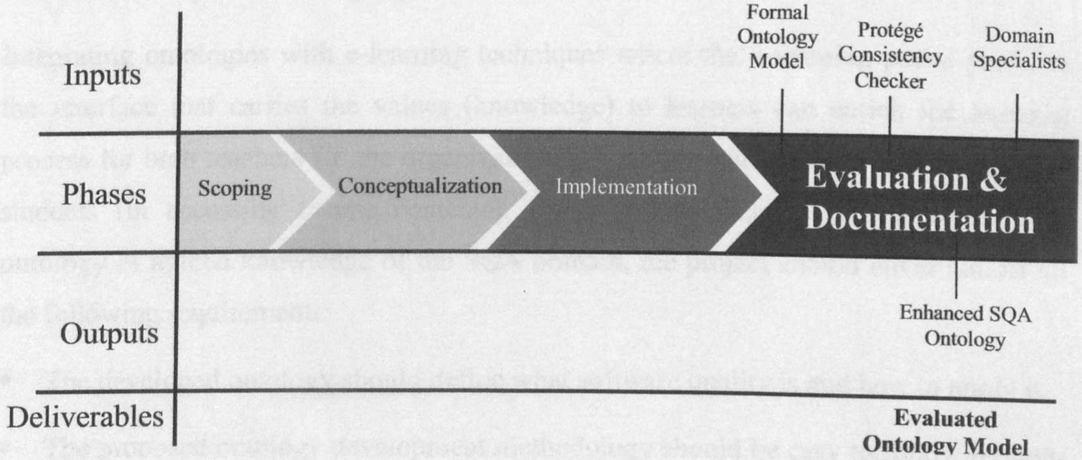


Figure 4.6: The Evalaution Phse of the SQA Ontology Development LCM

Parts of the METHONTOLOGY methods like the documentation and maintenance activities will be followed for their evolving life cycle which supports the adaptability and flexibility and extensibility needs. The conceptual model is verified according to the ontology requirements. The researcher will benefit from Protégé ability to check consistency and verify conciseness of the ontology. The researcher will also use domain specilaists to assesst the developed ontology. Detailed evaluation of the proposed SQA ontology is conducted in Chapter 6.

4.3 Requirements of Ontology for Teaching Software Quality

As the need for producing software increases and does the complexity of software, the need of high standard in the education of people involved in software developments raises. Software engineering textbooks provide sequential representation of the knowledge where the domain is considered as topics and subtopics that are learned linearly. In addition, students with different backgrounds and needs affect the ways of teaching that knowledge. Different views of the same knowledge may exist. Moreover, with large numbers of interrelated terms, meaning of terms may overlap which may lead to misunderstandings or wrong treatment of terms. A reusable and shared representation of the domain knowledge is an obvious solution. As knowledge in software engineering and so software quality is mostly stable, domain ontology will support the reusability and extendibility of knowledge by different users.

Integrating ontologies with e-learning techniques where the e-learning portal provides the interface that carries the values (knowledge) to learners can enrich the learning process for both teachers (in the organization of materials and course construction) and students (in accessing course contents). Since the researcher's aim is to propose ontology of agreed knowledge of the SQA domain, the project should cover almost all the following requirements:

- The developed ontology should define what software quality is and how to apply it.
- The proposed ontology development methodology should be easy to follow by non-ontology experts.
- The conceptual model of the domain should be understandable, sharable, and reusable.
- The knowledge sources should be agreed and standardized to minimize any encoding bias.

4.4 Conclusion

After a review of existing ontology development methodologies, a methodology to build the SQA ontology was presented in this chapter. The adopted methodology consists of four phases: scoping, conceptualization, implementation, evaluation and documentation. As in the USDP ontology development is an iterative process where each phase is repreated and at each cycle the ontology is detailed further and extended in an incremental way.

Chapter 5 shows how these phases are followed to develop the SQA ontology.

Chapter 5: Developing the SQA Domain Ontology

"There is no one correct way to model a domain. There are always viable alternatives...

Ontology development is an iterative process"

Noy and McGuinness, 2001

The domain specific ontology is an ontology that captures general concepts and properties about a learning knowledge domain (software quality assurance in our case). Based on the ontology design principles and criteria (Gruber, 1995), 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 concepts roles, relations, and constraints) open so it can be made later based on the actual need of the problem solving or application.

This Chapter is devoted in details to the SQA ontology development process based on the phases of the development methodology presented in Chapter 4.

5.1 Scoping

Higher quality ontologies can be easier reused and shared with confidence among applications and domains. Additionally in case of re-use, the ontology may help to decrease maintenance costs (Vrandečić, 2009). The SQA ontology must contain well-defined, structured and organized knowledge of the SQA domain including: the type of software process, as well as, its SQA requirements, quality attributes, and corresponding SQA measurement and metrics.

5.1.1 Context Specificity of the Ontology

Any ontology is developed to be used in a particular context. The context influences the ontology because the ontology is a model of some knowledge and any knowledge may be interpreted differently in different contexts. If ontology is created just to model particular 'pure' knowledge, it may be based on the body of the knowledge only (for example: Anquetil et al., 2005 and ~~Bertoa~~ Garcia et al., 2006). The SQA ontology is an engineering area's ontology where general engineering ideas and SQA features in specific should be presented in the ontology.

In this research and according to the requirements, the ontology will be used in a particular learning environment and the development methodology should take into account the following circumstances (Bajnaid et al., 2008):

1) It is an ontology to be used in a teaching environment, and teaching aspects for the discipline should be present in the ontology;
2) There are many 'languages' to describe SE areas, but only a language that best describe software engineering for teaching purposes will be chosen for the ontology;

5.1.2 Project Constraints

Software quality KA with a large number of overlapped terms which are intervened in other software engineering KAs is difficult to be ontologically modelled within the time boundary of this thesis. For this reason only a prototype ontology model is developed.

The lack of ontology development experts with software quality expertise is another constraint that affects the SQA ontology development and evaluation processes.

5.1.3 Selected SQA Ontology Development Language and Tool

As defined in Section 2.4, the Web Ontology Language OWL has been selected in this research as an ontology representation language. In addition, the Protégé ontology editing tool has been selected as defined in Section 2.5.

5.2 Conceptualization

The main description of the SQA is developed to provide agreed organized and structured conceptual model of the domain.

5.2.1 Existing Vocabularies

There are various vocabularies to describe the software quality domain knowledge. There is no single standard which embraces the whole software quality knowledge.

Different standards and proposals are using different terminologies for the same term. Similarly, the same term may be used to refer to different concepts. This issue has been recognized by the international standards and in 1987 the ISO/IEC has established the Joint Technical Committee 1 (JTC1) workgroup to guarantee consistency and coherency among standards. Also in 1999 the IEEE computer society and the ISOJTC1-SC7 agreed to harmonize terminology among their standards.

The primary source of the SQA ontology is the SWEBOK guide (SWEBOK 2004) in addition to above-mentioned ISO and IEEE standards (ISO 9126, IEEE 12207, IEEE 610.12, IEEE 00100, PMBOK 2008, CMMI v1.2) and research proposals.

Table 5.1 shows examples of paragraphs related to software quality as appear in the SWEBOK guide. In the table references such as p 2-1 means page 1 of Chapter 2 as appears in SWEBOK. 16 SQA terms have been extracted from the SWEBOK guide.

Table 5.1: SWEBOK Paragraphs Related to SQA

List of paragraphs in SWEBOK related to SQA	Corresponding terms
An essential property of all software requirements is that they be verifiable. (p2-1)	Verification
The choice of verification technique is one example. (p2-2)	Verification, Technique
Care must be taken to describe requirements precisely enough to enable the requirements to be validated, their implementation to be verified (p2-6)	Requirement, Validation, Verification
Requirement Validation (p2-8)	Requirement, Validation
Requirement Review (p2-9)	Review
Acceptance Test (p2-9)	Testing
Software Design Quality Analysis and Evaluation (p3-4)	Software Quality
Quality Attributes (p3-4)	Quality Attribute
Software Quality is also closely linked to Software Construction (chap4, introduction)Construction Quality (p4-4)	Software Quality

Table 5.1: continued

List of paragraphs in SWEBOK related to SQA	Corresponding terms
Test Techniques (p5-5)	Technique
There are likely to be specific SQA requirements for ensuring compliance with specified SCM processes and procedures (p7-5)	SQA
Audits can be carried out during the software engineering process (p7-5)	Audit
a project support library could support testing (p 7-7)	Testing
Software requirement methods for requirements elicitation (for example, observation), analysis (for example, data modelling, use-case modelling), specification, and validation (for example, prototyping) must be selected and applied… (p 8-3)	SW requirement, method, validation
Selection of the appropriate software life cycle model… and the adaptation and deployment of appropriate software life cycle processes are undertaken in light of the particular scope and requirements of the project. Relevant methods and tools are also selected. (p 8-4)	Process, method, tool
achievement of process and product improvement efforts can only be assessed if a set of baseline measures has been established (p 9-5)	Process, product , metric
Measurement can be performed to support the initiation of process … or to evaluate the consequences of process implementation and change, or it can be performed on the product itself. (p 9-5)	Process, product, measurement
Process Definition Review is a means by which a process definition (either a descriptive or a prescriptive one, or both) is reviewed (p 9-7)	Process, review
Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks, and guidelines for checking both the process and the product (p 10-1)	Method, task, process, product

Table 5.1: continued

List of paragraphs in SWEBOK related to SQA	Corresponding terms
Tools are often designed to support particular software engineering methods (p 10-1)	Tool, method
software requirements define the required quality characteristics of the software and influence the measurement methods (p11-1)	SW Requirement, Q characteristic, method
Specific process areas related to quality management are (a) process and product quality assurance, (b) process verification, and (c) process validation (p 11-3)	QA, verification, validation
A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements (p 11-4)	Verification, validation, SW quality, requirement

Traceability matrices was built to track the mentioning of the SQA terms in the SWEBOK guide as illustrated in Table 5.2. As the focus of the work is an SQA vocabulary, the root concept of the SQA ontology is the *SQAConcept* where all SQA terms are sub-concepts of it.

Table 5.2: Traceability Matrix of SQA terms in SWEBOK

Term	Its mentioning in the SWEBOK Guide
SW quality	• Requirement Validation (p2-8) • A number of key issues must be dealt with when designing software. Some are quality concerns that all software must address (p3-3) • Software Design Quality Analysis and Evaluation (p3-4) covers quality issues • Construction Quality (p4-4) • Software Quality is considered in the introduction of chap5 (Testing) • Sec (6.3.2.5) considers software quality (p 6-8) • Sec 10.1.9 considers SW quality tools (p10-3) • Chap 11 of the guide considers SW quality in all its sections

Table 5.2: Continued

Term	Its mentioning in the SWEBOK Guide
SW product	• Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem. (p2-1) • Product parameters are requirements on software to be developed (p2-2) • Testing is an activity performed for evaluating product quality (p5-1)
Requirement	• Chap2 of the guide considers SW requirement in all its sections • Requirement Validation (p2-8) • Process for the Review and Revision of Requirement (p 8-4) • Sec 10.1.1 considers SW requirement tools (p 10-2)
SW process	• According to the IEEE definition (IEEE 610.12-90), design is both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" (p1-4) • A process parameter is essentially a constraint on the development of the software (p2-2) • Chap 9 of the guide considers SE Process • Sec 10.1.8 considers SE process tools (p 10-3)
SW process	• In a standard listing of software life cycle processes such as IEEE/EIA 12207 Software Life Cycle Processes (p3-1) • Software maintenance is considered as one of the primary life cycle processes (p 6-1) • Software Configuration Management is considered as a SW life cycle process (p 7-1) • "the particular software life cycle process chosen for a software project… affect the design and implementation of the SCM process" (p 7-2) • SW life cycle process considered in sec 9.2 Process Definition (p 9-3) • Software development tools are the computer-based tools that are intended to assist the software life cycle processes (p 10-1)

Table 5.2: Continued

Term	Its mentioning in the SWEBOK Guide
Quality Assurance	• SCM is closely related to the software quality assurance (SQA) activity (p7-1) • Sec 11.2.1 considers QA (p 11-4)
Q characteristic	• Several Q characteristics are considered in sec 5.2.2 (Objective of Testing) • A software engineer should understand the underlying meanings of quality concepts and characteristics and their value to the software… (p 11-1) • Sec 11.1.3 considers Qcharacteristics (p 11-2)
Verification	• Considered in sec 11.2.2 Verification and Validation (p 11-4)
Validation	• The Software Requirements Knowledge Area (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements. (p2-1) • Requirement Validation (p2-8) • Considered in sec 11.2.2 Verification and Validation (p 11-4)
Measurement	• Measuring Requirement (p2-10) • Measures (p3-4) • Construction Measurement (p4-3) • Software Maintenance Measurement (p6-6) • SCM Measures and Measurement (p 7-5) • Implementation of Measurement Process (p 8-5) • SE Measurement (p 8-6) • Process and Product Measurement (p9-5) • Sec 10.1.7 considers SW measurement tools (p 10-3)
Testing	• Chap5 of the guide considers software testing in all its sections • Sec 6.2.1.2 considers testing • Sec 10.1.4 considers SW testing tools (p 10-2)

Table 5.2: Continued

Term	Its mentioning in the SWEBOK Guide
Review	• Requirement Review (p2-9) • Software Design Reviews considered in (p3-4) • Review and Evaluation (p 8-6)
Metric	• readers will encounter terminology differences in the literature; for example, the term "metrics" is sometimes used in place of "measures." (p 8-7) • achievement of process and product improvement efforts can only be assessed if a set of baseline measures has been established (p 9-5)
Method	• The availability of methods and tools. (p2-7) • a method is a notation (or set of notations) supported by a process which guides the application of the notations. (p2-7) • Relevant methods and tools are also selected (p 8-4) • Sec 10.2 considers SE methods in all its subsections
Tool	• The availability of methods and tools. (p2-7) • Sec 7.1.3.3 Tool Selection and Implementation (p 7-4) • Software is built using particular versions of supporting tools (p 7-9) • Relevant methods and tools are also selected (p 8-4) • Sec 9.2.5 considers automated tools (p 9-4) • Sec 10.1 considers SE tools in all its subsections

The previous sources aided by domain specialists have been used to build the glossary of terms illustrated in Table 5.3 (Bajnaid et al., 2013). In the Table the terms were classified based on the text from the different sources used to extrat these SQA terms.

Table 5.3: Glossary of Terms of the SQA Domain Ontology

Term	Super-concept	Definition	Source
SQA (SQAConcept)	owl:Thing	A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.	IEEE 610-12
Project	SQAConcept	A temporary endeavour undertaken to create a unique product, service, or result.	PMBOK 2008
Process (SQAProcess)	SQAConcept	A set of activities that can be recognized as implementation of practices for specific purpose A set of interrelated actions and activities performed to achieve a specified set of products, results, or services. Set of interrelated or interacting activities which transforms inputs into outputs	Adapted from CMMI v1.2 PMBOK 2008 ANSI/ISO/ASQ Q9000-2000
Attribute (Quality Attribute)	SQAConcept	A measurable physical or abstract property of an entity.	ISO/IEC 9126
Deliverable	SQAConcept	A software product that is required by the contract to be delivered to the acquirer or other designated recipient Any unique and verifiable product, result, or capability to perform a service that must be produced to complete a process, phase, or project.	IEEE 00100 PMBOK 2008

Table 5.3: Continued

Term	Super-concept	Definition	Source
Product	SQAConcept	A work product that is intended for delivery to a customer or end user. The form of a product can vary in different contexts. (1)The complete set of computer programs, procedures, and possibly associated documentation and data designated for delivery to a user. (2) Any of the individual items in (1) The set of computer programs, procedures, and possibly associated documentation and data Result of a process	CMMI v1.2 IEEE 610-12 ISO/IEC 12207 ANSI/ISO/ASQ Q9000-2000
Requirement	SQAConcept	A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents	IEEE 610-12
Requirement	SQAConcept	Need or expectation that is stated, generally implied or obligatory	ANSI/ISO/ASQ Q9000-2000
Functional Requirement	Requirement	A requirement that specifies a function that a system or system component must be able to perform. requirements which focus on "what" the software does	IEEE 610-12 (Paech, 2004)

Table 5.3: Continued

Term	Super-concept	Definition	Source
Non-functional Requirement	Requirement	An attribute of or a constrain on the system Requirements focusing on "how good" software does something as opposed to the functional requirements which focus on "what" the software does	(Chung, 2000) (Paech, 2004)
Resource	SQAConcept	Any capability that must be scheduled, assigned, or controlled by the underlying implementation to assure no conflicting usage by processes.	IEEE 00100
Technique	Resource	A defined systematic procedure employed by a human resource to perform an activity to produce a product or result or deliver a service, and that may employ one or more tool.	PMBOK 2008
Tool	Resource	A software or hardware devise used to analyse the performance of a software or system component	Adapted from IEEE 00100
Method	Resource	A formal, well-documented approach for accomplishing a task, activity, or process step governed by decision rules to provide a description of the form or representation of the outputs.	IEEE 00100

Table 5.3: Continued

Term	Super-concept	Definition	Source
Measurement	SQAConcept	The determination of the magnitude or amount of a quantity by comparison (direct or indirect) with the prototype standards of the system of units employed. the use of a metric to assign a value (which may be a number or category) from a scale to an attribute of an entity	IEEE 00100 ISO/IEC 9126
Measurement Metric	SQAConcept	A quantitative measure of the degree to which system, component, or process possesses a given attribute. the defined measurement method and the measurement scale	IEEE 610-12 ISO/IEC 9126

The terms *Product* and *Deliverable* are examples of SE terms with overlap meaning. In ISO/IEC 25010 (2011) the term *Product* specifies target and non-target software products; and the term *Deliverable* specifies non-executable software product such as documentations and manuals. In the SQA ontology developed in this research, the term *Deliverable* has been used to specify any work product produced in a software project as in SWEBOK (2004) and PMBOK (2008) (Bajnaid et al., 2010). In addition, the term *SQAProcess* will be used to represent the concept *Process* to differentiate SQA process(es) considered in the current research work to develop the SQA ontology from other software engineering process(es) (Bajnaid et al., 2013).

5.2.2 SQA Ontology Concepts

Basics concepts of the SQA domain are represented by OWL classes that are the roots of various taxonomic trees. The root class of any OWL ontology is the *owl:Thing* where every individual of the OWL world is a member of that class. Thus every class is a subclass of *owl:Thing*. The recommended naming convention for OWL classes is that all class names should start with a capital letter and should not contain spaces (Horridge et al., 2007). This naming convention is consistently used for creating the SQA ontology classes and subclasses.

As shown in Fig.5.1, the main class in the domain ontology is class *SQAConcept,* a subclass of *owl:Thing*, is the upper class of all other classes of the SQA ontology that is used to conceptualize and to represent the knowledge of the SQA domain. It is important to know that in OWL classes are overlapping until they are specified as *disjoint*. An individual cannot be an instance of more than one of disjoint classes. The "Disjoint Widget" of the Protégé tool is used to specify disjoint classes. In the SQA ontology, *Process*, *Project*, *Deliverable*, *Measurement*, *MeasurementMetric*, *Resource*, and *Quality_Attribute* have been made disjoint from one another. For example it is not acceptable for an individual to be a *Process* and a *Deliverable* at the same time.

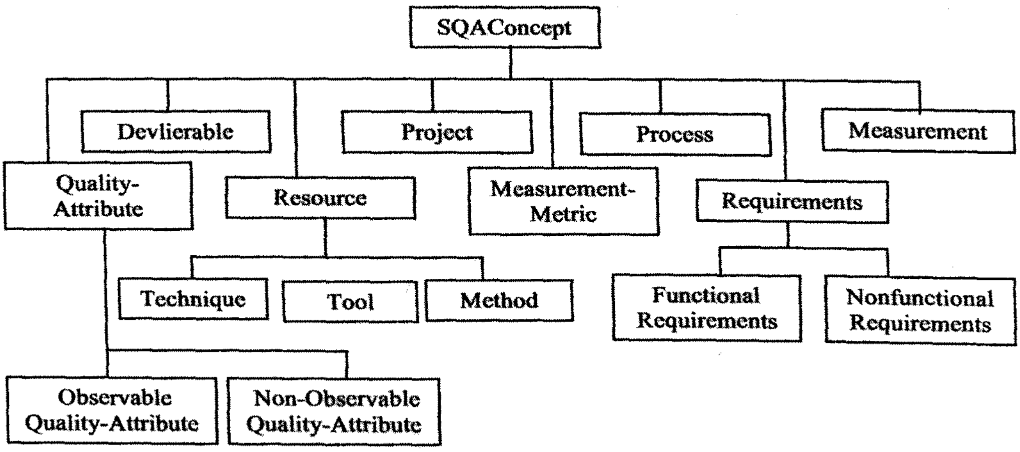


Figure 5.1: Top Level of the SQA Ontology Concepts

5.2.3 SQA Ontology Properties

As it has been defined in Section 2.3, the ontology properties are used to describe relationships among individuals of the classes. Various properties are used to describe both static and dynamic aspects of the SQA knowledge, such as SQA-processes and related SQA issues. The ontology provides a formal description for SQAProcess which may have Quality Attributes (QAs) which can be measured by a quality measurement. Various quality assurance processes, such as *Validation*, *Verification*, and *Audit* can be instantiated. A process may use various resources (e.g. techniques and CASE tools). The recommended naming convention is that a property names start with a lowercase letter and the remaining words capitalized with no spaces. To make the intent of the property clear to human, it is also recommended that a property is prefixed with the words 'has' or 'is', such as *hasPart*, and *isPartOf*. This convention has been used to describe the properties of the SQA ontology.

An object property may have a corresponding inverse property. For instance the properties *use (p, r)* and *isUsedBy (r, p)* that relate a process with a resource are inverse properties. Another characteristic that are added to the property description is the cardinality constraint. Cardinality constraint is a built-in OWL property used to describe the number of relationships an individual may participate in for a given property. An OWL property relates individuals of the domain class to individuals of the range class. Story board technique was used to define properties among the SQA concepts. Table 5.4 presents properties of the SQA ontology. For each property, the table presents its domain, range, inverse property (if any), and cardinality.

Table 5.4: SQA Ontology Properties

Name	**Domain**	**Range**	**Cardinality**	**Inverse Property**
hasProcess	Project	Process	Multiple: a project may have more than one process	-
enforces	Process	Quality-Attribute	Multiple: a process may enforces (ensures) more than one attribute	enforcedBy
uses	Process	Resource	Multiple: a process may use more than one resource	isUsedBy
isInputTo	Deliverable	Process	Multiple: a process may have more than one deliverable as input	isInputTo
invokes	Process	Process	Multiple: a process might invoke other process (es)	-

Table 5.4: Continued

Name	Domain	Range	Cardinality	Inverse Property
measures	Measurement	Quality-Attribute	Single: a measurement can be used to measure specific quality attribute	isMeasuredBy
produces	Process	Deliverable	Multiple: A process might *produce* one or more deliverables	isProducedBy
hasMeasurement Metric	Measurement	Measurement Metric	Multiple: a measurement may have one or more metric	isMeasurement MetricOf
conductedUsing	Measurement Metric	Process	Multiple: a measurement metric maybe conducted using one or more process(es)	-

For each class in Fig. 5.1 we build a structure to represent it. Example structure of the *Process* class is shown in Fig. 5.2 while *Appendix A* shows the structure of other SQA classes.

OWLClass: Process supClassOf: owl:Thing Examples: quality assurance, validation, and verification all are individuals (instances) of the class process			
Object Property	*Domain*	*Range*	*Cardinality*
uses	Process	Resource	1..n
invokes	Process	Process	1..n
produces	Process	Deliverable	1..n
enforces	Process	Quality_Attribute	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

Figure 5.2: Structure of the *Process* class

5.2.4 Quality Measurements and Metrics

For any 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 (Bishop and Lehman, 1991).

Measurement plays an important part in software development. It can be used to indicate the quality of the product being developed (Pressman, 2005). According to Pressman's categorization of software metrics, quality metrics, which measure how the customer requirements are fulfilled, indicate how closely software conforms to explicit and implicit customer requirements.

In this study, software measurements and metrics are at the heart of the SQA ontology design. All aspects of SQA measurement and metric as described in the ISO/IEC 9126 standard are reflected in the proposed SQA ontology as instances (OWL individulas) of the *Measurement* and *Measurement-Metric* classes respectively. Table 5.5 shows the knowledge about the SQA measurements and metrics related to different quality attributes extracted from the ISO/IEC 9126 standard (Bajnaid et al., 2012).

Based on the international standard of software engineering product quality ISO/IEC 9126, each quality attribute associated with several characteristics and sub-characteristics.

In the table, *Measurement* represents quality characteristics while the *Metric* name represents the quality sub-charaterstics. The input represents source of data used in the measurement process (or measurement formula) while the ISO/IEC 12207 reference identifies software life cycle process(es) where the metric is applicable.

Table 5.5: Quality Measurements and Metrics According to ISO/IEC 9126

Quality Attribute	Measurement	Metric Name	Input to Metric	ISO/IEC 12207 Ref.
Reliability	Recoverability	Availability	Test report	Qualification testing
		Restartability	Test report	Qualification testing Validation
		Restorability	Req. specification User manual Test report Review report	Qualification testing Validation Verification Joint review
	Maturity	Failure resolution	Test report	Qualification testing
		Fault density	Test report Operation report Problem report	Qualification testing Quality Assurance
		Mean Time Between Failures	Test report	Qualification testing
		Test coverage	Req. specification Test report User manual	Qualification testing Validation Quality Assurance
		Fault detection	Review report	Verification Joint review
		Fault removal	Test report Fault removal report Review report	Verification Joint review
	Fault Tolerance	Failure avoidance	Test report Review report Req. specification	Validation Qualification testing Verification Joint review

Table 5.5: Continued

Quality Attribute	Measurement	Metric Name	Input to Metric	ISO/IEC 12207 Ref.
Usability	Learnability	Ease of function learning	Test report User monitoring record	Validation Qualification testing
	Operability	Error correction	Test report User monitoring record	Validation Qualification testing
		Undoability	Test report User monitoring record	Validation Qualification testing
		Input validity checking	Req. specification Design Review report	Verification Joint review
		Message clarity	Test report User monitoring record	Validation Qualification testing
	Understand-ability	Completeness of description	User manual Test report Req. specification Design Review report	Qualification testing Verification Joint review
		Understandable input and output	User manual Test report	Validation Qualification testing
Functionality	Accuracy	Accuracy to expectation	Req. specification User manual Test report	Validation Quality Assurance

Table 5.5: Continued

Quality Attribute	Measurement	Metric Name	Input to Metric	ISO/IEC 12207 Ref.
Functionality	Accuracy	Computational Accuracy	Req. specification Test report Design Source code Review report	Validation Quality Assurance Verification Joint review
		Precision	Req. specification Test report Design Source code Review report	Validation Quality Assurance Verification Joint review
	Interoperability	Data exchangeability	Req. specification User manual Test report Design Source code Review report	Validation Verification Joint review
	Security	Access controllability	Test specification Test report Operation report Req. specification Design Source code Review report	Validation Quality Assurance Joint review

Table 5.5: Continued

Quality Attribute	Measurement	Metric Name	Input to Metric	ISO/IEC 12207 Ref.
Functionality	Security	Data corruption prevention	Test specification Test report Operation report Req. specification Design Source code Review report	Validation Qualification testing Operation Joint review
Port-ability	Installability	Ease of installation	Problem report Operation report	Qualification testing
		Installation flexibility	Req. specification Review report	Validation
	Portability compliance	Portability compliance	User manual Test report Design Source code Review report	Validation Qualification testing Verification Joint review
Efficiency	Time behaviour	Response time	Testing report	Qualification testing
	Resource utilization	I/O utilization	Source code	verification
	Efficiency compliance	Efficiency compliance	User manual Testing report Design Source code Review report	Validation Qualification testing Verification Joint review
Maintain-ability	Maintainability compliance	Maintainability compliance	User manual Test report Design Source code Review report	Validation Qualification testing Verification Joint review

5.2.5 SQA Ontology Individuals

Individuals represent instances of the domain. The following list represents examples of the software quality related processes extracted from the ISO 12207 and ISO 15288 standards as instances of the concept *Process*:

- Software Qualification Testing process
- Software Quality Assurance process
- SW Verification process
- SW Validation process
- SW Review process
- SW Audit process

Table5.6 shows the list of individuals of each SQA ontology class. The developed ontology contains 16 deliverable concepts, 24 SQA measurement concepts, 27 measurement metric concepts, 11 processes, 8 quality attributes, and 8 resources (partially in Bajnad et al., 2011; 2012).

Table 5.6: List of Class Individuals

SQA Ontology Class	List of Individuals
Process	Validation, verification, audit, review, inspection, joint review, technical review, management review, testing, quality assurance, SW design quality evaluation.
Quality_Attribute	Efficiency, functionality, maintainability, portability, reliability, usability, reusability.
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.

Table 5.6: Continued

SQA Ontology Class	List of Individuals
Metric	Access controllability, accuracy to expectation, availability, completeness of description, computational accuracy, data corruption prevention, data exchangeability, ease of installation, ease of function learning, error correction, failure avoidance, failure resolution, fault density, fault detection, fault removal, I/O utilization, input validity checking, installation flexibility, mean time between failure, message clarity, precision, response time, restartability, test coverage, restorability, Undoability.
Resource	Check list, complexity analysis, control flow analysis, meeting, prototyping, simulation, use cases, and walk through.
Measurement	Accuracy, efficiency compliance, fault tolerance, Installability, interoperability, learnability, maintainability compliance, maturity, operability, portability compliance, recoverability, resource utilization, security, time behaviour, understandability.

5.2.6 The SQA Taxonomy

A complete taxonomy of the SQA ontology is illustrated in Fig.5.3 (Bajnaid et al., 2013). The figure shows the main SQA concepts as OWL classes where the arrows represent relationships (OWL object properties) between domain classes (the head of the arrow) and range classes (the tail of the arrow) where the name on the line depicts the name of the relationship. The individuals are modelled as 'objects' or literals in the rectangular boxes. The is-a property relates an SQA concepts with its instances (OWL individuals). In the model, *Process* and *Measurement* are concepts (classes) while Use Cases and Test Coverage are instances of the classes *Technique* and *Measurement-Metric* respectively. Here we have followed some of the RDF graph notation for describing tuples.

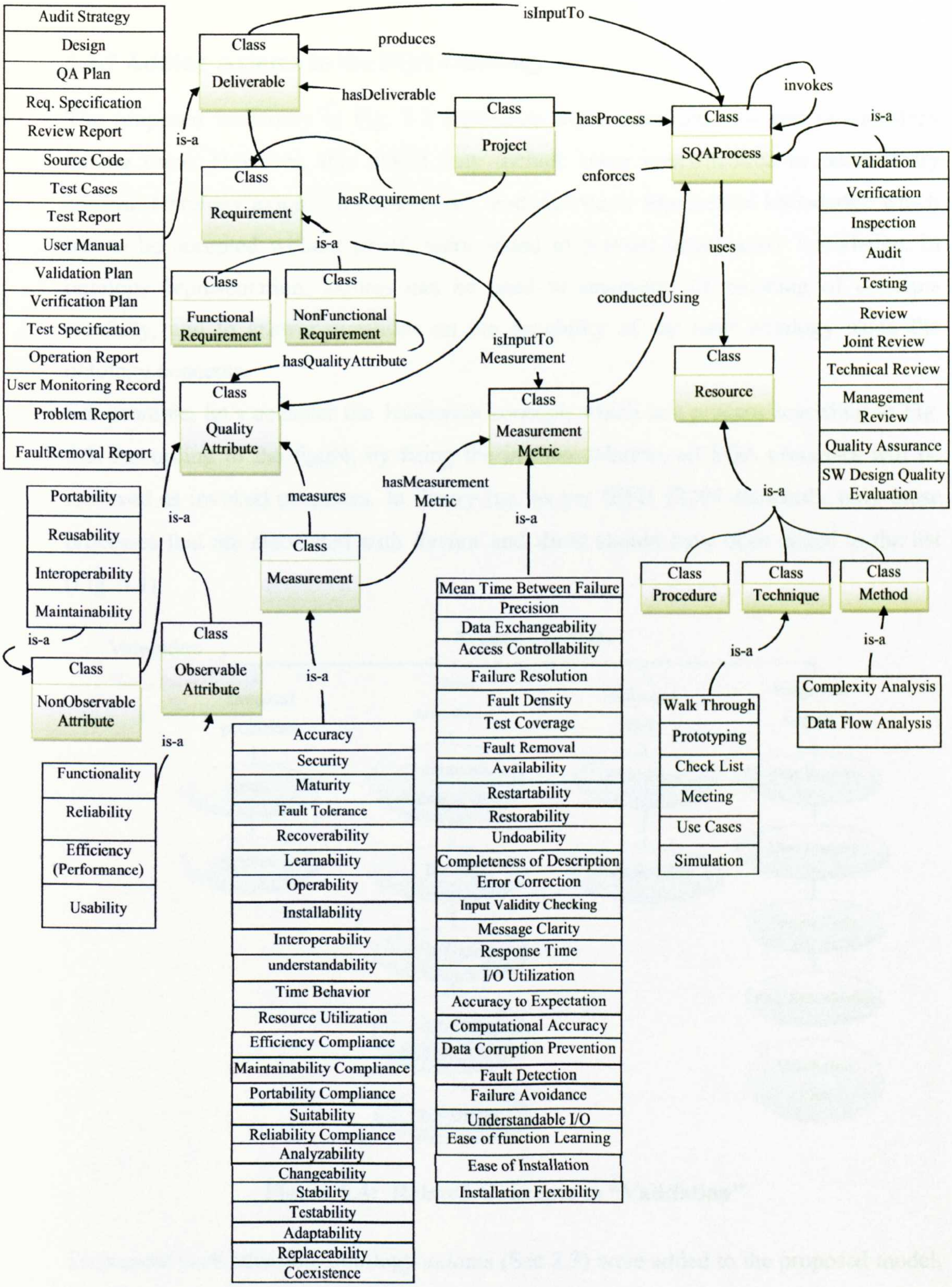


Figure 5.3: Proposed Taxonomy of the SQA Ontology

5.2.7 Adding Axioms to the SQA Ontology

The proposed taxonomy in Fig. 5.3 represents SQA main concepts and relationships among them. However, this model may include some overwhelmed or unnecessary content. Ontology axioms, 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 carefully, and to answer questions on the capability of the built ontology using the ontology concepts.

For example, let's consider the *Validation* concept, which is a process according to Fig. 5.3. According to the figure, by firing the *invokes* relation, all SQA processes will be retrieved as invoked processes. 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 (Fig. 5.4).

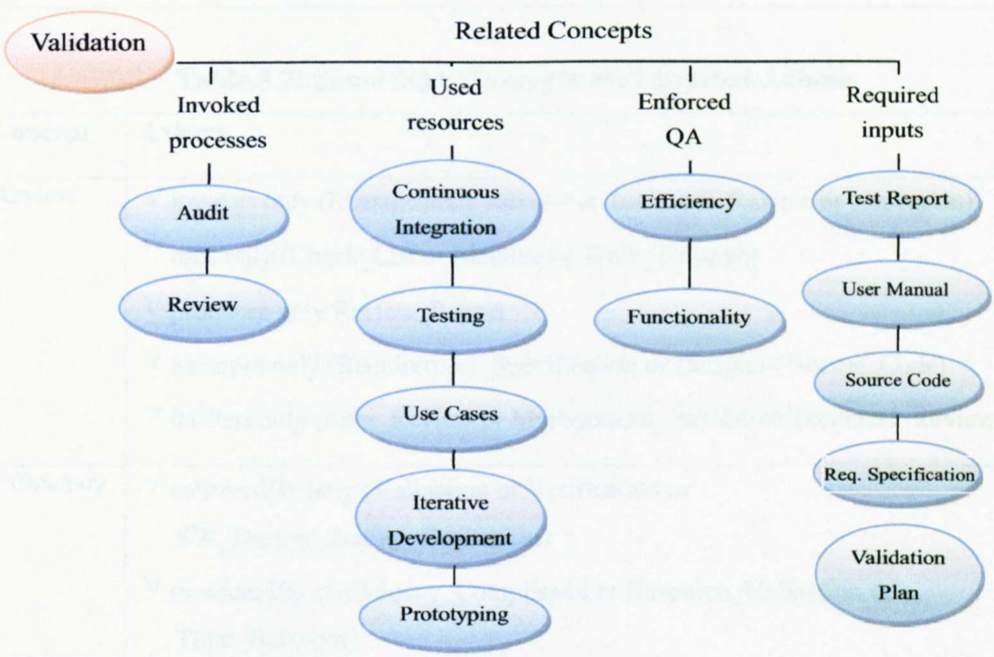


Figure 5.4: Related Concepts to "Validation"

To prevent such situation, ontology axioms (Sec 2.3) were added to the proposed model. By referring back to our example related to *Validation* concept and according to ISO/IEC 9126 standard, a *Validation* process produces *TestReport* and *ValidationPlan* and requires *RequirementSpecification, Source Code, Test Report* and *User Manual* as inputs. In addition, *Validation* has *Efficiency* and *Functionality* as quality attributes and uses *Use-Cases, Prototyping,* and *Measurement* as resources. The above knowledge can be represented with the following axioms added to the *Validation* concept of the SQA ontology model while Table 5.7 shows examples of other SQA concepts and corresponding axioms and *Appendix B* represents the remaining axioms of the SQA ontology:

```
∀produces only (Test_Report or Validation_Plan)
∀invokes only (Review or Audit)
∀ensuresQA only (Efficiency or Functionality)
∀uses only (Use_case or Measurement or Prototyping)
∀hasInput only (Requirement_Specification or Source_Code or
Test_Report or User_manual)
```

Table 5.7: Some SQA Concepts with Related Axioms

Concept	Axioms
Review	∀ invokes only (Management_Review or Technical_Review or Inspection) ∀ uses only (Check_List or Meeting or Walk_Through) ∀ produces only Review_Report ∀ hasInput only (Requirement_Specification or Design or Source_Code) ∀ hasPart only (Joint_Review or Management_Review or Technical_Review)
Efficiency	∀ enforcedBy only (Validation or Verification or SW_Design_Quality_Evaluation) ∀ measuredBy (Efficiency_Compliance or Resource_Utilization or Time_Behavior)
Failure Avoidance	∀ conductedUsing only (Joint_Review or Qualification_Testing or Validation or Verification) ∀ isMeasurementMetricOf only (Fault_Tolerance) ∀ hasMeasurementMetricInput only (Requirement_Specification or Review_Report or Test_Report)

5.3 Implementation of SQA Ontology

The Semantic Web is built on XML and RDF's approach to representing data. The level above RDF is the web ontology language OWL that can formally describes the meaning of terminology used in Web documents in a machine processable respresntation.

In this section the proposed conceptual model resulted from section 5.2 is transformed into formal OWL ontology. As illustrated in Fig. 5.5, the Protégé editing tool is used to translate the SQA conceptual model into machine processable ontology represented in OWL language. The Jambalay tab, a Protégé plug in used for ontology visualization generates graphical representation of the ontology. More over, the Protégé checker is used to verify the ontology concisence while the Racer Pro reasoner is used as a Protégé plug in to check the consistency of the developed ontology.

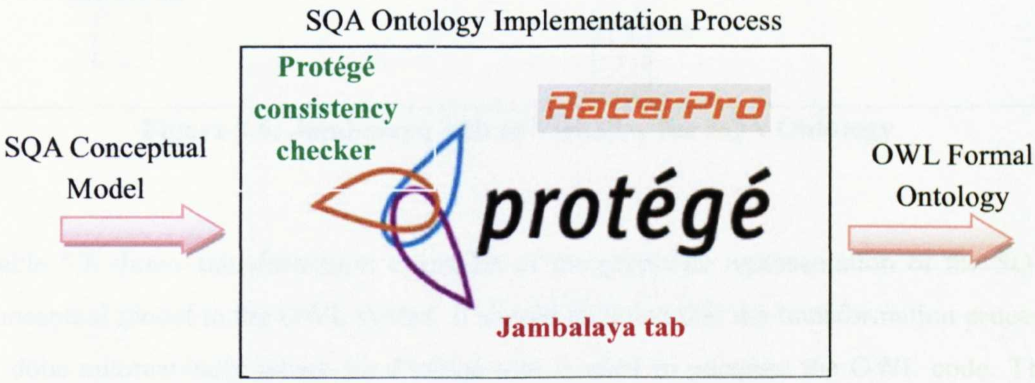


Figure 5.5: From Conceptual Model to OWL Ontology

A top level of the SQA ontology as displayed by the Jambalaya tab is shown in Fig. 5.6 where the property measures with its domain and range is highlighted while Fig.5.7 is screenshot of the SQA ontology edited with Protégé.

Fig. 5.8 shows a class hierarchy of the software quality domain ontology. The figure shows classes and individuals of the SQA ontology where blue arrows represent the subclass relationships and the red arrows represent individuals of the class.

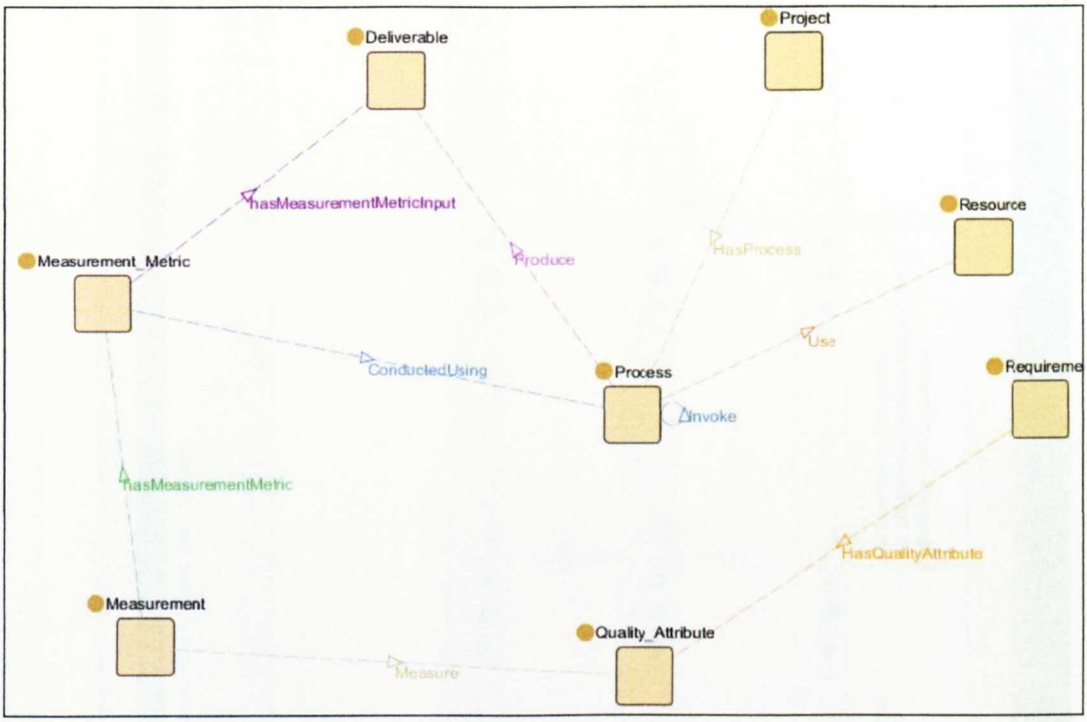


Figure 5.6: Jambalaya Tab to Visualize the SQA Ontology

Table 5.8 shows transformation examples of the graphical representation of the SQA conceptual model to the OWL syntax. It should be noted that the transformation process is done automatically where the Protégé tool is used to generate the OWL code. The OWL description of the software quality ontology generated by Protégé is presented in *Appendix C*.

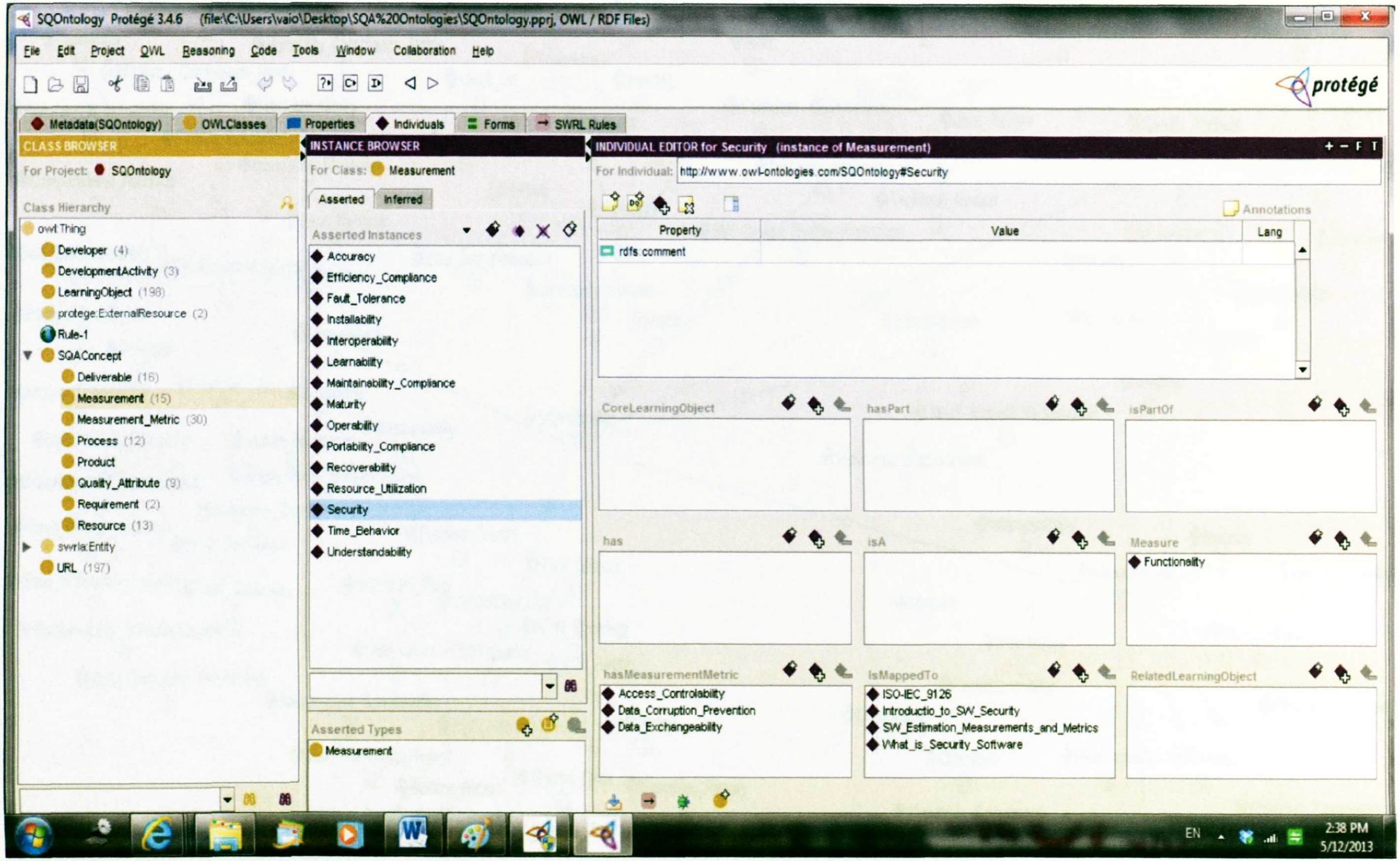


Figure 5.7: The SQA Ontology as Displayed in Protégé

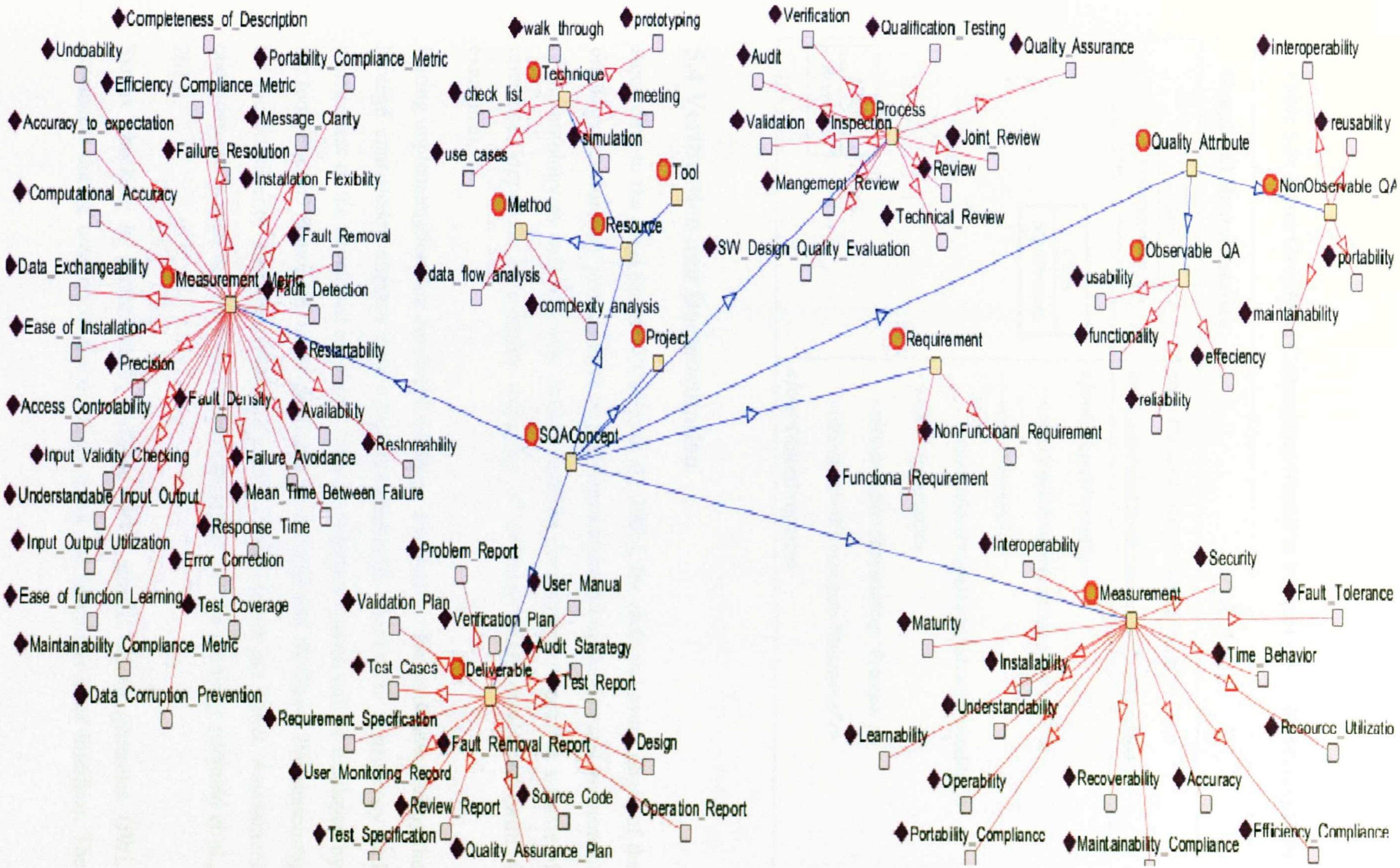
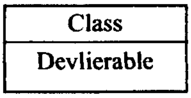
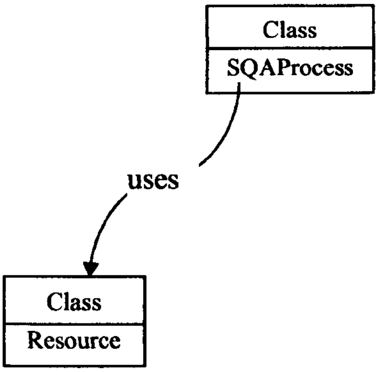


Figure 5.8: Class Hierarchy of the SQA Ontology

Table 5.8: From Graphical Conceptual Model to Formal OWL Representation

Graphical Representation	OWL Code
	<owl:Class rdf:about="Deliverable"> <rdfs:subClassOf rdf:resource="SQAConcept"/>
	</owl:ObjectProperty> <owl:ObjectProperty rdf:about="uses"> <owl:inverseOf> <owl:ObjectProperty rdf:about="usedBy"/> </owl:inverseOf> <rdfs:domain rdf:resource="Process"/> <rdfs:range rdf:resource="Resource"/> </owl:ObjectProperty>

5.4 Verification and Documentation

According to the good practice (Calero et al., 2006), for each and every phase of the ontology development process must be performed technical evaluation and assessment of the ontology as well as a new version must be documented to provide a frame of reference. *Appendix D* contains examples of evolving SQA ontologies (with 4 examples).

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. Fig. 5.9 shows the result generated by Protégé and the Racer Pro reasoning for the consistency checking where no inconsistence classes are listed. Assessment questionnaire is used to verify the logical concistency of the ontology (Bajnaid et al., 2013).

Syntax checking is performed by Protégé OWL plugin which generates OWL statements during creation of the ontology using the Graphical User Interface. The plugin ensures that the generated OWL statements adhere to the rules of the OWL language.

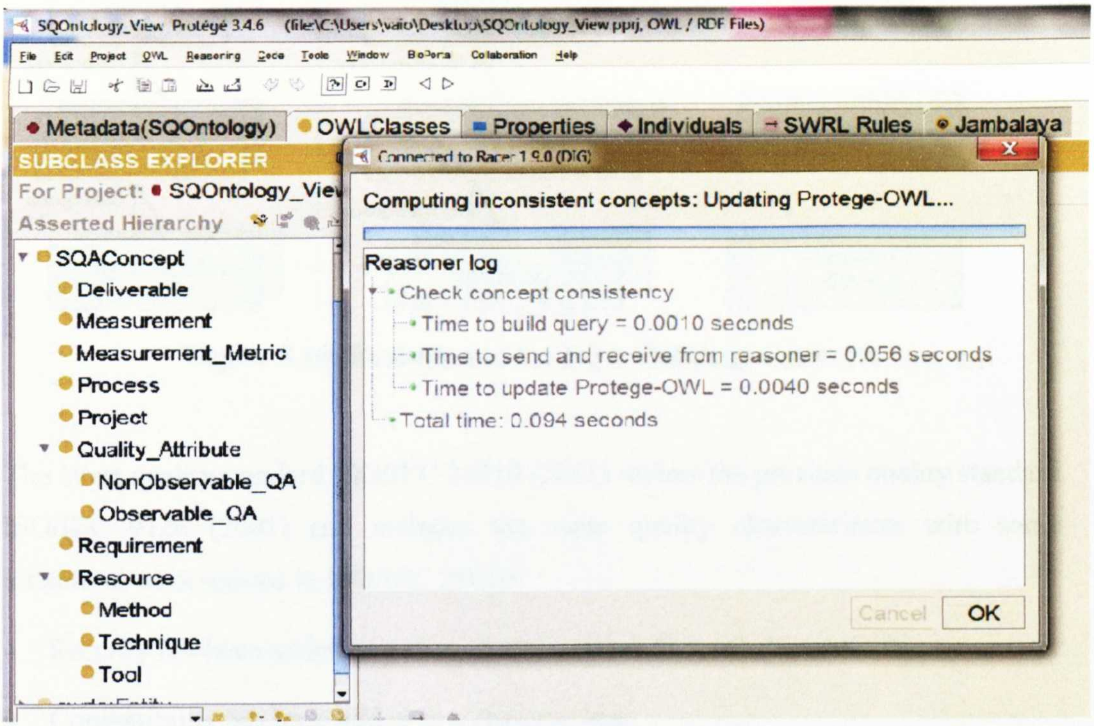


Figure 5.9: Protégé Consistency Checking Result for SQA Concepts as a Whole

In addition, the visualization tab (another Protégé plugin), enables a view of the graph representation of the ontology to ensure the ontology is consistent with the conceptual model (Fig. 5.3).

A detailed evaluation of the developed SQA ontology is presented in Chapter 6.

5.5 An Enhanced Version of the SQA Ontology

Based on the results and findings of the ontology evaluation process (Section 6.4.2), enhanced version of the ontology is developed. In the new version, the ontology concepts "Quality Attribute" and "Measurement" are renamed "Quality Characteristic" and "Quality Sub-characteristic" respectively. The concept "Measurement Metric" is also renamed "Measure" to follow the last quality standard ISO/IEC 25010 (2011) as illustrated in Fig 5.10.

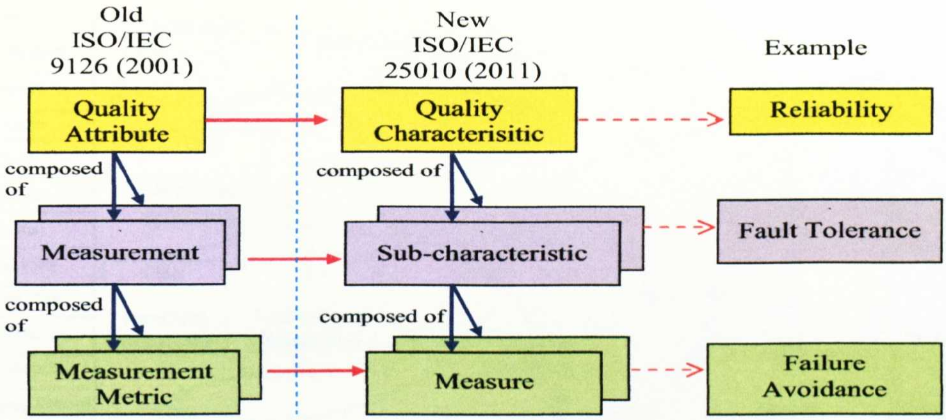


Figure 5.10: Evolution of the SQA Ontology Concepts

The latest quality standard ISO/IEC 25010 (2011) revises the previous quality standard ISO/IEC 9126 (2001) and includes the same quality characteristics with some alterations as described in ISO/IEC 25010:

- Security has been added as a characteristic rather than subcharacterisitics.
- Compatibility has been added as a characteristic.
- New sub-characteristics such as: functional completeness, capacity, user error protection, accessibility, availability, modularity and reusability have been added to existing quality characteristics.
- Compliance with standards and regulations were a subcharacterisitics in ISO/IEC 9126 and now it is outside the scope of the quality model in ISO/IEC 25010.
- Several characteristics and sub-characteristics have been given more accurate names.

Additionally to what is presented in Fig. 5.10, *Appendix E* shows a comparison between the quality characteristics and sub-characteristics in the two standards as adopted from the ISO/IEC 25010 (2011) which is used in addition to the ISO/IEC 25023 (2011) standard for development of a new enhanced SQA ontology as illustrated in Fig. 5.11. New names of quality charactersistics and sub-characerstics are reflected in the enhanced version of the SQA ontology and are shown in blue.

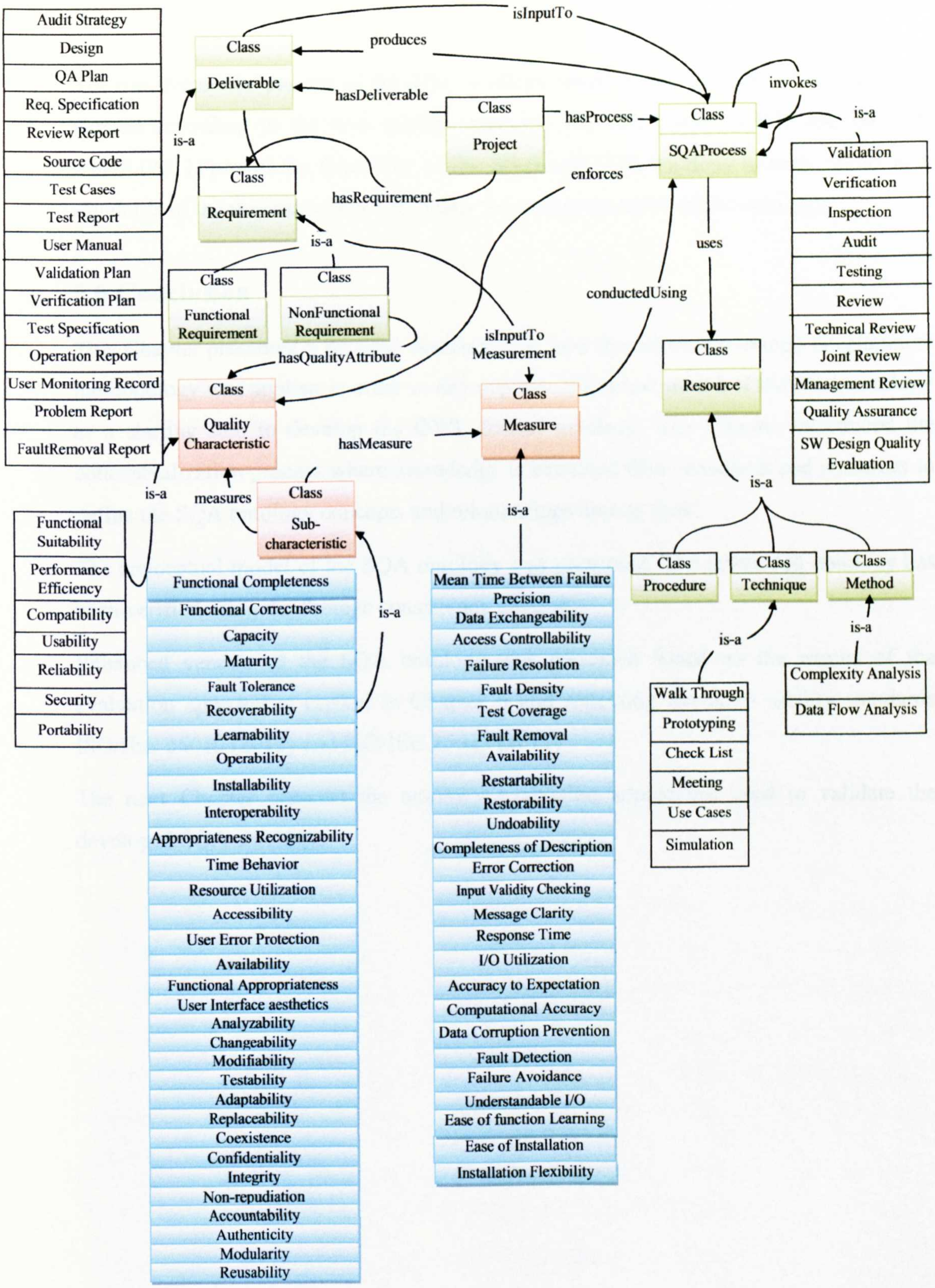


Figure 5.11: Enhanced Version of the SQA Ontology According to ISO/IEC 25010(2011)

The transformation process of the SQA ontology based on the ISO/IEC 9126 to the new version according to the new quality standards ISO/IEC 25010 (2011) and ISO/IEC 25023 (2011) proves the flexability of the developed SQA ontology to easly reflect new standards of the domain without affecting the current semantic of the ontology.

5.6 Conclusion

This Chapter presented a detailed description of how the selected ontology development methodology was applied in order to develop the conceptual model of the SQA ontology as a starting step to develop the OWL formal ontology. The Chapter introduced the conceptualization process where knowledge is extracted from standards and resources to define the SQA ontology concepts and relationships among them.

The conceptual model of the SQA ontology was presented. The developed ontology has been verified using the Protégé consistency checker.

Enhanced version of the SQA ontology was presented based on the results of the evaluation approaches carried in Chapter 6 and reflecting the latest quality standards ISO/IEC 25010 (2011) and ISO/IEC 25023 (2011).

The next Chapter presents the ontology evaluation approaches used to validate the developed SQA ontology.

Chapter 6: Evaluation of the SQA Ontology

Ontology evaluation is an important step followed its development which includes assessing the usefulness of the ontology for the purpose it was built for and evaluating the quality of the ontology (its conceptual coverage, clearness, etc.). This Chapter presents in details the different methodologies applied in this research in evaluating the SQA ontology. This thesis does not claim that the developed SQA ontology is a complete one. It is a version that meant to evolve and aims to model core and main concepts and knowledge of the SQA domain into a practical, sharable and extensible ontology.

6.1 Introduction to Ontology Evaluation

Before publishing ontology or building a software application that relies on ontologies, there is a need for evaluation of the ontology contents (its concepts definitions, taxonomy and axioms). Evaluating ontology is not an evidence of the absence of problems, but it will make its use safer. The main efforts towards evaluating ontology content were made by Gómez-Pérez (1996; 2001) in the framework of METHONTOLOGY and by Welty and Guarino (2001) with the OntoClean method. A survey on evaluation methods and tools can be found in (Gómez-Pérez et al., 2004).

Vrandečić (2009) argues that ontology evaluation is important and worthwhile task. Mistakes and omissions in ontologies can lead to applications not realizing the potential of exchanging data. In addition, ontology evaluation increases the availability and thus reusability of the ontology and decreases maintenance costs. Ontology evaluation assesses the quality of the ontologies and thus encourages their publication and reusability since the confidence of the re-users in the quality of these ontologies increases.

According to (Gómez-Pérez et al., 2004) ontology evaluation requires:

- ***Verification*** which refers to building the ontology correctly;

- ***Validation*** which refers to whether the ontology definitions really model the domain for which the ontology was created. Ontology validation ensures that the correct ontology was built. The goal is to show that the world model is compliant with the formal model;
- ***Assessment*** which focuses on judging the ontology from users' points of view (human judgment).

A common approach is to evaluate the ontology according to a set of ontology design principles and criteria as it was evaluated in (Gruber, 1995; Gomez-Pérez, 2001;Obrst, 2007; Vrandečić, 2009):

- its coverage of the modelled domain;
- the application and data sources it was developed to address;
- its completeness and consistency;
- the structure, syntax and vocabulary; and the representation language in which it is modelled.

The above principles have been used to guide development of the developed SQA ontology. Also it is important to leave some representational choices (such as concepts roles, relations, and constraints) open so that they can be made later based on the actual need of the problem solving or application.

This Chapter is focusing on SQA ontology evaluation using various approaches generally accepted in Software Engineering area. In this thesis ontology evaluation is limited to the criteria identified by Gómez-Pérez (2001) such as: completeness, consistency, conciseness, and expandability.

Completeness: all knowledge that is expected to be in the ontology is either explicitly stated in it or can be inferred. In other words, how well the ontology covers the real world (software quality assurance in our case). Completeness comply to the minimal ontology commitment criteria where the ontology does **not** intend to describe **all** the knowledge involoved in a domain, but only the one that is essential to conceptualize the domain.

Consistency: refers to the absence (or not) of contradictory information in the ontology

Conciseness: if 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.

In this thesis we distinguish between two types of consistency: formal consistency and logical consistency. Verification was held during the ontology implementation (Section5.4) where the SQA ontology was checked for formal consistency. Therefore in this Chapter by consistency we refer to logical consistency.

6.2 Selection of Evaluation Methods

Different ontology evaluation approaches have been considered in literature depending on the purpose of the evaluation and the type of the ontology being evaluated. Brank and colleagues (2005) classify ontology evaluation approaches as following:

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 or application-based ontology evaluation;
3. Those involving comparison with a source of data (e.g. a collection of documents) about the domain to be modelled by the ontology;
4. Those where evaluation is done by humans who try to assess how well the ontology meets a set of predefined criteria, standards, requirements, etc.

The first approach is not applicable due to the lack of a "golden standard" or upper-level (Section 2.2) Software Engineering ontology.

The second approach has been adopted and an application-based ontology evaluation was conducted using a prototype system which was implemented for this purpose.

The third approach was held during development of the ontology when the evolving conceptual model (finalized in Fig. 5.3) was compared to the sources of knowledge.

The fourth approach included usage of the ontology assessment questionnaire which was distributed among SE specialists to evaluate the quality of the ontology.

The applied approaches are detailed in the following sections.

6.3 Validating the SQA Ontology

Recall that the goal of validating the ontology is to show that the world model is compliant with the formal model, i.e. the formal OWL representation of the ontology compliant with the defined conceptual model. Figures 6.1 shows the top level of the SQA concepts as generated by the Jambalaya tab, a Protégé plugin used for ontology visualization. The figure represents the main SQA concepts as in the conceptual model (Fig. 5.3).

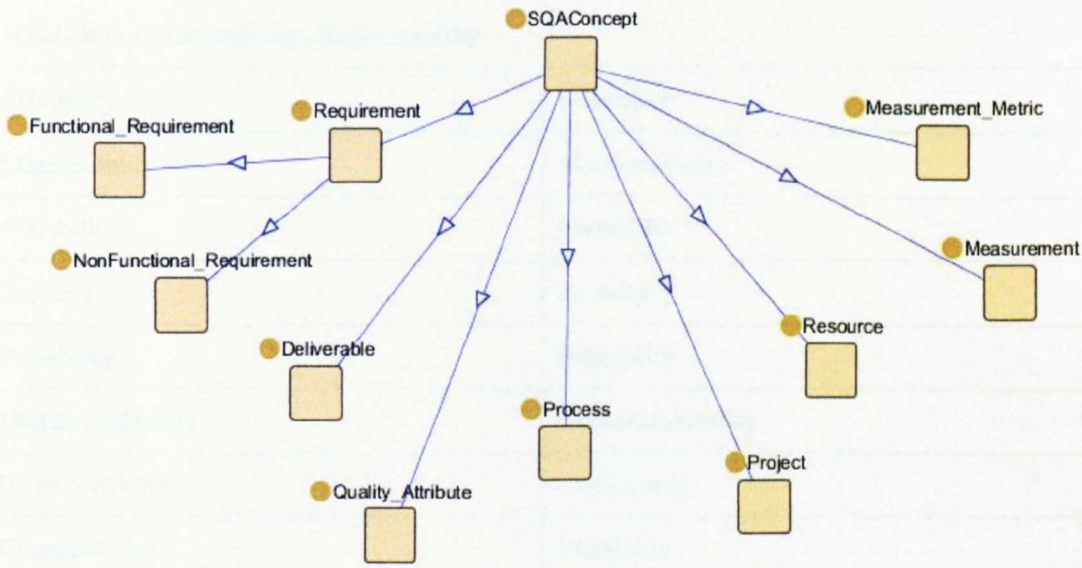


Figure6.1: The Top Level of the SQA Ontology

An ontology evaluation approach is to measure the correspondence between textual sources and the target ontology. The developer of the SWPQA ontology (Sahman, 2008) claims that the ontology covers 80% of the studied domain and can be used as a common agreement of SWPQAs pool of knowledge and can provide a base to evaluate any related presented semantic for one of the studied attributes. In this research, the SWPQA framework (Section 3.3.6) was used to measure its correspondence with the extracted SQA concepts, the quality attributes and measurements in particular. Table 6.1 shows the SQA concepts and their correspondences in the SWPQA frameworks.

Table 6.1: Correspondence of the SQA Concepts and the SWPQA Concepts

SQA concept or term	**Correspondence SWPQA concept**
Quality attribute	Attribute
Accuracy	Accuracy
Stability	Stability
Testability	Testability
Usability	Usability
Recoverability, Learnability, Operability Installability, Analyzability, Replaceability	Could be mapped to the Abililty concept
Efficiency	Efficiency
Maniainabilty	Maintainability
Portability	Portability
Security	Security
Reliability	Reliability
Understandability	Understandability
Error correction	Correctness
Changeability Adaptability Installation flexibility	Flexibility
Interoperability	Interoperability
Availability	Availability

The SWPQA concepts were partially published in (Kayed et al., 2009) where 75% of the SQA ontology's quality attributes and 58.3% of the SQA measurments can be mapped to SWPQA concepts.

A complete framework of the SWPQA (Sahman, 2008) covers 100% of the SQA quality attributes and 91.6% of the SQA measurements concepts.

Ontology development is an evolving process and there is no single ontology to model a domain it is difficult to decide the preciseness of mapping the SQA ontology to other exisiting ones and as ontology evaluation is not a mature research area, in this research we tried to use evaluation approaches that are applicable to our case. Hence, this confirms that the research area is still developing and required further research.

6.4 Assessing the Quality of the SQA Ontology

Ontology assessment was conducted by judging the ontology content from SE specialists' point of view. An introductory document (*Appendix F*) of the SQA ontology with graphical representation of the conceptual model was introduced to the participants with the questionnaire (*Appendix G*).

6.4.1 SQA Ontology Assessment Questionnaire Design

Conceptual model supports clarity where the graphical representation is easier to understand and use (Kablain, 2007). The use of the conceptual model ease the assessment process in this research where the domain specialists can validate wither the model matches the purpose it was built for. The conceptual model (Fig. 5.3) with a link to the questionnaire in Survey Monkey, a free widely used online survey tool (available at: http://www.surveymonkey.com), has been sent to domain specialists inviting them to participate in the SQA ontology assessment process to verify its coverage of the SQA domain, structure, clarity, and extendibility.

The ontology assessment questionnaire designed into four parts:

Part I contains closed questions about the respondent such as experience in the SQA and ontology domains, involvement in teaching the SQA domain and the respondent opinion in the usefulness of using ontologies in teaching SQA.

Part II consists of 7 closed questions with a scale of 1-5, where 5 = strongly agree and 1 = strongly disagree, to validate the following criteria (Gruber, 1995; Gomez-Pérez, 2001; Obrst, 2007; Vrandečić, 2009):

Completeness: the model covers major concepts of the domain;

Structure: the taxonomy and relationships are represented correctly in the model;

Clarity: the model is free from unnecessary and redundant concepts;

Consistency: the model is free from explicitly or implicitly contradictory knowledge;

Expandability: new knowledge can be added to the model without altering the existing semantic.

Parts III and IV consist of open questions about the respondent suggestions of non-relevant concepts to be removed from the model and missing concepts to be added to the model respectively.

6.4.2 Statistical Results and Analysis of the Assessment Questionnaire

Collecting responses from domain experts was a challenge step due to the limited number of experts in the SE domain and in SQA in particular. It took more than 7 months to get 16 of responses only. The problem of limited number of participants faces many researchers in their ontology evaluation process (Alyahya, 2006; García et al., 2006). Although the sample is small it is considered acceptable to judge domain ontology. Table 6.2 shows the respondents' expertise in SQA and ontology domains while Table 6.3 shows the respondents' involvement in teaching SE and Table 6.4 summarises the respondents' agreements on the usefulness of using ontologies in teaching SQA.

Among the 16 respondents 68.8% were involved in teaching software engineering while 31.3% of them have not been involved in such teaching. 64.7% of the respondents agree that ontology can be useful in teaching SQA and 11.8% strongly agree while 25% have borderline opinion.

Table 6.2: Respondents' Expertise in SQA and Ontology Domain

Respondent's expertise	Null	Poor	Average	Above average	Excellent
SQA domain	0	1	4	7	4
Ontology domain	0	1	10	4	1

Table 6.3: Respondents' Involvements in Teaching SE

Statement	Yes	No
Are you now (or ever been) involved into the teaching of Software Engineering?	11	5

Table 6.4: Respondents' Agreements on Using Ontologies in Teaching SQA

Statement	Strongly disagree	Disagree	Borderline	Agree	Strongly agree
Do you think ontology can be useful for teaching SQA?	0	0	4	10	2

Responses on statements relevant to the assessment of the conceptual coverage of the SQA model (Part II of the questionnaire) as shown in Survey Monkey is illustrated in Table 6.5 while the respondents' comments and suggestions of Parts (III and IV) of the questionnaire are shown in *Appendix H.*

The results of the survey are presented below where an enhanced version of the ontology is being developed to reflect the main suggestions from the questionnaire:

Completeness: Majority of the participants (81.3%) agreed that the ontology developed in this research covers major concepts of the SQA domain. 15.4% of them strongly agree and none of the respondents disagree with the completeness of the ontology. However, an important suggestion to add testing related concepts (black and white box, system and unit test...etc.) was made. Though, the current ontology is not heavily focused on testing techniques, it is worth investigating this ontology 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 SE ontology.

Structure: A reasonable majority of the respondents (62.5%) agreed with the ontology taxonomy as is, with no 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, just around the mean (3.13). However, we believe that this reasonably good result 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 that will be adopted in the enhanced version is to use the terms Quality_Characteristic and Sub-characteristic instead of Quality_Attribute and Measurements respectively. We can also replace the term Measurement_Metric with the term Measure as per the latest quality standard ISO/IEC 25010 (2011).

Consistency: A reasonable majority of the responses (68.8%) agreed that the developed ontology is consistent where 27.3% of them strongly agreed on its consistency. Ontology formal-consistency was verified using the Protégé consistency checker plugin.

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. Suggestions to include agile terminology with new quality measurements and metrics (as in ISO/IEC 25010) will be considered as extensions in the enhanced version of the ontology.

Table 6.5: Respondent's View on the SQA Ontology as Shown in Survey Monkey

	1	2	3	4	5	Rating Average	Rating Count
Completeness: The model covered the major concepts of the SQA domain	0.0% (0)	0.0% (0)	18.8% (3)	68.8% (11)	12.5% (2)	3.94	16
Structure: The taxonomy ("is-a" relationship) is presented correctly in the model	0.0% (0)	0.0% (0)	37.5% (6)	37.5% (6)	25.0% (4)	3.88	16
Structure: Other relationships among the SQA concepts are presented correctly in the model (invokes, produces, measures, uses, hasQualityAttributes…etc)	0.0% (0)	0.0% (0)	37.5% (6)	43.8% (7)	18.8% (3)	3.81	16
Clarity: There are some redundant concepts in the model	6.3% (1)	12.5% (2)	50.0% (8)	25.0% (4)	6.3% (1)	3.13	16
Clarity: There are some ambiguous concepts in the model	12.5% (2)	12.5% (2)	37.5% (6)	25.0% (4)	12.5% (2)	3.13	16
Consistency: The model is logically consistent. Ex: x instance of classes A and B, but A and B are disjoint. This is a contradiction	0.0% (0)	0.0% (0)	31.3% (5)	50.0% (8)	18.8% (3)	3.88	16
Extendibility: New terms can be introduced without the need to revise existing structure of the model	0.0% (0)	12.5% (2)	12.5% (2)	43.8% (7)	31.3% (5)	3.94	16

Although, there is no such a single ontology that can unanimously represent any knowledge area, especially for an evolving domain like SQA, the survey shows a high level of agreement around the major assessment criteria. This is despite the fact that each participant responds based on their own view, background and context.

Participants' responses to Part II of the assessment questionnaire are illustrated in Figure 6.2. Responses of participants who are considered to be expert in the field and those with average expertise are represented in figures 6.3 and 6.4 respectively.

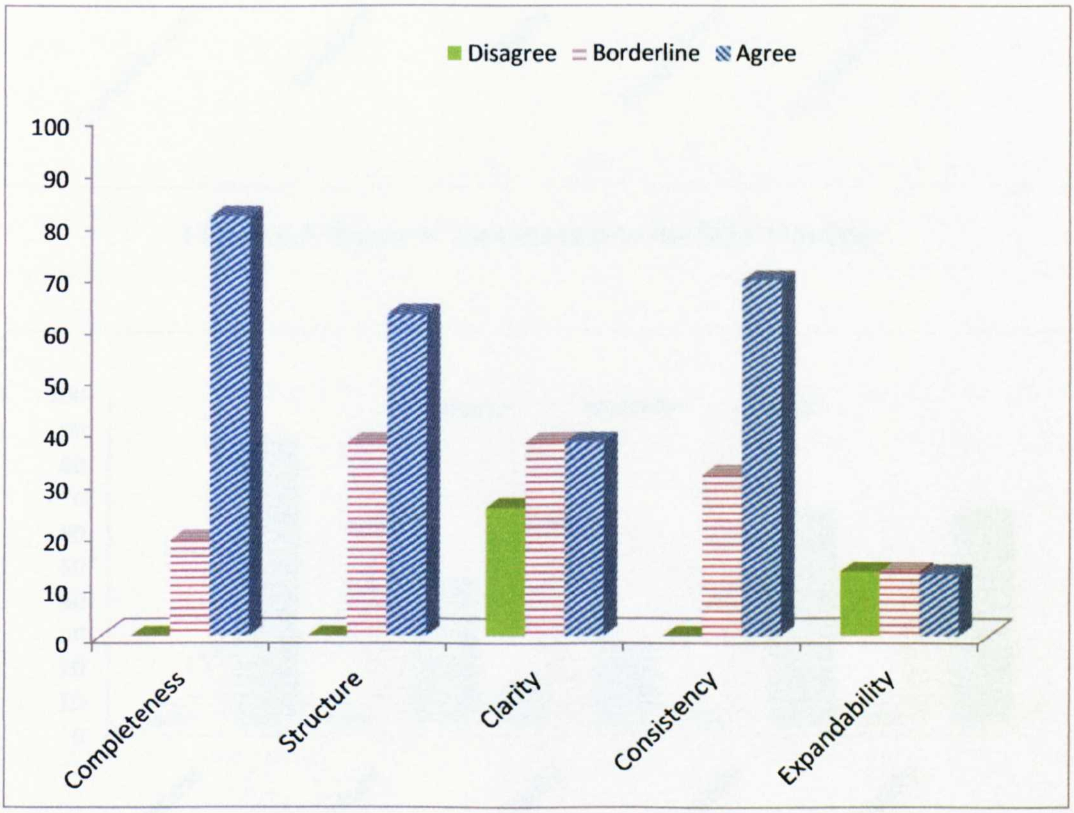


Figure 6.2: Participants' Assessments of the SQA Ontology

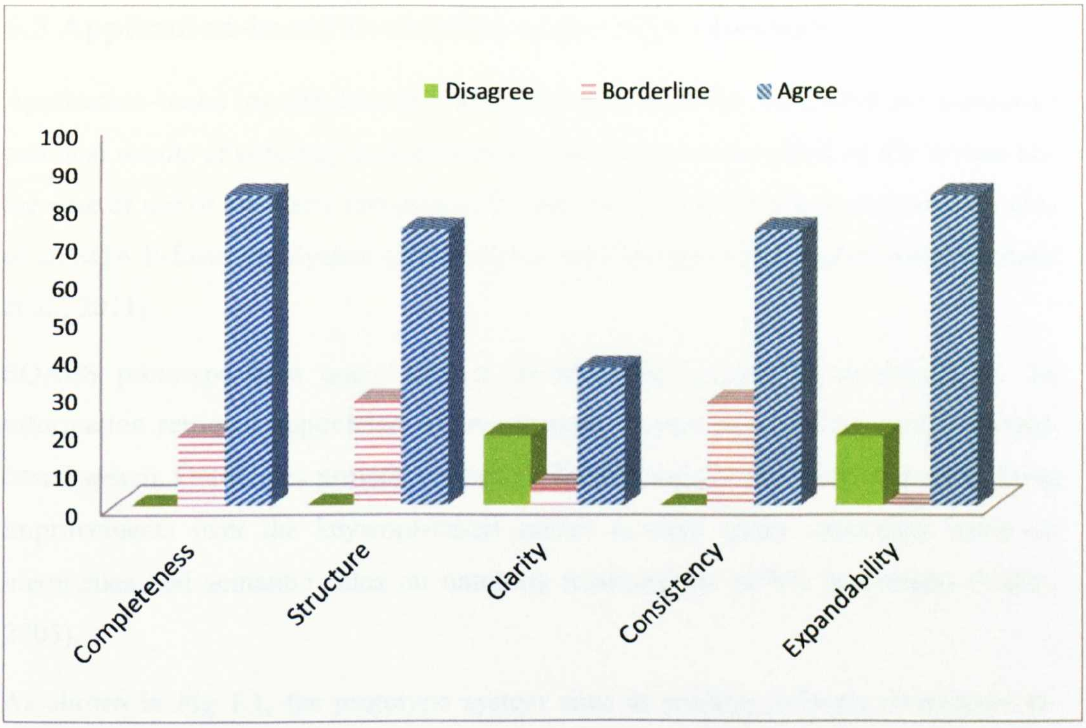


Figure 6.3: Experts' Assessments to the SQA Ontology

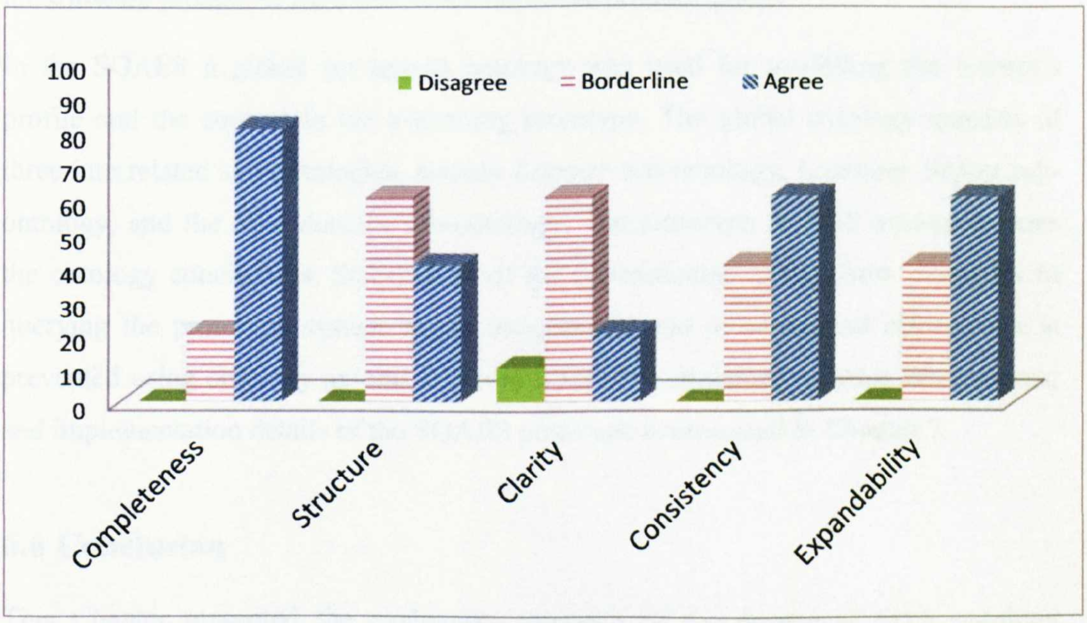


Figure 6.4: Assessments of Participants with Average Experience in the Domain

6.5 Application-based Evaluation of the SQA Ontology

Application-based (or task-based) evaluations offer a useful framework for measuring practical results of ontology conciseness such as responses provided by the system and the ease of use of the query component (Obrst, 2007). A querying prototype consisting of an SQA E-Learning System (SQAES) has been designed and implemented (Bajnaid et al., 2011).

SQAES prototype is a query tool to evaluate the impact of ontologies on the information retrieval application where semantic search is combined with keyword-based search. Ontologies provide controlled vocabularies of the domain that can bring improvements over the keyword-based search through query expansion based on hierarchies and semantic rules on ontology relationships (OWL properties) (Vallet, 2005).

As shown in Fig 1.1, the prototype system aims at guiding software developers (e-learning in the workplace) or student (in traditional learning scenario) through the necessary QA practices by providing resources that deal with SQA related aspects of the software process in hand and hence improves product quality.

In the SQAES a global (or upper) ontology was used for modelling the learner's profile and the context in the e-learning prototype. The global ontology consists of three interrelated sub-ontologies, namely *Learner* sub-ontology, *Learning Object* sub-ontology, and the *SQA* domain sub-ontology. The prototype SQAES system ensures the ontology conciseness. Screenshots of the experimental results show examples of querying the prototype system where unnecessary and overwhelmed information is prevented using ontology axioms (Section 7.3.2).The structure, software components, and implementation details of the SQAES prototype is presented in Chapter 7.

6.6 Conclusion

This Chapter presented the evaluation approach of the developed SQA ontology model. The ontology development is an iterative process where the ontology was verified during implementation as described in Section 5.4. The evaluation methodology includes assessing the quality of the developed ontology and evaluating the ontology for the purpose it was built for. The quality of the ontology was validated against several criteria. The consistency and conciseness of the developed ontology were automatically validated during the implementation process using the Protégé consistency checker tool (Bajnaid et al., 2011). Ontology querying e-learning prototype was built to evaluate the SQA ontology conciseness (Bajnaid et al., 2012). Ontology assessment questionnaire was developed to evaluate the quality of the SQA ontology. The discussion and findings of the evaluation was also presented in the Chapter. The next chapter presents in details the general system structure and implementation details of SQAES.

Chapter 7: Ontology-Based e-Learning System: Case Study

As there is no fixed learning path that can fit all learners' needs, most systems in the e-learning literature have combined more than one knowledge source to contextualize the learning sequence and the learning content aiming to provide the best personalized learning experience. In personalized e-learning or context-aware e-learning environment, the system responds differently according to the learner characteristics (i.e. learner's needs, learning style, preferred presentation formats, learner's previous knowledge of the subject domain, etc.) and performance (gathered in user profile) (Gómez-Pérez et al., 2006).

Ontology as a promising approach plays an important role in the development of enhanced and effective learning by providing machine-readable content (Stojanovic et al., 2001; Hatem et al., 2005, Kontopoulos et al., 2007). Unlike the linear organization of textbooks, access to learning resources in an e-learning course using ontologies is structured (see Section 2.7).

In order to evaluate the developed SQA ontology the prototype Software Quality Assurance e-learning System (SQAES) has been developed. SQAES prototype is a query tool to evaluate the impact of using ontologies on the information retrieval where semantic search is combined with keyword-based search. Ontologies provide controlled vocabularies of the domain that can bring improvements over the keyword-based search through query expansion based on hierarchies and semantic rules on ontology relationships (OWL properties) (Vallet, 2005).

This chapter first presents the learning aspects of SQAES (e.g. learning scenario and learner profile). Later the Chapter describes how SQAES is implemented, the overall system architecture with a detailed description of its major software components. It also introduces the design of the global ontology space consisting of the learner, the learning objects, and the domain sub-ontologies.

7.1 The SQAES Prototype

In the current research, SQAES can be used in two learning scenarios: by software developers in workplace learning; and by software engineering student in a traditional learning scenario as illustrated in Fig 7.1.

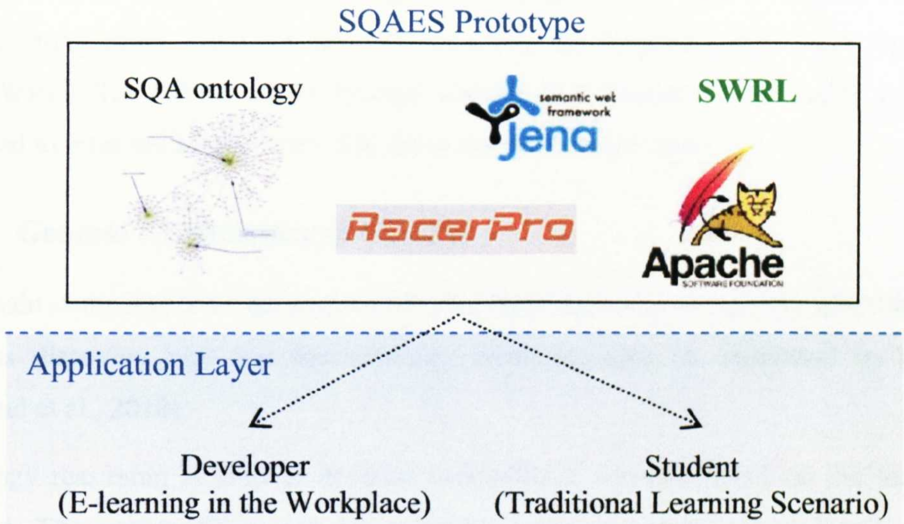


Figure 7.1: Macro View of SQAES

Either it is a developer or a student in this chapter we will use the term learner to describe the suggested scenario.

7.1.1 Requirements to SQAES

Before describing architecture and overall design of the SQAES there is a need to define requirements to such a system. They can be summarized as follows:

- SQAES shall guide learners through the necessary SQA practices by providing resources that deal with all SQA related aspects of the software process at hand.
- This could be achieved by sensing the learner's current activity and suggesting relevant learning resources (e.g. recommendations for good practices, example code, and graphical description of a related methodology/process) that deal with all SQA aspects of the process at hand.

- The system shall be able to determine the learner's current software development context and infer related SQA knowledge by invoking the appropriate reasoning mechanisms.
- Besides the SQA domain ontology and the associated axioms (section 5.2.7), there is a need to define the system's global ontology which shall be augmented with reasoning rules. They can be encoded using the Semantic Web Rule Language (SWRL). The SWRL tab of Protégé and the Jess inference rule engine might be used to infer the needed rules that drive the learning process.

7.1.2 General Architecture of SQAES

The main components of the system are: the learning recommendation generator, the process discovery unit and the ontology reasoning unit as illustrated in Fig.7.2 (Bajnaid et al., 2010).

Ontology reasoning is used to develop personalized services based on the learner's context. The system filters out the available learning objects (LOs) based on the learner's usage profile and guided by related ontology-based reasoning. The output is a set of domain concepts that are directly related to the learner selected query. The extracted query-related concepts are mapped to a set of learning objects which are provided to the learner. Ontological rules are applied to track previously consumed learning objects and dynamically infer implicit knowledge based on the user profile.

Context model is divided into global ontology (upper ontology) and specific ontology (the SQA ontology). The global ontology is a high-level ontology that presents general features of the context. The specific ontology is a domain ontology that captures general concepts and properties of domain knowledge (in this case Software Quality).

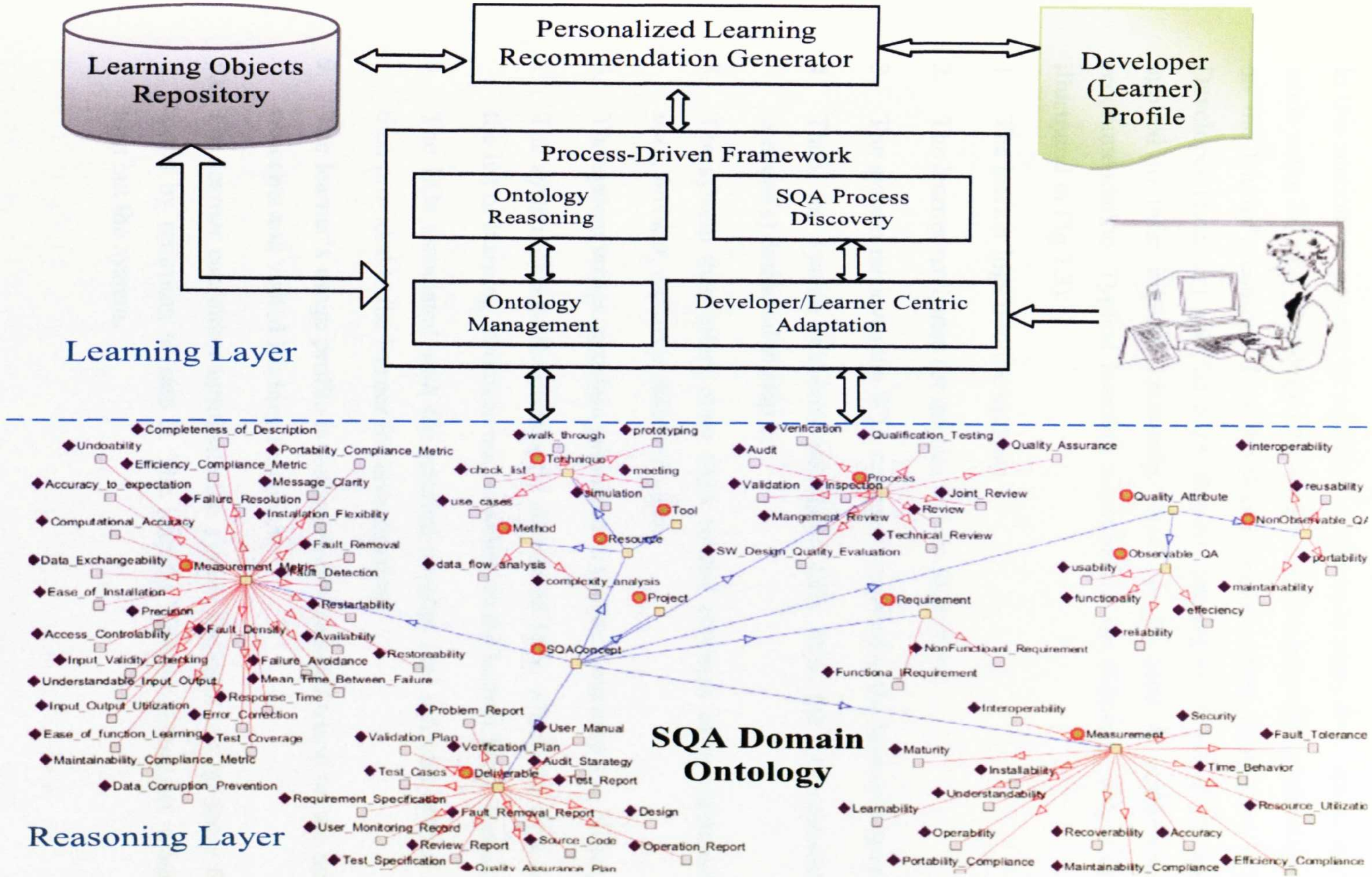


Figure 7.2: SQAES Architecture

7.1.3 Learning Scenario in SQAES

In this section we present an overview of the main steps in a typical learning scenario while using SQAES. Ontology reasoning is used to personalize learning services based on the learner's context. This developer/learner centric adaptation is based on the Developer (Learner) and the SQA domain ontologies. A set of ontological rules is applied to infer implicit knowledge that can be used to customize the learning recommendation. Typical learning scenario has the following sequence of steps (as illustrated in Fig.7.3):

1. The learner logs into the system;
2. The learner navigates (or queries for) an SQA term;
3. The system retrieves the SQA concept(s) related to the learner's queried term;
4. Then, the system retrieves associated LOs from the LO repository using the concept(s) extracted in step 2;
5. The system then infers other SQA related concepts using relationships such as, *uses*, *invokes*, *enforces*, *isInputTo*, etc.;
6. The system writes metadata generated in the previous step to a buffer;
7. The system checks for previously consumed LOs, which are then removed from the list of learning resources but presented to the learner for re-learning;
8. The LOs associated with the queried concept and inferred related concepts are then provided to the learner for investigation;
9. The learner's usage profile is automatically updated based on the newly selected concepts and visited learning resources;
10. The learner can either terminate the system by login out or query for new SQA terms by returning to step 2. The learning activity terminates when the learner logs out the system.

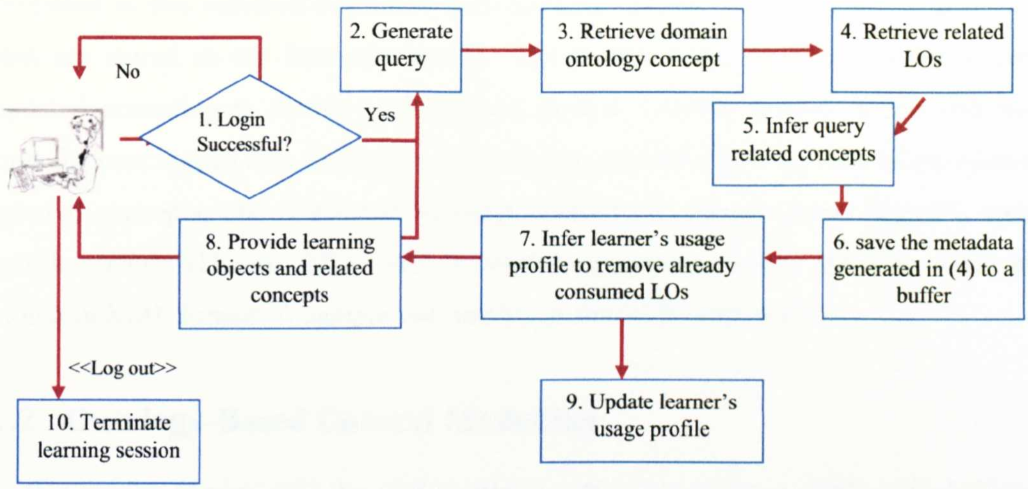


Figure 7.3: A Typical Learning Scenario Processing Steps

For example if the developer/learner queries about the *Validation* process. The system retrieves unconsumed learning objects that are directly associated to the term *Validation* (already consumed LOs are presented for the user for re-learning). The system will then use the reasoning rules, given in step 4, to infer other concepts related to the validation process. For example: a *Validation* process *enforces* quality attributes such as *Functionality* and *Efficiency* and *invokes* the *Review* and *Audit* processes. It also *uses* the *Prototyping* as resources. The system then saves these related concepts in a buffer. Associated LOs and related concepts are then displayed as recommendations to the learner for investigation.

7.1.4 Developer/Leaner Usage Profile

According to Das (2010) context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that are considered relevant to the interaction between a user and an application including the user and the application themselves. Context-aware learning or personalized learning provides learning contents according to learner's needs, preferences, style and previous knowledge of the subject domain. Various context parameters are considered in existing e-learning system such as: learner personnel profile, expertise level, learning preferences, learning situation, network, device, etc. (Das et al., 2010). The system proposed in this research takes into consideration already consumed learning objects that are stored in the learner's profile. The learner usage profile is automatically updated according to his/her performance. A new learning session is initiated each time the user logged into the system. Information about the starting time of the session, queried concepts, and consumed learning resources is stored in the learner's usage profile. JDOM (Hunter, 2008) will be used to manipulate users' profiles which are stored in XML format. A sample user profile is shown in *Appendix I*.

7.2 Ontology-Based Context Modelling

In this section is presented the global ontology that is used for modelling the learning context in the proposed e-learning prototype. The global ontology consists of three interrelated sub-ontologies, namely *Learner* sub-ontology, *Learning Object* sub-ontology, and the *SQA* domain sub-ontology. These sub-ontologies are used to represent the most fundamental context elements for capturing information about any software development scenario undertaken by a learner. Fig.7.4 shows the general structure of the upper ontology among with the relationships to other sub-ontologies in the global ontology space. It should be noted that relationships are represented by arrows where the domain of the relationship is represented by the literal D while the range is represented by literal R in all graphs.

The global ontology space was developed using OWL. Each entity is associated with attributes (defined by *owl:DatatypeProperty*) and related to other entities (defined by *owl:ObjectProperty).* The built-in *owl:subClassOf* property is used for hierarchically structuring sub-class entities. Ontology reasoning techniques are used to enable personalized learning that can be achieved through learner centric adaptation where the learner's implicit knowledge is used to create recommendations.

The *Learner* sub-ontology represents the learner's activity profile which consists of already consumed learning resources. The learner'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 at hand.

The *SQA* domain sub-ontology captures general concepts and properties about the SQA knowledge domain. The main class in this ontology is *SQAConcept* that is used to conceptualize and represent all concepts of the software quality ontology. The property *makeQuery* associates SQA-related keywords entered by the learner to the most relevant concept in the SQA domain sub-ontology. The property *isMappedTo* relates the *SQAConcept* class to the *Learning Object* class. The property *isMappedTo* is used to map LOs metadata to the SQA ontology concepts and thus allow sharing of resources. The property *consumedLearningObject* tracks LOs previously consumed by a specific learner.

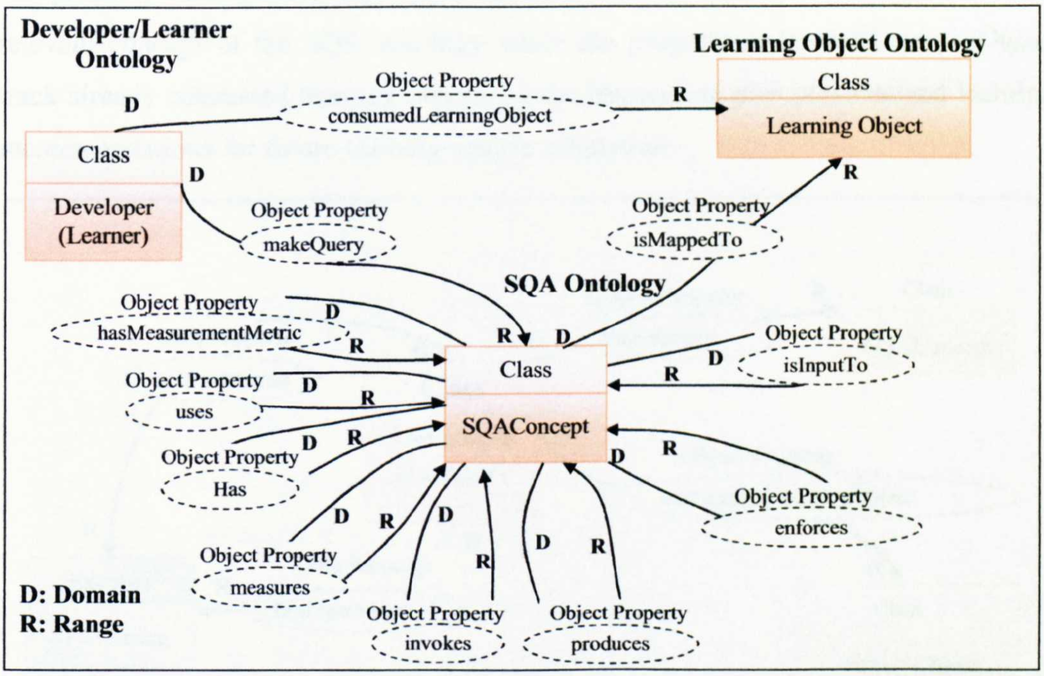


Figure 7.4: Macro View of the Global Ontology

The following subsections describe the *Learner* and the *Learning Object* sub-ontologies respectively while the domain ontology was described in detail in the Chapter 5.

7.2.1 Developer/Learner Ontology

The learner sub-ontology represents the contextual knowledge about the learner that helps the system to adapt and deliver learning recommendations with the most relevant learning objects in response to queries made by the learner. The structure and relationships properties – both data properties and object properties – are illustrated in Fig.7.5.

The properties *hasUserName* and *hasPassword* relate individuals of class *Learner* to their identification and authentication information. In relation to the domain ontology, the property *makeQuaery* associates keyword entered by the learner to the most relevant concept of the SQA ontology while the property *consumedLearningObject* track already consumed learning objects by the learner and plan personalized learning recommendations for future learning centric adaptation.

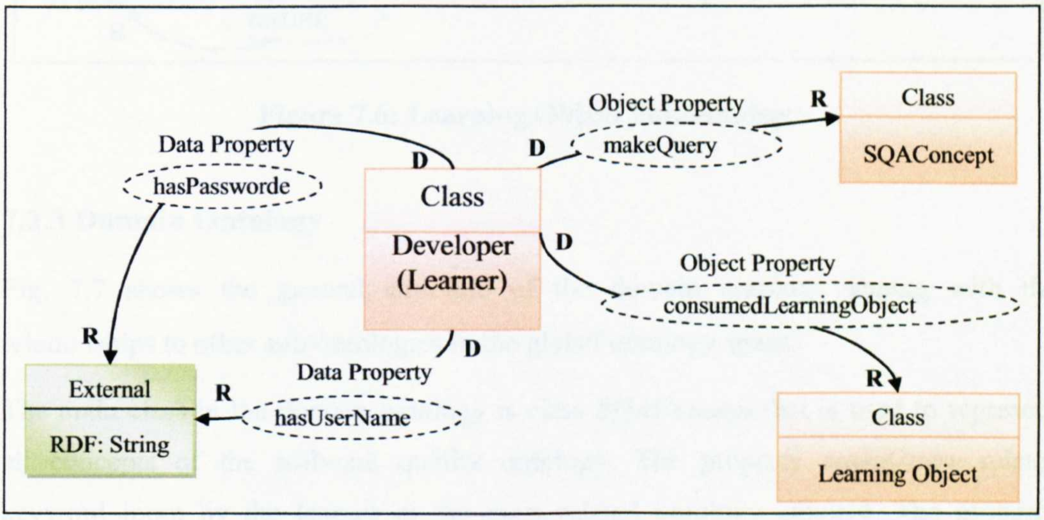


Figure 7.5: Developer/Learner Sub-ontology

7.2.2 Learning Object Ontology

The learning object is a value integrator of a learner's need, knowledge element, or any learner-centric value ingredients. LO is the minimal unit of pedagogically reasonable learning content consists of random content (video, image, text, etc.) (Schmidt and Winterhalter, 2004). Each concept of the SQA domain is associated with some learning objects by the property *isMappedTo*. It should be noted that each

SQA concepts is mapped to multiple LOs, i.e. the property *isMappedTo* (*SQAConcept*, *LO*) has a cardinality of 1..n.

Already consumed learning objects by specific learner are shown by the property *consumedLearningObject*. The property *hasURL* relates an individual of the *Learning Object* class to its corresponding URL. The structure and relationships of this ontology is illustrated in Fig.7.6.

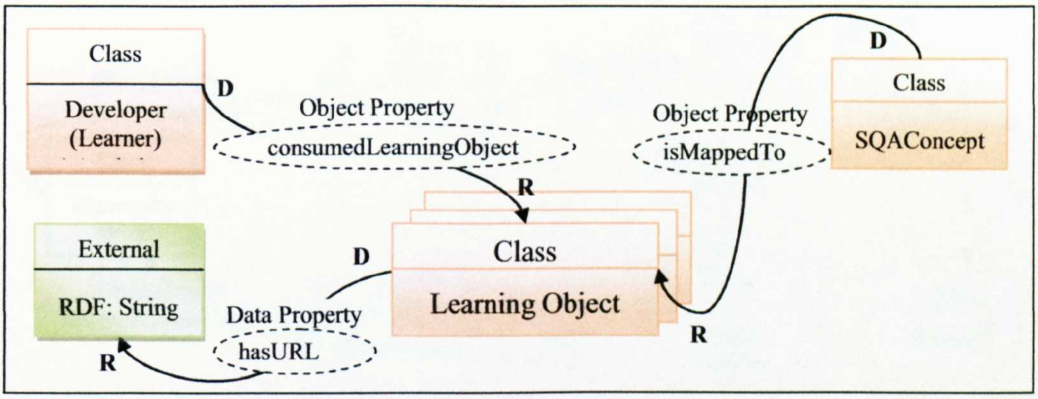


Figure 7.6: Learning Object Sub-ontology

7.2.3 Domain Ontology

Fig. 7.7 shows the general structure of the domain ontology among with the relationships to other sub-ontologies in the global ontology space.

The main class in the domain ontology is class *SQAConcept* that is used to represent all concepts of the software quality ontology. The property *makeQuery* relates keyword input by the learner to the most related ontology concept. The property *consumedLearningObject* track previously consumed LOs by a specific learner. The property *isMappedTo* relates the SQAConcept class to the learning object class. Properties of the SQA domain ontology were described in chapter5.

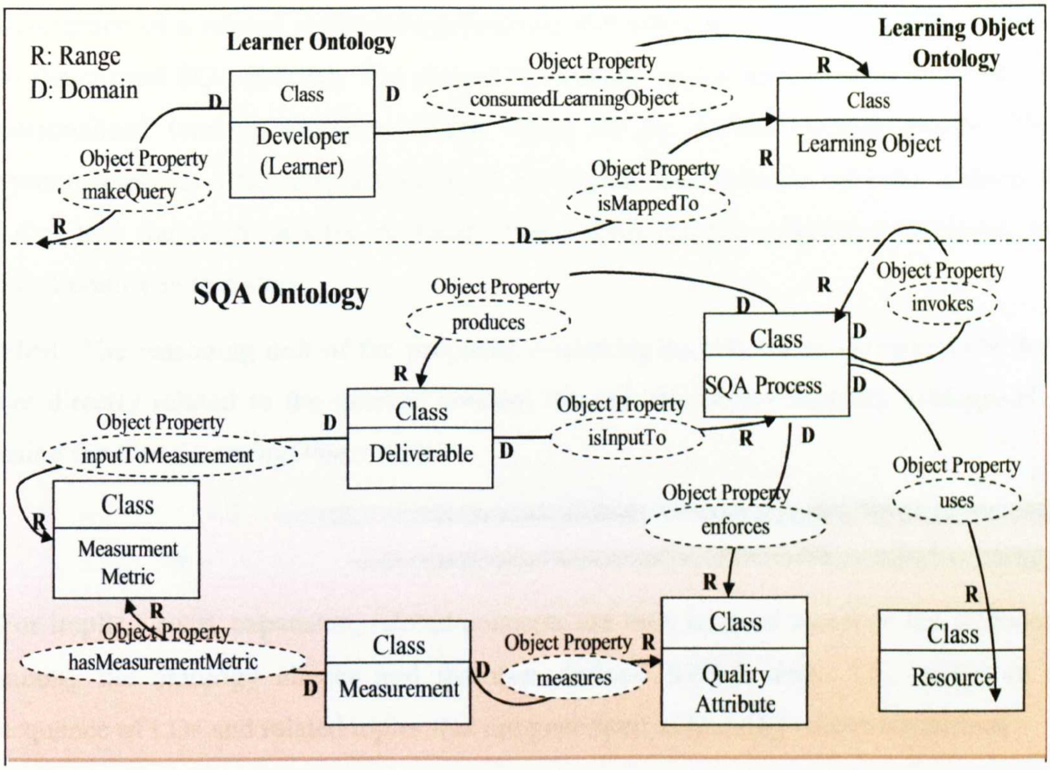


Figure 7.7: SQA Domain Ontology

7.3 Context Reasoning

Ontology reasoning is used to develop personalized services based on the developer's (our target learner) context. This learner centric adaptation is enabled by integrating knowledge components from the three sub-ontologies (learner, learning object, and SQA domain ontology). Many ontological rules are applied to dynamically infer metadata that can be used to customize the learning recommendation (Bajnaid et al., 2010).

7.3.1 Developer/Learner Centric Adaptation

Prototype system aims to guide learner through the necessary SQA practices by providing resources that deal with SQA related aspects of the current SQA process at hand. This is achieved by sensing the learner's current activity and suggesting relevant LOs (e.g. recommendations for good practices, example code, and graphical description of a related methodology/process) that deals with all SQA aspects related to the current SQA process. The aim of the learner centric adaptation is to construct personalized learning recommendation based on the learner's usage profile. The system responds differently according to the learner performance (already consumed LOs) and the SQA process at hand. The learner centric adaptation achieves its functionality in two steps:

First: The reasoning unit of the proposed e-learning system infers the core LOs that are directly related to the queried concept through the object property *isMappedTo* using the *CoreLearningObject* rule:

[redacted]

→ [redacted]

For implicit query expansion, related concepts are then inferred based on the relations among the ontology classes and the user defined SWRL rules. The output is a sequence of LOs and related topics that are generated as learning recommendations.

Second: recommendations generated from the previous step are then semantically refined and adjusted according to the learner's profile where the system distinguishes LOs objects that have already been consumed by the developer.

Besides the OWL ontology reasoning rules (subClassOf, subPropertyOf, inverseOf, etc...), 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. Some of the user defined SWRL rules used to infer related LOs expressed in the first order logic are shown in Table 7.1.

The property *isMappedTo (?C, ?LO)* maps the learner's query related concept to a corresponding learning object. The property ¬ *consumed (?L, ?LOj)* relate a learner to a learning object that has not been consumed so far. It should be noted that the system automatically establishes relation of ¬ *Consumed (?L, ?LOj)* for all those learning objects that have not been consumed by particular learner.

Table 7.1: SWRL Rules for Related Concepts Construction

UsedResourceRule retrieves related software resources (uses cases, prototyping, check list, etc.) that can be used to perform a specific SQA process C: `Learner (?L) ^ makeQuery (?L,?C) ^ SQAProcess (?C) ^ uses (?C,?R) ^ Resource(?R) → RelatedConcept (?C,?R)`
EnforcesRule retrieves all quality attributes that can be used to assess a specific process C: `Learner (?L) ^ makeQuery (?L,?C) ^ enforces (?C,?QA) ^ QualityAttribute (?QA) → RelatedConcept (?C,?QA)`
InvokedProcessRule allows the system to infer all SQA processes that can be invoked by a specific process (C) that is currently under development by the user (i.e. user queried process): `Learner (?L) ^ makeQuery (?L, ?C1) ^ Process (?C1) ^ invokes (?C1, ?C2) ^ Process (?C2) → RelatedConcept (?C1, ?C2)`
IsInputRule retrieves SQA process(es) for which a deliverable C is an input to: `Learner (?L) ^ makeQuery (?L,?C) ^ Deliverable (?C) ^ isInputTo (?C,?P) ^ Process (?P)→ RelatedConcept (?C,?P)`
ProducedDeliverableRule retrieves deliverable(s) that can be produced by a specific process C: `Learner (?L) ^ makeQuery (?L,?C) ^ Process (?C) ^ produces (?C,?D) ^ Deliverable(?D) → RelatedConcept (?C,?D)`
MeasuredByRule retrieves SQA measures that can be used to measure a specific quality attribute C: `Learner (?L) ^ makeQuery (?L,?C) ^ Quality_Attribute(?C) ^ measuredBy (?C,?M) ^ Measurement(?M) → RelatedConcept (?C,?M)`

Table 7.1: Continued

MeasurementMetricRule retrieves measurement metric(s) related to specific SQA measurement C: `Learner (?L) ^ makeQuery (?L,?C) ^ Measurement (?C) ^ hasMeasurementMetric (?C,?M) ^ MeasurementMetric (?M)→ RelatedConcept (?C,?M)`
ConductingProcessRule retrieves SQA process(es) that is associated with specific measurement metric C: `Learner (?L) ^ makeQuery (?L,?C) ^ MeasurementMetric(?C) ^ conductedUsing (?C,?P) ^ Process (?P) → RelatedConcept (?C,?P)`
MeasuringQARule retrieves quality attributes that can be measured by a specific SQA measurement C: `Learner (?L) ^ makeQuery (?L,?C) ^ Measurement (?C) ^ measures (?C,?QA) ^ Quality_Attribute (?QA) → RelatedConcept (?C,?QA)`

7.3.2 SQA Ontology Axioms

The prototype system provides the learner with a recommendation list based on the initial query. However, this list may include some overwhelmed LOs or unnecessary content. Ontology axioms were added to prevent unnecessary knowledge. In ontology representation, axioms (see Section 2.3) can be used to represent the meaning of concepts carefully, and to answer questions on the capability of the built ontology using the ontology concepts.

Consider the case when the user queries the *Validation* concept, which is a process according to the SQA ontology (see Fig. 5.3), 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 overwhelmed or unnecessary contents. In the example of *Validation*, by firing the *Invokes* rule, all SQA processes will be added to the list of recommendation. 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 (Fig. 7.8).

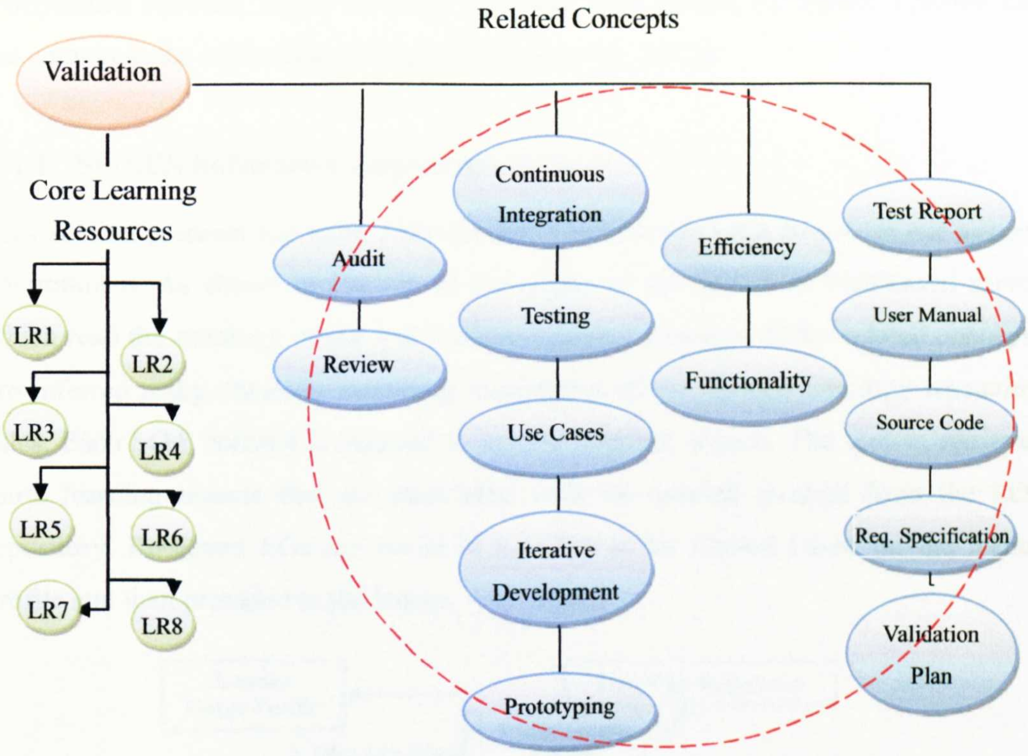


Figure 7.8: Provided LOs for the Concept "Precision"

To prevent such situation, recommendation refining is guaranteed by adding ontology axioms to the ontology model. By referring back to our example related to *Validation* concept and according to ISO/IEC 9126 standard, a *Validation* process produces *TestReport* and *ValidationPlan* and requires *RequirementSpecification*, *Source Code, Test Report* and *User Manual* as inputs. In addition, *Validation* has *Efficiency* and *Functionality* as quality attributes and uses *Use-Cases, Prototyping,* and *Measurement* as resources (see Section 5.2.7).

According to Fig. 7.8 the system provides the learner with learning materials (LOs) of the quaried concept and a list of related SQA concepts for further investigation. The list of recommended LOs consists of random content (vedio, image, text, etc.) of pedagically reasonable learning content that are available in the net.

7.4 Implementation of SQAES

SQAES has been designed and implemented using free open source and platform independent software. Upper ontology was used for modelling the learner's profile and the context in the e-learning prototype (Bajnaid et al., 2012).

7.4.1 SQAES Software Components

This section presents the main software and technologies used to set up our system environment. As shown in Fig.7.9, in the center of the system is Web-based server which read the ontology model and retrieves queried concepts. Other related concepts are inferred using ontology reasoning mechanism of the defined ontology reasoning rules. Each SQA concept is mapped to several learning objects. The system retrieves those learning objects that are associated with the queried concept from the LOs repository. Retrieved LOs are saved in a buffer to be filtered based on the leaner profile and then provided to the leaner.

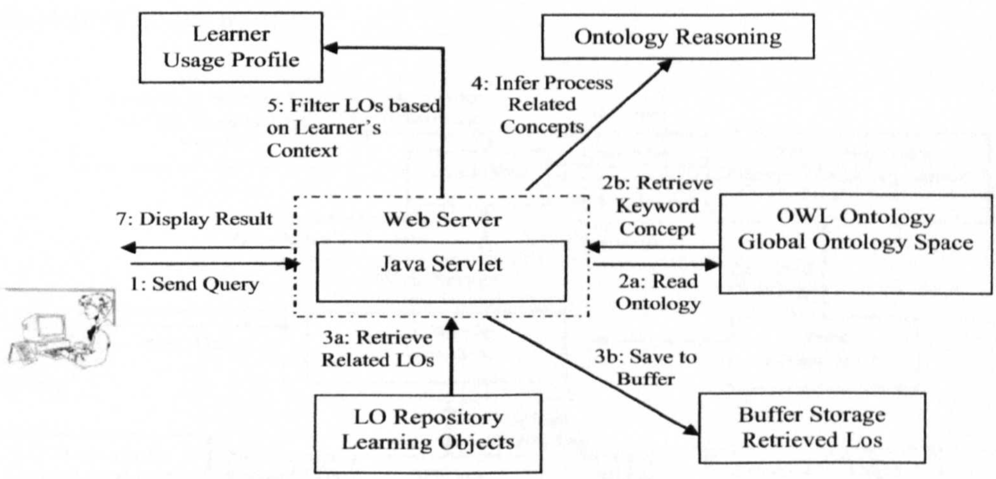


Figure 7.9: Logical Diagram of SQAES Software Components

For SQAES implementation is used a set of tools and libraries already developed for the Java programming language and the integrated development environment (IDE) such as Eclipse Software Development Kit (SDK). All components are free open source and platform independent software. The main components and processing steps are shown in Fig.7.10 and the Java code of the implemented prototype is presented in *Appendix J*.

As server software, Apache Tomcat 7 (2012) allows to develop the container with a few servlets has been chosen. Here servlet means a software component which is providing service to other software components.

Jena (2012), a Semantic Web framework for Java, is used to extract data from and write to the developed OWL ontology model. The Jena framework offers a convenient way to work with ontologies and in particular for integrating ontologies into applications. The Jena framework is used to read the ontology and to create prerequisite individuals. The system uses the SWRL Tab of Protégé to build SWRL rules for ontology reasoning.

RacerPro (2011) is a Description Logic (DL) reasoner used as an interactive tool for manipulating the ontology and the SWRL rules.

Finally, JDOM (Hunter, 2008) is XML framework for Java used to process XML files of developers' usage profiles.

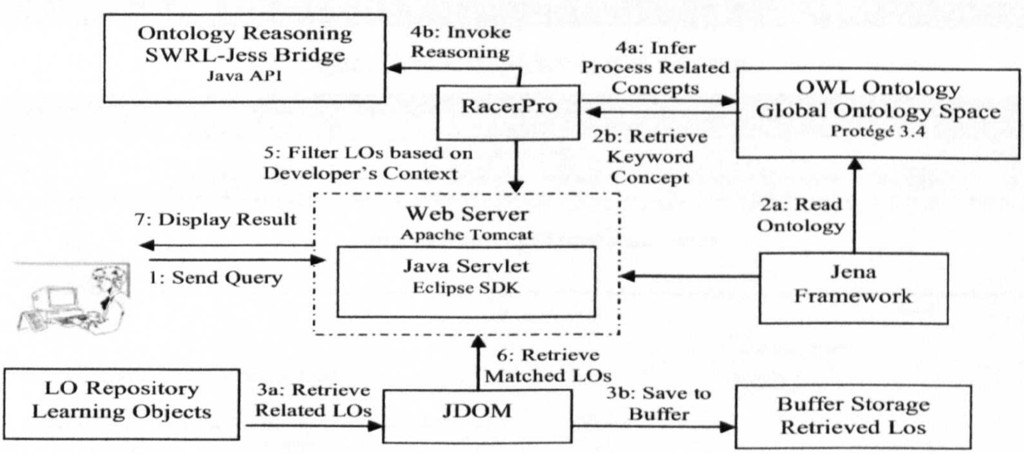


Figure 7.10: Implementation-specific Diagram of SQAES Software Components

7.4.2 Experimental Results

The prototype system provides the learner 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. Fig. 7.11a shows the login screen of SQAES where Figs. 7.11b & 7.11c are screen shots of the SQAES provided information when the user queries about the *Validation* process without the use of the ontology axioms. The user can query about an SQA concept either by typing the queried concept in the query textbox or by navigating SQA concepts.

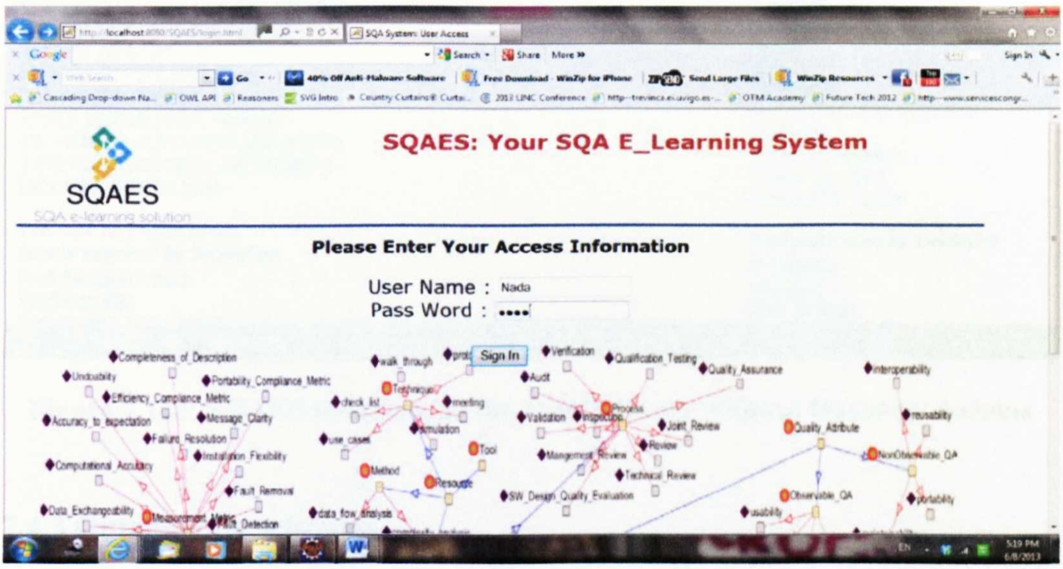


Figure 7.11a: SQAES Login Screen

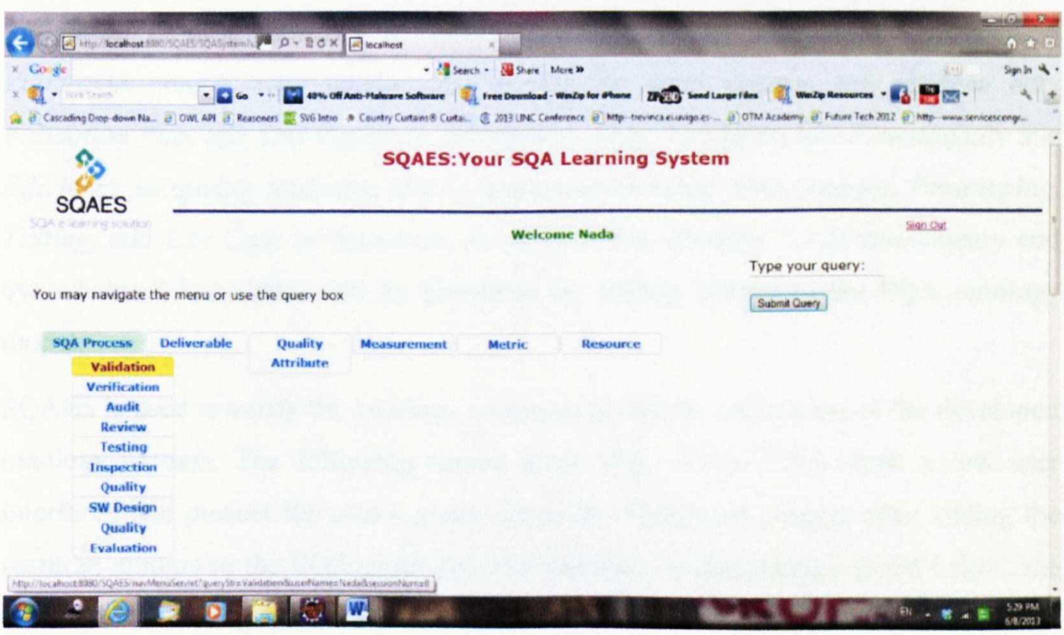


Figure 7.11b: The User Queries about the *Validation* Process

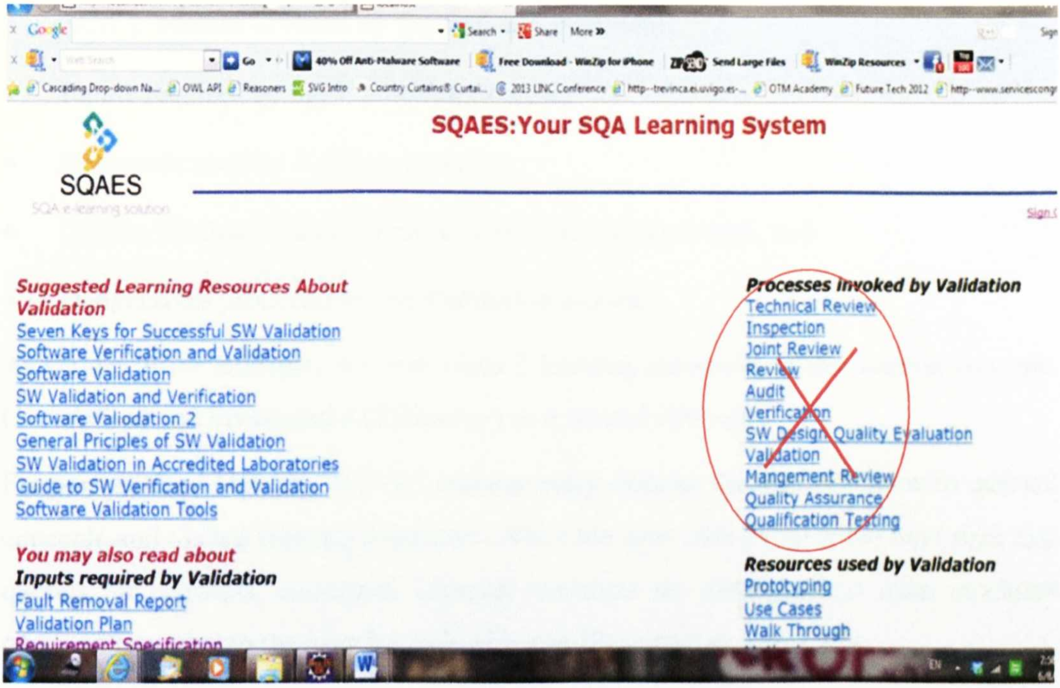


Figure 7.11c: SQAES Response to the User's Query without Ontology Axioms

7.4.3 Ontology Conciseness

In Fig. 7.11c the system displays all SQA processes as invoked processes by the *Validation* process however, in theory according to the IEEE 12207 standard, a *Validation* process may invoke only *Review* or *Audit* process and produce only *Validation Plan* and *Test Report* as deliverable. Also, *Validation* has *Functionality* and *Efficiency* as quality attributes and is implemented using *Measurement*, *Prototyping*, *Testing*, and *Use Case* as resources. As described in (Section 7.3.2) unnecessary and overwhelmed knowledge can be prevented by adding axioms to the SQA ontology model.

SQAES is used to verify the ontology conciseness and the correctness of the developed ontology axioms. The following screen shots (Fig. 7.12a-7.12e) show a few user interfaces that present the user's query about the *Validation* concept after adding the required axioms to the SQA ontology. For instance, in the example given below, the developed reasoning system allows to infer:

- SQA processes invoked by the *Validation* process;
- Inputs required by the *Validation* process;
- Resources used by *Validation* process;
- Quality attributes that are enforced by *Validation* process; and
- Deliverables produced by the *Validation* process.

As shown in the example, the user visits 2 learning resources of the queried concepts (*Validation*) and investigated (*Efficiency*) as a related concept.

For personalized learning, SQAES automatically updates the user profile with queried concepts and visited learning resources. When the user uses SQAES the next time and queries for concepts, consumed learning resources are distinguished from unvisited ones and provided to the user for re-learning as illustrated in Fig. 7.12a.

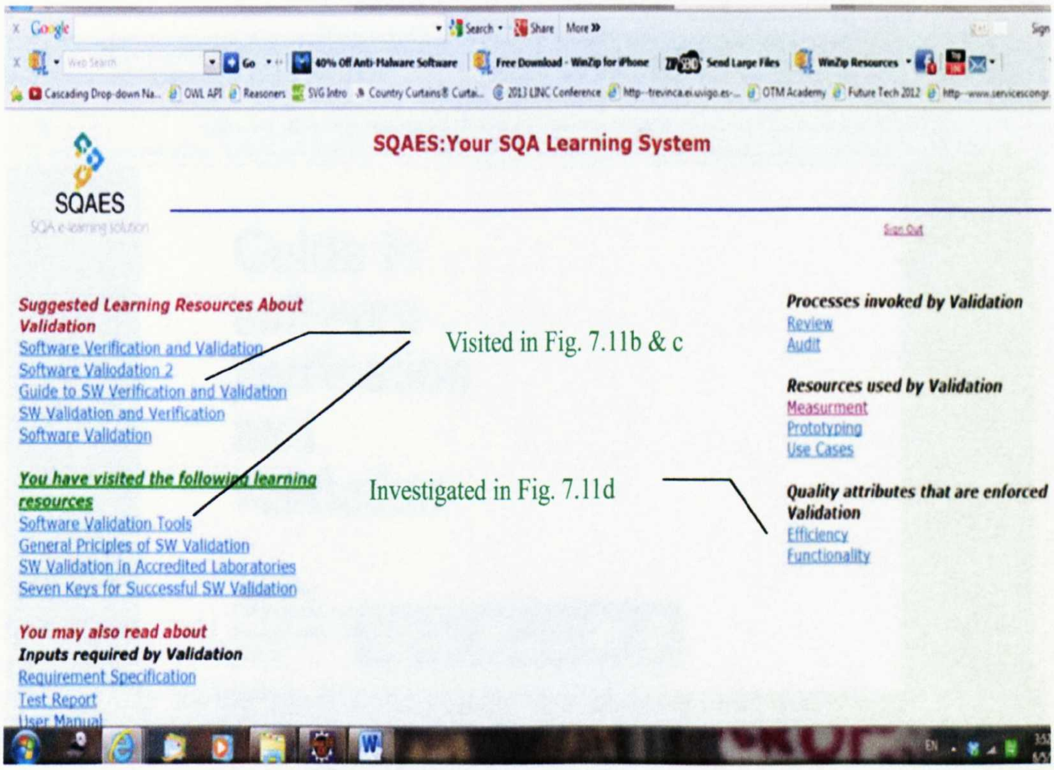


Figure7.12a: SQAES Response to the User's Query using Ontology Axioms

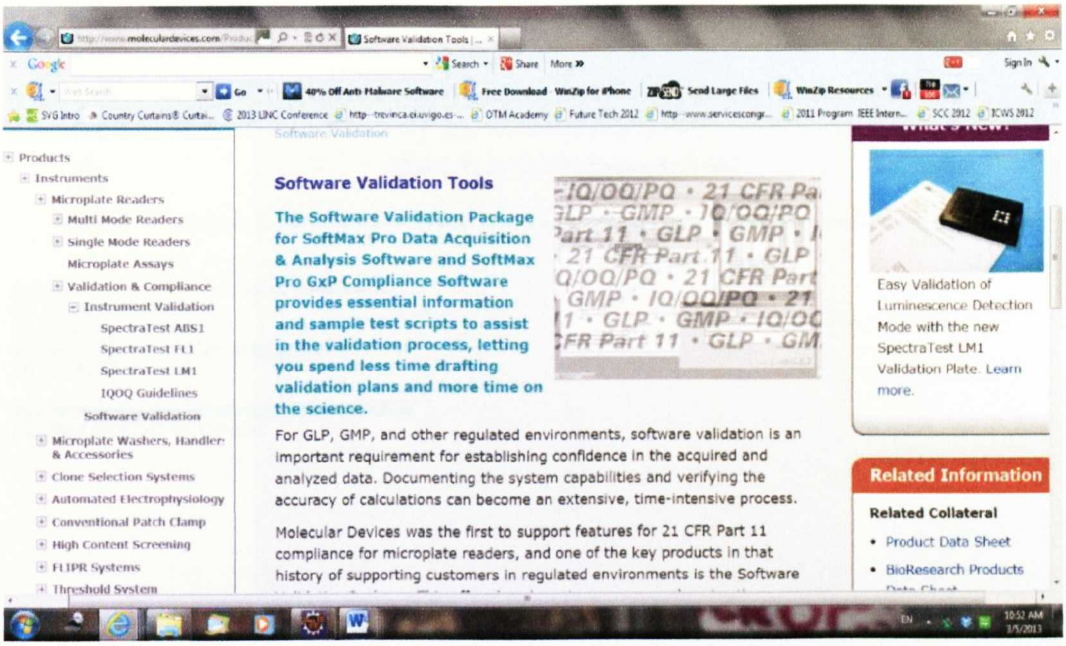


Figure 7.12b: The SQAES System

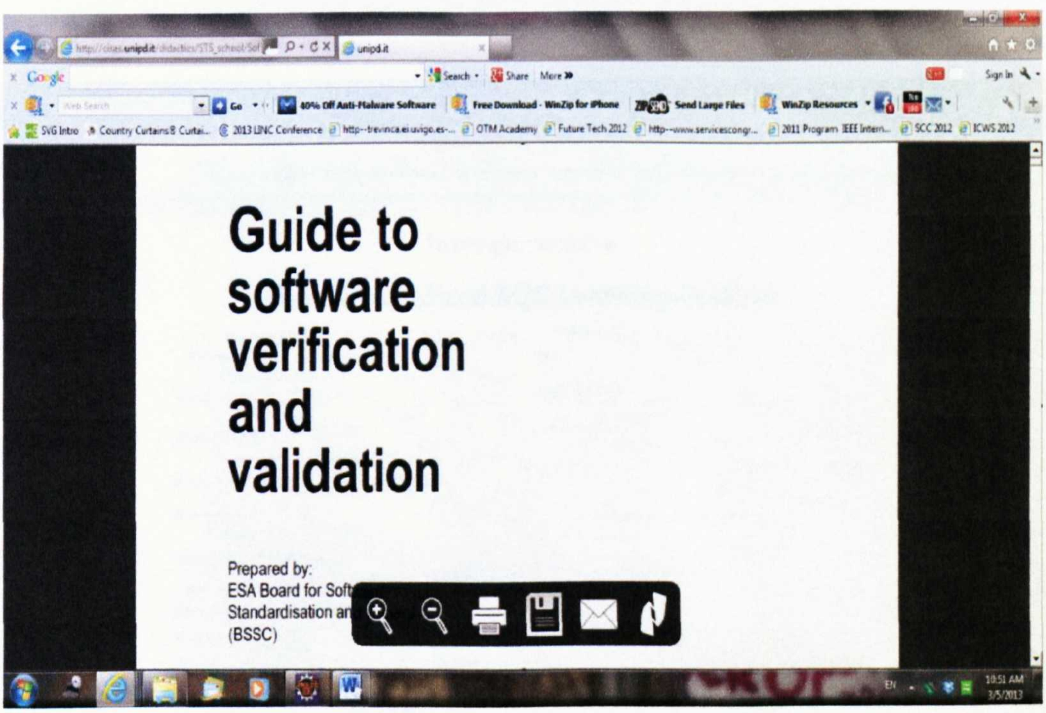


Figure7.12c: The SQAES System

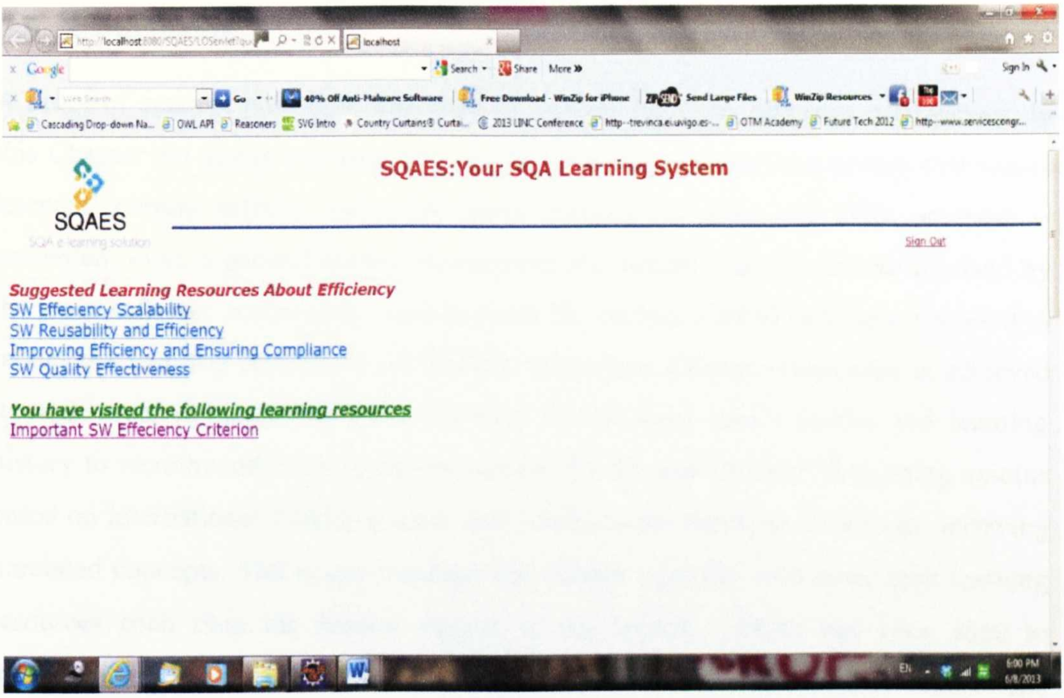


Figure7.12d: The SQAES System

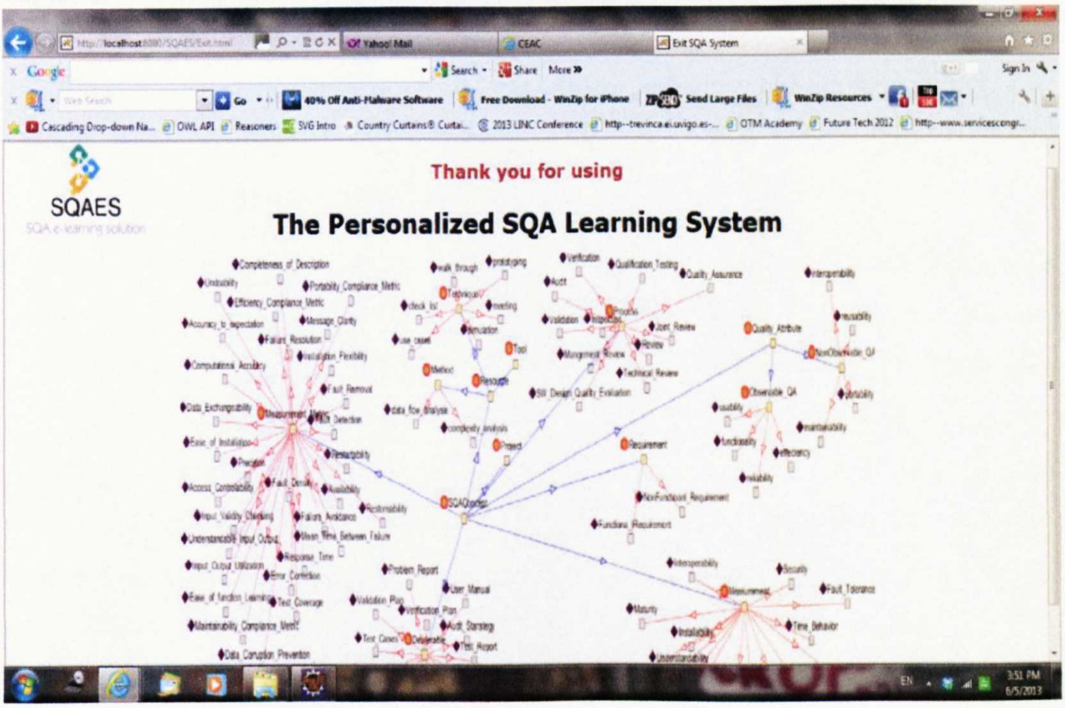


Figure7.12e: The SQAES Exit Screen

7.5 Conclusion

A proof of concept prototype was used to validate the SQA ontology deployment. In this Chapter the design and structure of a process-driven e-learning system that senses learners' current activity and guide them through the necessary SQA practices is presented. First, a general system architecture and design was introduced followed by the main software components used to build the system. Global ontology was used to model the learning context in the SQAES prototype. 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. Reasoning axioms based on international standards have been added to the ontology to prevent retrieving unrelated concepts. The system updates the learner's profile with consumed learning resources each time the learner logged in the system. JDOM has been used to manipulate developers' profiles in XML format. Finally experimental results and screen shots of using the system are provided.

Conclusions and contributions of this research work are summarized and presented in Chapter 8.

Chapter 8: Conclusions and Future Work

The major research contributions in the area of modelling the SQA knowledge for learning are presented in this Chapter followed by suggested future work.

8.1 Research Contributions

This research was aimed to investigate, design, implement and evaluate a model of the SQA knowledge area that would facilitate automated retrieval of the domain knowledge using ontologies. This research defines a framework of building ontology-based application for SQA e-learning (Fig. 4.1). The presented framework can be easly transformed to reflect new standards in the domain (see the enhanced version of the SQA ontology Section 5.5). This is the first time where both domain (SQA concepts) and operational (SQA processes) knowledge are integrated into ontology along with a set of axioms and ontology reasoning tools to help developer/learner query process-realted SQA resources. This section presents a summary of the main contributions achieved to meet the research objectives.

- **Define a conceptual model of the SQA knowledge area**. Section 5.2 presented the SQA conceptual model (Fig.5.3) which is the key output of the conceptualization process (Bajnaid et al., 2010, 2011, 2012, 2013).
- **Implement machine-readable SQA ontology based on the conceptual model**. Section 5.3 presented the use of the Protégé tool to edit the formal SQA ontology in OWL, a Semantic Web open standard recommended by W3C. In contrast with other ontologies the developed SQA ontology is not just taxonomy of the domain. It is an operational ontology where the knowledge is inferred based on the SQA process the user is dealing with. Ontology axioms have been added to the SQA ontology according to the best practices and the software development life cycle. The developed formal ontology which contains 16 deliverable concepts, 24 SQA measurement concepts, 27 measurement metric concepts, 11 processes, 8 quality

attributes, 8 resources and 198 learning objects partially in (Bajnad et al., 2011; 2012).

Evaluate the developed SQA ontology. Chapter 6 presented detailed description of the evaluation approach used to validate the developed SQA ontology. The ontology was verified for consistency using Protégé and the Racer Pro ontology reasoning tool (Fig. 5.9). Clarity and completeness have been evaluated using the SQA ontology assessment questionnaire (Bajnaid et al., 2013). Chapter 6 also presented the discussion and results of the ontology assessment questionnaire distributed among SE specialist. Application-based ontology evaluation was also performed using a prototype of the ontology-based e-Learning system as presented in Chapter 7 (Bajnaid et al., 2013). For the development of the SQAES prototype, Apache Tomcat 7 has been chosen as server software. Jena is used to extract data from and write to the developed OWL ontology model. The system uses the SWRL Tab of Protégé to build SWRL rules for ontology reasoning. RacerPro has been used as an interactive tool for manipulating the ontology and the SWRL rules. Based on the suggestions and results of the evaluation an enhanced version of the SQA ontology model has been developed (Section 5.5).

In addition to the above, the followings outcomes are other achievements which are linked to the main research contributions:

- The vocabulary and relationships in the developed SQA ontology (Tables 5.3, 5.4 and 5.5) are built based on SWEBOK guide (2004) and international standards (ISO 9126, IEEE 12207, IEEE 610.12, IEEE 00100) and other documents (PMBOK 2008, CMMI v1.2, and ANSI/ISO/ASQ Q9000-2000) partially in (Bajnaid et al., 2012; 2013);
- Approach to implement semantic representation of the user profile (the Developer/Learner sub-ontoloyg Section 7.2.1) in the integrated ontology-based prototype SQAES (Bajnaid et al., 2012; 2013).

8.2 Future Work

This research area is very rich and many ideas can be developed as extension to this research. Some attempts has been done to carry out certain extensions, however, they were excluded as they don't contribute to the main objectives of this research. Some of these extensions are listed below.

8.2.1 Towards Task-Level SQA Ontology

The IEEE 12207 (2008) defines an activity as a set of cohesive tasks of a process where tasks are requirements, recommendation or permissible action intended to contribute to the achievement of one or more outcomes of a process.

Additionally to the current process-level SQA ontology (shaded in orange), the ontology can be extended to be task-based level ontology where each SQA process is composed of activities and tasks as illustrated in Fig. 8.3.

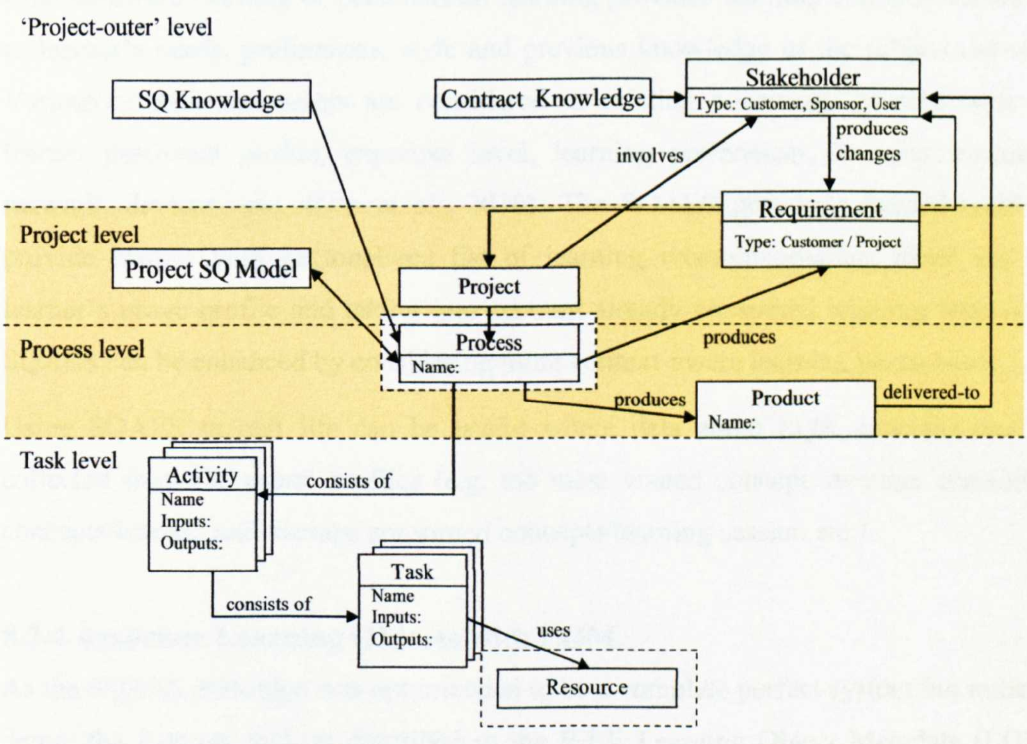


Figure 8.3: Future view of the SQA including task, project and project-outer levels.

8.2.2 Merging the SQA Ontology with other SE Knowledge Areas

SQA is not a separate SE process. Quality implies in every action and step of the software development process from requirement specification to post-delivery evolution. A future work might be conducted towards merging the developed SQA ontology to an upper level SE ontology.

8.2.3 Enhancement of the SQAES Prototype

The current version of SQAES has been improved to allow the user to navigate SQA concepts in addition to quering for a concept. I such a case the user doesnot need to remember all SQA concepts. For an attractive and flexible e-learning environment, the SQAES prototype can be supported with a graphical generator for better representation of the outputs. The use of the Scalable Vector Graphics (SVG), an XML-based vector image format for two dimensional graphics (2002), can be a way forward.

Context-aware learning or personalized learning provides learning contents according to learner's needs, preferences, style and previous knowledge of the subject domain. Various context parameters are considered in existing e-learning systems such as: learner personnel profile, expertise level, learning preferences, learning situation, network, devices…etc. (Das et al., 2010). The SQAES prototype implemented to provide learner with personalized list of learning recommendations based on the learner's usage profile and taking into account already consumed learning resources. SQAES can be enhanced by considering more context-aware learning parameters.

Using SQAES in real life can be useful where data about SQA concepts can be collected from the users' profiles (e.g. the most visited concept, average consumed concepts/learner, and average consumed concepts/learning session etc.)

8.2.4 Associate Learning Objects with LOM

As the SQAES prototype was not intended to be a complete perfect system but rather a demo, the features such as described in the IEEE Learning Object Metadata (LOM) standard were not addressed. LOM is a meta-date conceptual model with different attributes such as language, title, date, format, teaching style, and prerequisite enables the sharing and exchange of learning objects across any technology supported learning systems (Holzinger et al., 2001). The use of LOM to find and retrieve the learning objects in SQAES will help to provide structured description of the learning objects that can be used by various applications and hence will offer an open pool of learning resources to the learner.

8.2.5 Towards an Extension of SQAES for Agile Software Development

Agile software development methods aim to develop software as fast as possible with continuous feedback from customers (Rech, 2007). Although agile methods produce software faster, they need to produce quality products. While quality software is the output of quality process, it is not clear how current agile practices and methods attain quality under time pressure and in an unpredictable requirements environment. As an extension of the use of SQAES, the system can be used to provide agile developers with, just-in-time and in a contextualized way, resources that deal with SQA related aspects of the software process at hand and hence might improve quality in an agile software development environment.

8.2.5.1 Extending of the SQA Ontology with Agile Terminology

SQAES, the prototype system developed in this research, has to be extended to address the challenges related to the role of Quality Assurance in agile projects by developing a process-driven recommender that takes into consideration the type of software process the developer is dealing with, as well as its SQA requirements, quality attributes, SQA measurements and metrics, related techniques and procedures. The main motivation is to achieve the Agile Manifesto' principle, that is "Build projects around motivated individuals, give them the environment and support they need and trust them to get the job done" (Judy, 2009).

Ontology expandability, refers to the ability to add new definitions to the ontology without altering the already stated semantic (Gómez-Pérez, 2001), has been evaluated. The SQA ontology is partially extended to include agile terminology. To support agility that relies on individual's tacit knowledge and that is very much based on standard work practices and methodologies, the software engineering knowledge sources (Section 5.2) and some agile software development methods resources

(Mankandla and Dwolatzky, 2006; Abrahamsson et al., 2002) have been used aided by domain specialists to extend the vocabulary and relationships of the SQA ontology developed in chapter 5. Table 8.2 shows extracted knowledge used to extent the conceptual model of the SQA ontology (Bajnaid et al., 2012).

It should be noted that the 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.

Table 8.2: Additional Agile Terminology

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

8.2.5.2 Possible Scenario of Using Agile-Oriented SQAES

To use SQAES in an agile development environment the ontology has to be automatically used to annotate software development related keywords. The possible scenario could be as follows: 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. Example of such implementation is shown in Fig. 8.4 with 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) where selected LO about the *Continuous_Integration* technique used by the *Validation* process is visited.

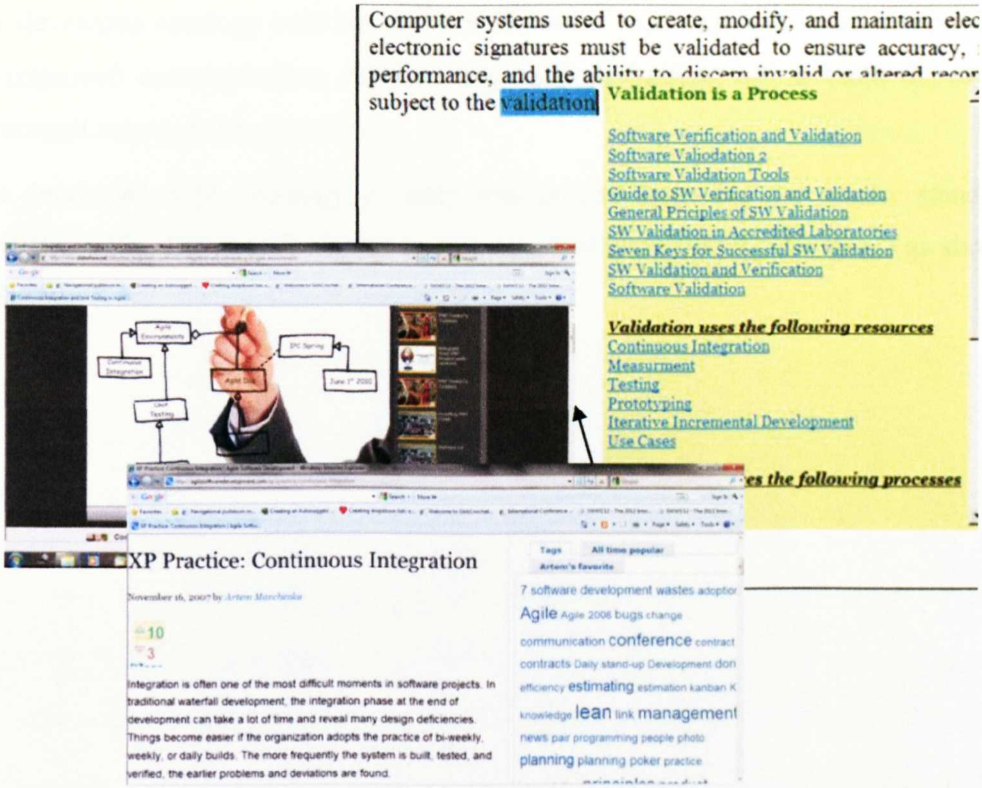


Figure 8.4: Combined view of the SQAES System for Agile SW Development (Bajnaid et al., 2012)

8.3 Epilogue

This research has designed and developed a Software Quality Assurance ontology that at the first time represents both domain and operational knowledge of the SQA knowledge area. The ontology provides consistent terminology that aims to support communication between people and software agents.

The common vocabulary and relationships modelled in the developed ontology is an attempt to resolve the problem of inconsistency among current standards and proposals. With a goal to develop a consistent terminology for software quality assurance, different ISO and IEEE standards were used in the ontology conceptualization activity while the Software Engineering Body of Knowledge (SWEBOK) remains the important and primary source for developing the SQA ontology.

The developed ontology built based on international standards and hence can provide an improved communication medium among organizations and a basis for future agreement among SQA community.

The developed SQA ontology is easly transformed from the old quialty standard (ISO/IEC 9126, 2001) to the latest quality standard (ISO/IEC 52010, 2011) as shown in Section 5.5.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. VTT Publications 478, 2002.

Alyahya M. (2006). *A New Model for Identifying and Describing Question/Answer Resource Semantic for Distributed Access.* PhD Thesis, University of Nottingham, UK.

Andreasen T. and Bulskov H., (2007). On Browsing Domain Ontologies for Information Base Content. Lecture Notes in Artificial Intelligence 2007.

Anquetil N., Oliveira K.M., and Dias M., "Software Maintenance Ontology". In: Calero, C., Ruiz, F. and Piattini, M. Ontolgies in Software Engineering and Software Technology, Springer-Verlag, Berlin, 2006, pp.153-174.

ANSI/ISO/ASQ Q9000-2000. Quality Management Systems: fundamentals and Vocabulary, American Society of Quality, American National Standard, 2000.

Antoniou, G., and Harmelen, F., (2003). Web ontology language: Owl, Handbook on Ontologies in Information Systems, Springer, p. 67--92, 2003.

Apache (2012) The Apache Tomcat Fondation [online]. Available at: http://tomcat.apache.org/ (accessed: 3 March 2013).

Apache Jena (2012) Java framework for building Semantic Web applications [online]. Available at: http://jena.apache.org/ (accessed: 3 March 2013).

Baader, F., Horrocks, I., & Sattler, U. (2005). Description logics as ontology languages for the semantic web. *Mechanizing Mathematical Reasoning,* p. 228-248.

Bajnaid N., Benlamri R. and Cogan B. (2010), "Ontology-Based E-Learning System for SQA Compliant Software Development", International Journal of Information Studies July 2010, Volume 2, Issue 3, pp. 174 – 181 http://www.dline.info/diwt2010.php.

Bajnaid N., Benlamri R. and Cogan B. (2011). Context-Aware SQA E-learning System, *ICDIM 2011: Proc. of the Sixth International Conference on Digital Information Management*, Melbourne, Australia, 26-28 Sept., 2011, pp. 327-331.

Bajnaid N., Benlamri R. and Cogan B. (2012), "An SQA e-Learning System for Agile Software Development", Proc. of the Fourth International Conference on Networked Digital Technologies, Dubai, UAE, April 24-26, 2012. Communications in Computer and Information Science(CCIS 7899) Series of Springer LNCS (294), 2012, pp. 69-83. ISBN: 978-3-642-30566-5. E-ISBN: 978-3-642-30567-2.

Bajnaid, N., Cogan, B., and Al-nuaim, H., (2008).Software Quality Ontology for Teaching: A Development Issues. *IIT 2008: Proc. Int. Conf. on Innovations in Information Technology, 2008*, Dec. 16-18, 2008 p.352-356. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4781698&isnumber=4781627

Bajnaid N., Benlamri R., Pakstas A. and Salekzamankhani S. (2013). "Software Quality Assurance Ontology: from Development to Evaluation". SEKE 2013: Proc. Of the 25th International Conference on Software Engineering and Knowledge Engineering, Boston, USA, 27-29 June 2013, in press.

Barbosa E. F., Nakagawa E. Y., Maldonado J. C., (2006).Towards the establishment of an ontology of software testing, in: 18th Int. Conf. on Soft. Engineering and Knowledge Engineering (SEKE'06), 2006, p.522-525.

Berri, J., Benlamri, R., and Atif, Y., (2006). "Ontology-based Framework for Context-aware Mobile Learning", *Int. Workshop Context-aware Mobile Computing '06*, Vancouver, Canada, July 3-6, 2006, pp.1307-1310

Bishop R., Lehman M.M. (1991). A View of Software Quality. IEEE Col. on Designing Quality into Software Based Systems. London, 14 Oct. 1991.

Bloom, B.S., et al., (1956). Taxonomy *of Educational Objectives: The Classification of Educational Goals. Handbook I: The Cognitive Domain.* New York: Longman. 1956.

Boehm, B., Chulani, S., Verner, J., Wong, B., (2009). "Seventh workshop on Software Quality," Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on, vol., no., pp.449-450, 16-24 May 2009.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5071056&isnumber=5070947

Bourque, P., Buglione, L., Abran, A., April, A., (2004). Bloom's Taxonomy Levels for Three Software Engineer Profiles, *Proceedings of the 11Annual International Workshop on Software Technology and Engineering Practice(STEP'04),* 2004, pp. 123-129.

Brank, J., Grobelnik, M. and Mladenic, D. (2005). A survey of ontology evaluation techniques. In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005), Ljubljana, Slovenia.

Calero, C., Ruiz, F. and Piattini, M., (2006). Ontologies in Software Engineering and Software Technology, Springer

Chen, C. M. (2009). Ontology-based concept map for planning a personalized learning path. *British Journal of Educational Technology,* vol. 40, no. 6, pp. 128-158, 2009.

Chung, L, Nixon, B., Yu, E. & Mylopoulos, J. (2000). Non- Functional Requirements in Software Engineering. Boston: Kluwer Academic Publishers.

Corazzon R. (2013), Theory and History of Ontology [online]. Available at: http://www.ontology.co/ accessed by: 27 April 2013.

Das, M. et al., (2010), Context Aware E-Learning system with Dynamically Composable Learning Objects. (IJCSE) International Journal of Computer Science and Engineering, Vol. 2, No. 4, 2010, pp. 1245-1253.

Dey AK and Abowd GD (2001), "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications". Human-Computer Interaction Journal, 16(2–4), pp. 97–166

Dupuis, R.; Bourque, P.; Abran, A., (2003). SWEBOK guide an overview of trial usages in the field of education, Frontiers in Education, 2003. FIE 2003. 33rd Annual , vol.3, no., pp. S3C-19-23 vol.3, 5-8 Nov. 2003.

Dzcmydiene D. and Tankeleviciene L. (2008), "On the Development of Domain Ontology for Distance Learning Course", in L. Sakalauskas, G.W. Weber and E.K.

Zavadskas (Eds.): Selected papers from the The 20th international conference EURO mini conference "Continuous optimization and knowledge-based technologies" EurOPT'2008: May 20-23, Neringa, Lithuania, ISBN 978-9955-28-283-9. pp. 4744 79, 2008.

Fernández-Lopez, M., et al., (1999). Building a Chemical Ontology using METHONTOLOGY and the Ontology Design Environment, IEEE Intelligent Systems and their Applications 4(1), 1999, 37-46.

FernÜndez-López M., Gómez-Pérez A., (2002). *Overview and analysis of methodologies for building ontologies*. The Knowledge Engineering Review 17(2):129–156

García F., Bertoa, M., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M. and Genero, M., (2006). Towards a consistent terminology for software measurement, Information & Software Technology, 48(8), pp.631-644.

García F., et al., (2009). Effective Use of Ontologies in Software Measurements. The Knowledge Engineering Review, 24, pp. 23-40. doi:10.1017/S0269888909000125.

Genesereth MR, Fikes RE (1992) Knowledge Interchange Format.Version 3.0.Reference Manual. Technical Report Logic-92-1. Computer Science Department. Stanford University, California. http://meta2.stanford.edu/kif/Hypertext/kif-manual.html

Glass, R., (1997). "An Early History of Software Engineering". In the Beginning: Personnel Recollections of Software Pioneers. Wiley-IEEE Computer Society Press, 1997.

Gómez-Pérez A.,Rodrigues L., Santos A., Barbeira J., and Carvalho R. (2006), "Using Ontologies for eLearning Personalization," Proc. of the 3rd E-learning Conference - Computer Science Education, Coimbra, Portugal, pp. 155-160, Sept. 2006.

Gómez-Pérez A (2001) Evaluation of Ontologies. International Journal of Intelligent Systems 16(3):391–409

Gómez-Pérez A, Corcho O (2002) Ontology Languages for the Semantic Web. IEEEIntelligent Systems & their applications 17(1):54–60

Gómez-Pérez, A., Fernandez-López, M. & Corcho, O.,(2004). Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic Web, Springer-Verlag, New York; London.

Gómez-Pérez A., Fernández-López M., de Vicente A., (1996). Towards a method to conceptualize domain ontologies. In: van der Vet P (ed) ECAI'96 Workshop on Ontological Engineering. Budapest, Hungary, pp. 41–52

Gruber TR (1993). A Translation Approach to Portable Ontology Specification Knowledge Acquisition 5(2):199–220.

Gruber, T., (1995). Towards principles for the design of ontologies used for knowledge sharing, *Int. Journal of Human-Computer Studies*, Volume 43, No. 5/6.

Gruber, T., (2008). Ontology, Entry in the Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds.), Springer-Verlag, to appear in 2008.

Grubišić A., Stankov S., Rosić M. and Žitko B., (2009). Controlled Experiment Replication in Evaluation of e-Learning System's Educational Influence, Computers & Education, 53 (2009), pp. 591–602.

Gruninger, M. and Fox, M., (1995). Methodology for the Design and Evaluation of Ontologies, *Proceeding of Workshop on Basic Ontological Issues in Knowledge Sharing* in IJCAI 95, Mont-real, Canada, 1995.

Happel, H-J. and Seedorf, S., (2006). Applications of Ontologies in Software Engineering. In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA, Nov. 2006.

Hatem M., A Ramadan H., Neaguin D., (2005), e-Learning Based on Context Oriented Semantic Web, *Journal of Computer Science*, vol. 1, issue 4, pp. 500-504, 2005

Henze, N., Dolog, P., and Nejdl, W., (2004). "Reasoning and Ontologies for Personalized E-Learning in the Semantic Web". *Educational Technology and Society*, 7(4), 2004. pp. 82-97.

Hilburn, Y. and Towhidnejad, M., (2002). "Software Quality Across the Curriculum," Frontiers in Education, 2002. FIE 2002. 32nd Annual, vol.3, no., pp. S1G-18-S1G-23 vol.3, 6-9 Nov. 2002URL:

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1158641&isnumber=25193

Holzinger A., Kleinberger T., and Müller P., (2001). Multimedia Learning Systems based on IEEE Learning Object Metadata (LOM), presented at ED-Media World Conference on Educational Multimedia, Hypermedia and Telecommunications, Tampere (Finland), 2001. p. 772-777.

Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., Wroe, C., (2007). A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. The University Of Manchester.

Horrocks I, van Harmelen F (eds) (2001) Reference Description of the DAML+OIL(March 2001) Ontology Markup Language. Technical report. http://www.daml.org/2001/03/reference.html

Horrocks I., Patel-Schneider P.F., Boley H., Tabet S., Grosof B., and Dean M., (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.

Hunter J. (2008) JDOM [online]. Available at: http://www.jdom.org/index.html (acceded: 3 March 2013).

IEEE 610.12:1990. Glossary of Software Engineering Terminology: ANSI/IEEE. In: IEEE Software Engineering Standards, vol. 1, Customer and Terminology Standards. IEEE, Inc. 1999.

IEEE 1074:1996. IEEE Standard for Developing Software Life Cycle Processes. Std. 1074–1995 IEEE Computer Society.

IEEE 00100:2000. The Aauthoritative Dictionary of IEEE standards terms.

ISO. "JTC 1 - Information technology". ISO. Retrieved 2009-11-1

ISO 9000:1992. International Standards for Quality Management. Genève, Switzerland, International Organization for Standardization, 1992.

ISO/IEC 9126-1:2001. Software Engineering – Product Quality, Part1: Quality Model, 2001

ISO/IEC 9126-2:2003. Software Engineering – Product Quality, Part1: External Metrics, 2003

ISO/IEC 9126-3:2003. Software Engineering – Product Quality, Part1: Internal Metrics, 2003

ISO/IEC 9126-4:2004. Software Engineering – Product Quality, Part1: Quality In Use Metrics, 2004

ISO/IEC 12207, IEEE Std 12207-2008: System and Software Engineering – Software Life Cycle Processes

ISO/IEC 15288:2008. System and Software Engineering: system Life Cycle Processes.

ISO/IEC 25010:2011. Systems and Software Engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, 2011

ISO/IEC 25023:2011. Systems and Software Engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality, 2011

Jakkilinki, R., Sharda, N., and Georgievski, M., (2005). Developing an Ontology for Teaching Multimedia Design and Planning. M2USIC 2005, MMU International Symposium on Information and Communication Technologies.

Jeremić, Z., Jovanović, J., Gašević, D., (2011). An Environment for Project-based Collaborative Learning of Software Design Patterns, International Journal on Engineering Education, Vol. 27, No. 1, pp. 41-51.

Jovanović, J., Knight, C., Gašević, D., Richards, G. (2006). Learning Object Context on the Se mantic Web. In: ICALT 2006. Proc. of the 6th IEEE Int'l Conference on Advanced Learning Technologies, Kerkrade, The Netherlands, pp. 669–673.

Jovanović, J., Gašević, D., Brooks, C., Devedžić, V., Hatala, M., (2007). LOCO-Analyst: a Tool for Raising Teachers' Awareness in Online Learning Environments, In Proceedings of the 2nd European Conference on Technology Enhanced Learning, Crete, Greece, 2007, (Lecture Notes in Computer Science, Vol. 4753), pp. 112-126

Judy, K.H., "Agile Principles and Ethical Conduct," *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on* , vol., no., pp.1,8, 5-8 Jan. 2009 doi: 10.1109/HICSS.2009.53

Kabilan, V., (2007). Ontology for Information Systems (O4IS) Design Methodology; Conceptualizing, Designing and Representing Domain Ontologies, PhD thesis.

Kassab M., (2009). Formal and quantitative approach to non-functional requirements modelling and assessment in software engineering. PhD thesis, Concordia University, Canada.

Kayed A., Hirzalla N., Samhan A., Alfayoumi M., (2009). Towards an Ontology for Software Product Quality Attributes, Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services, p.200-204, May 24-28, 2009 [doi>10.1109/ICIW.2009.36]

Klapholtz, D., McDonald, J., and Pyster, A.,(2009). *The Graduate Software Engineering Reference Curriculum (GSwERC).* In Proceedings of the 2009 22nd Conference on Software Engineering Education and Training - Volume 00 (February 17 - 20, 2009). CSEET. IEEE Computer Society, Washington, DC, 290-291. DOI= http://dx.doi.org/10.1109/CSEET.2009.62

Kontopoulos, E.,Vrakas D., Kokkoras F., Bassiliades N., Vlahavas I.,(2007). An Ontology-based Planning System for E-course Generation. Expert Systems with Applications,2007, doi:10.1016/j.eswa.2007.07.034

Kusters, R.J.; van Solingen, R.; Trienekens, J.J.M., (1999).Strategies for the Identification and Specification of Embedded Software Quality, Software Technology and Engineering Practice, 1999. STEP '99. Proceedings , vol., no., pp.33-39.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=798477&isnumber=17345

Lassila O. and Swick R. (1991). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, World Wide Web Consortium, Cambridge (MA), February 1999

Lenat, DB, and Guha, RV, (1990). Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project. Addison-Wesley, Boston, Massachusetts.

MacGregor R (1991). Inside the LOOM Classifier. SIGART bulletin 2(3):70–76

Mahoney, M.S., (2004). "Finding a History for Software Engineering". Annals of the History of Computing, IEEE, vol. 26, issue 1, 2004, pp. 8-19.

Mnkandla, E., Dwolatzky, B. (2006). Defining Agile Quality Assurance. Proc. ICSEA 2006: *International Conference on Software Engineering Advances*.

Mendes, O., and Abran, A., (2004). Software Engineering Ontology: A Development Methodology, Position Paper, Metrics News 9:1, August, pp. 68-76.

Neches R. et al., (1991). Enabling technology for knowledge sharing, AI Magazine Volume 12 No.3 pp.16-36 Fall 1991.

Nicola, A., Missikoff, M., and Navigli, R., (2005). A proposal for a Unified Process for ONtology building: UPON, *Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA)*, Copenhagen, Denmark.

Nicola, A., Missikoff, M., and Schiappelli, F. (2004). Towards an Ontological Support for Elearning Courses, OTM Workshops, LNCS 3292, pp. 773–777, 2004. Springer-Verlag Berlin Heidelberg.

Noy, N. & McGuinness, D., (2001). Ontology development 101: A guide to creating your first ontology, Technical report, Stanford University.

Obrst L., et al. (2007). The Evaluation of Ontologies: Toward Improved Semantic Interoperability. Chapter in: Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, Christopher J. O. Baker and Kei-Hoi Cheung, Eds., Springer.

Osterweil, L. J.,(2007). A Future for Software Engineering?. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC.

Paech, B., Kerkow, D., (2004). Non-Functional Requirements Engineering - Quality is Essential.In: 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ 2004.

http://www.sse.uni-essen.de/refsq/downloads/toc-refsq04.pdf

Parsia, B., Sirin, E., and Grau, P., (2005). Cautiously Approaching SWRL, Technical Report, University of Maryland, MIND Lab, 23 February, 2005.

Perez, A. and Benjamins, V., (1999). Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem Solving Methods, IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, August 2, 1999.

PMBOK (2008). A Guide to the Project Management Body of Knowledge, Project Management Institute, 4th edition.

Pressman, R.S., (2005). Software Engineering: a Practitioner's Approach, Sixth edition. McGraw-Hill Inc.

RacerPro (2011). The RacerPro Knowledge Representation and Reasoning System [online]. Available at: http://www.racer-systems.com/index.phtml (accessed: 13 April 2013).

Rech, J., (2007) "Handling of Software Quality Defects in Agile Software Development," in Agile Software Development Quality Assurance, I. Stamelos and P. Sfetsos, Eds.: Idea Group Inc., 2007.

Rothman, J, (2002). What Does it Cost to Fix a Defect?, Column Archive, StickyMinds.com. http://www.stickyminds.com.

Samhan A.,(2008). An Ontology for Software Product Quality Attributes, Middle East University for Graduate Studies Master's thesis.

Saiedian, H. & Weide, B., (2005). The New Context for Software Engineering Education and Training, The Journal of Systems and Software 74 (2005), pp. 109-111.

Scalable Vector Graphics (SVG) 1.2 W3C Working Draft. World Wide Web Consortium. 15 November 2002. Retrieved 29 August 2010.

Schmidt A. and Winterhalter C. (2004) User Context Aware Delivery of e-Learning Material: Approach and Architecture. J Univers Comput Sci 10(1):28–36

Shehzad, A., and Ngo, H., (2004). Formal Modelling in Context Aware Systems, KI-Workshop Modelling and Retrieval of Context (MRC2004), University of Ulm, Ulm, Germany, September 23-27, 2004, pp.13-24

Simons, C., Parmee, I., and Coward P., (2003). 35 Years On: To What Extent Has Software Engineering Design Achieved its Goals?, IEE Proceedings-Software, vol. 150, no. 6, December 2003, pp. 337-350.

Smith, B., (2003). Ontology, Preprint version of Chapter "Ontology", in L. Floridi (ed.), Blackwell Guide to the Philosophy of Computing and Information, Oxford: Blackwell, 2003, 155–166.

Smith, M., Welty, C., and McGuinness, D., (2004). OWL Web Ontology Language Guide, *W3C Recommendation*, 10 February, 2004, Available at: http://www.w3.org/TR/owl-guide/#owl_Class

Sonsovsky S. and Gavrilova T. (2006). Development of Educational Ontology for C-Programming, Information Theories and Applications, vol. 13 (4), pp. 303-308.

Spyns, P., Meersman, R., and Jarrar, M., (2002). Data Modelling versus Ontology Engineering, ACM SIGMOD Record, v.31 n.4, December 2002

Stojanovic, L., Staab, S., Studer, R., (2001).e-Learning Based on the Semantic Web. In: Fowler, W., Hasebrook, J. (eds.) Proc. of WebNet'2001, World Conference of the WWW and Internet, pp. 1774-1783. AACE (2001).

SWEBOK, (2004). Guide to the Software Engineering Body of Knowledge, ed. Bourque P., and Dupuis R. IEEE Computer Society Press, 2004. Available at: http://www.swebok.org

Thomas, S.A.; Hurley, S.F.; Barnes, D.J., (1996). Looking for the human factors in software quality management, Software Engineering: Education and Practice, 1996. Proceedings. International Conference, vol., no., pp.474-480.

Uschold, M., and King, M., (1995). Towards a Methodology for Building Ontologies, *Proc. of Work-shop on Basic Ontological Issues in Knowledge Sharing in IJCAI 1995*, Montreal, Canada.

Uschold, M., and Gruninger, M., (1996). Ontologies: Principles, Methods, and Applications, Knowledge Engineering Review, Volume 11 number 2.

Vallet D., Fernándndez M. and Castells P., (2005). An Ontology-Based Information Retrieval Model, Proc. Second European Semantic Web Conf. (ESWC ',05).

Vrandečić, D.,(2009). Ontology Evaluation, Handbook on Ontologies, International Handbooks in Information Systems, 2nd edition, Springer, Heidelberg, 2009, pp. 293-313.

Wang, X., et al., (2004). Ontology-Based Context Modelling and Reasoning using OWL, Proc. 2nd IEEE Annual Conf. Pervasive Computing and Communications Wksps. (PERCOMW'04).

Welty C., Guarino N., (2001). Supporting Ontological Analysis of Taxonomic Relationships. Data and Knowledge Engineering 39(1):51–74.

Wille, C., Abran, A., Desharnais, J.M. and Dumke, R.(2003).The Quality Concepts and Sub Concepts in SWEBOK: An ontology Challenge, in Investigations in Software Measurement, Proceedings of the 13th International Workshop on Software Measurement, Montreal, Canada, pp.113-130.

Wille, C.,Dumke R., Abran A. and Desharnais J.M.,(2004). E-learning Infrastructure for Software Engineering Educations: Steps on Ontology Modelling for SWEBOK, Proceedings of the IASTED International Conference on Software Engineering, pp. 520-525.

Yu, Z., Zhou, X., and Shu, L., (2010). Towards a Semantic Infrastructure for Context-aware E-Learning, Multimedia Tools and Applications, 47(1), pp. 71-86.

Zhao Yajing, Dong Jing, Peng Tu,(2009). Ontology Classification for Semantic-Web-Based Software Engineering, *IEEE Transactions on Services Computing*, v.2 n.4, 303-317.

PUBLICATIONS

1. Accepted Publications

2. Bajnaid N., Pakstas A., Salekzamankhani S.and Benlamri R. "Ontology-Based Personalized SQA e-Learning System". CECIIS 2013: Proc of the Centeral Europian Conference on Information and Intelligent Systems. Varazdin, Croatia, 18-20 Sept. 2013 (in press).

Bajnaid N., Benlamri R., Pakstas A. and Salekzamankhani S. (2013). "Software Quality Assurance Ontology: from Development to Evaluation". SEKE 2013: Proc. Of the 25th International Conference on Software Engineering and Knowledge Engineering, Boston, USA, 27-29 June 2013, in press.

Bajnaid N., Benlamri R. and Cogan B. (2012), "An SQA e-Learning System for Agile Software Development", Proc. of the Fourth International Conference on Networked Digital Technologies, Dubai, UAE, April 24-26, 2012. Communications in Computer and Information Science(CCIS 7899) Series of Springer LNCS (294), 2012, pp. 69-83. ISBN: 978-3-642-30566-5. E-ISBN: 978-3-642-30567-2.

http://link.springer.com/chapter/10.1007/978-3-642-30567-2_7#

Bajnaid N., Pakstas A., and Salekzamankhani S.(2012). "Ontology-Based Modelling of the Software Quality Assurance Knowledge", Proc. of the Semat Workshop on a. General Theory of Software Engineering GTSE 2012. November 8-9, 2012. KTH Royal Institute of Technology. Stockholm, Sweden.

http://semat.org/wp-content/uploads/2012/10/GTSE-2012-Proceedings.pdf

Bajnaid N., Benlamri R. and Cogan B. (2011), "Context-Aware SQA E-learning System", Proc. of the Sixth International Conference on Digital Information Management ICDIM 2011, Melbourne, Australia, 26-28 Sept., 2011.

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6093327&isnumber=6093313

Bajnaid N., Benlamri R. and Cogan B. (2010), "Ontology-Based E-Learning System for SQA Compliant Software Development", International Journal of Information

Studies July 2010, Volume 2, Issue 3, pp. 174 - 181 http://www.dline.info/diwt2010.php

Bajnaid, N., Cogan, B., and Al-nuaim, H. (2008), "Software Quality Ontology for Teaching: A Development Issues". *Innovations in Information Technology, 2008. IIT 2008. International Conference on* , vol., no., pp.352-356, 16-18 Dec. 2008 Indexed by IEEEXplore 10 Feb 2009, ISBN 978-1-4244-3396-4 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4781698&isnumber=4781627

3. Submitted Publications

Bajnaid N., Benlamri R., Pakstas A., and Salekzamankhani S."Towards Ontology-Based SQA Recommender For Agile Software Development" submitted to the Knowledge and Information Systems, Springer.

APPENDIX A: STRUCTURE OF THE SQA ONTOLOGY'S CLASSES

OWLClass: Product supClassOf: owl:Thing Examples: project management plan, operation report, user manual, and source code test cases all are individuals (instances) of the class product			
Object Property	*Domain*	*Range*	*Cardinality*
isInputTo	Product	Process	n..n
isInputToMesurement	Product	Measurement	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Resource supClassOf: owl:Thing superClassOf: Technique* superClassOf: Tool superClassOf: Method Examples: walk through, prototyping, and check list all are individuals (instances) of the class Resource			
Object Property	*Domain*	*Range*	*Cardinality*
usedBy	Resource	Process	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Project supClassOf: owl:Thing			
Object Property	*Domain*	*Range*	*Cardinality*
hasProcess	Project	Process	1..n
hasProduct	Project	Product	1..n
hasRequirement	Project	Requirement	1..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Requirement supClassOf: owl:Thing superClassOf: Functional Requirement superClassOf: Non-functional Requirement			
Object Property	*Domain*	*Range*	*Cardinality*
associatedWith	Requirement	Product	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Measurement supClassOf: owl:Thing			
Object Property	*Domain*	*Range*	*Cardinality*
measures	Measurement	Quality_Attribute	n..1
hasMetric	Measurement	Metric	1..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Metric supClassOf: owl:Thing			
Object Property	*Domain*	*Range*	*Cardinality*
conductedUsing	Metric	Process	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

OWLClass: Quality_Attribute supClassOf: owl:Thing superClassOf: Observable Quality Attribute superClassOf: Non-observable Quality Attribute			
Object Property	*Domain*	*Range*	*Cardinality*
measuredBy	Quality_Attribute	Measurement	1..n
enforcedBy	Quality_Attribute	Process	n..n
Data Type Property		*Type*	
Description		String	
Reference		String	

APPENDIX B: SQA CONCEPTS WITH RELATED AXIOMS

SQA Concept	Related OWL Axioms
SW Design Quality Evaluation	∀ invokes only (Inspection or Review) ∀ enforces only (Efficiency or Functionality or Maintainability or Portability or Usability) ∀ uses only (Prototyping or Simulation)
Validation	∀ invokes only (Audit or Review) ∀ produces only (Test_Report or Validation_Plan) ∀ hasInput only (Requirement_Specification or Scource_Code or Test_Report or User_Manual) ∀ enforces only (Efficiency or Functionality) ∀ uses only (Measurement or Prototyping or Use_Cases)
Verification	∀ invokes only (Audit or Review) ∀ produces only (Test_Report or Verification_Plan) ∀ hasInput only (Design or Requirement_Specification or Review_Report or Source_Code) ∀ enforces only Efficiency
Test Report	∀ isInputTo only (Quality_Assurance or Validation) ∀ ProducedBy only (Validation or Verification) ∀ isInputToMeasurementMetric only (Access_Controlability or Accuracy or Accuracy_to_Expectation or Availability or Completeness_of_Description or Data_Corruption_Prevention or Ease_of_Function_Learning or Error_CorrectionFailure_Avoidance or Fault_Density)
Functionality	∀ enforcedBy only (Validation or SW_Design_Quality_Evaluation) ∀ mesuredBy only (Accuracy or Interoperability or Security)
Reliability	∀ enforcedBy only Quality_Assurance ∀ measuredBy only (Fault_Tolerance or Maturity or Recoverability)
Audit Strategy	∀ producedBy only Audit

Data Exchangeability	∀ conductedUsing only (Joint_Review or quality_Assurance or Validation) ∀ isMeasurementMetricOf only (Security) ∀ hasMeasurementMetricInput only (Requirement_Specification or Review_Report or Test_Report or Design or Operation_Report or Source_Code)
Test Coverage	∀ conductedUsing only (Qualification_Testing or quality_Assurance or Validation) ∀ isMeasurementMetricOf only (Maturity) ∀ hasMeasurementMetricInput only (Requirement_Specification or User_Manual or Test_Report)
Design	∀ isInputTo only (Review or Verification) ∀ isInputToMeasurementMetric only (Access_Controlability or Completeness_of_Description or Computational_Accuracy or Data_Corruption_Prevention or Data_Exchangeability or Input_Validaty_Checking or Precision)
Fault Removal Report	∀ isInputToMeasurementMetric only Fault_Removal
Quality Assurance Plan	∀ producedBy only Quality_Assurance
Requirement Specification	∀ isInputTo only (Quality_Assurance or Review or Validation or Verification) ∀ isInputToMeasurementMetric only (Access_Controlability or Accuracy or Accuracy_to_Expectation or Completeness_of_Description or Computational_Accuracy or Failure_Avoidance or Input_Validaty_Checking or Installation_Flexability or Precision or Restorability or Test_Coverage)
Source Code	∀ isInputTo only (Review or Validation or Verification) ∀ isInputToMeasurementMetric only (Access_Controlability or Computational_Accuracy or Data_Corruption_Prevention or Data_Exchangeability or Efficiency_Compliance_Metric or

	Maitainability_Compliance_Metric or Precision)
Operation Report	∀ isInputToMeasurementMetric only (Access_Controlability or Data_Corruption_Prevention or Data_Exchangeability or Ease_Of_Installation or Fault_Density)
Problem Report	∀ isInputToMeasurementMetric only (Ease_Of_Installation or Fault_Density)
Review Report	∀ producedBy only (Review or Mangment_Review) ∀ isInputTo only Verification ∀ isInputToMeasurementMetric only (Access_Controlability or Completeness_of_Description or Computational_Accuracy or Data_Corruption_Prevention or Data_Exchangeability or Failure_Avoidance or Fault-Detection or Fault-Removal or Input-Validaty_Checking or Installation_Flexability or Precision or Restorability)
Test Report	∀ producedBy only (Validation or Verification) ∀ isInputTo only (Quality_Assurance or Validation) ∀ isInputToMeasurementMetric only (Access_Controlability or Accuracy or Accuracy_to_Expectation or Availability or Completeness_of_Description or Data_Corruption_Prevention or Data_Exchageability or Ease_of_Function_Learning or Error_Correction or Failure_Avoidance or Failure_Resolution or Fault_Density or Fault_Removal or MTBF or Access_Clarity or Precision or Response_Time or Restartability or Test_Coverage or Undoability)
User Manual	∀ isInputTo only (Quality_Assurance or Validation) ∀ isInputToMeasurementMetric only (Accuracy_to_Expectation or Accuracy or Completeness_of_Description or Restorability or Test_Coverage or Understandable_Input_Output)
Operability	∀ measures only Usability ∀ hasMeasurementMetric only (Error_Correction or Unodability or Input_Validaty_Checking or Message_Clarity)
Recoverability	∀ measures only Reliability ∀ hasMeasurementMetric only (Availability or Restorability or

	Restartability)
Accuracy	∀ measures only Functionality ∀ hasMeasurementMetric only (Accuracy_to_Expectation or Computational_Accuracy or Percision)
Fault Tolerance	∀ measures only Reliability ∀ hasMeasurementMetric only Failure_Avoidance
Installability	∀ measures only Portability ∀ hasMeasurementMetric only (Ease_of_Installation or Installation_Flexability)
Learnability	∀ measures only Usability hasMeasurementMetric only Ease_of_Function_Learning
Maintainability Compliance	∀ measure only Maintainability ∀ hasMeasurementMetric only Maintainability_Compliance_Metric
Maturity	∀ measures only Reliability ∀ hasMeasurementMetric only (Failure_Resolution or Fault_Density or Fault_Detection or Fault_Removal or Mean_Time_Between_Failure or Test_Coverage)
Resource Utilization	∀ measures only Efficiency ∀ hasMeasurementMetric only Input_Output_Utilization
Security	∀ measures only Functionality ∀ hasMeasurementMetric only (Access_Controlability or Data_Corruption_prevention or Data_Exchangeability)
Time Behaviour	∀ measures only Efficiency ∀ hasMeasurementMetric only Response_Time
Understandability	∀ mesures only Usability ∀ hasMeasurementMetric only (Completeness_of_Description or Understabndable_Input_Output)
Error Correction	∀ conductedUsing only (Testing or Validation) ∀ isMeasurementMetricOf only Operability
Access Controlability	∀ isMeasurementMetric of only Security ∀ conductedUsing only (Joint_Review or Validation or

	Quality_Assurance) ∀ hasMeasurementMetricInput only (Design or Operation_Report or Requirement_Specifiction or Review_Report or Source_code or Test_Report)
Accuracy to Expectation	∀ isMeasurementMetricOf only Accuracy ∀ conductedUsing only (Quality_Assurance or Validation) ∀ hasMeasurementMetricInput only (Requirement_Specification or Test_Report or User_Manual)
Availability	∀ isMeasurementMetricOf only Recoverability ∀ conductedUsing only Qualification_Testing ∀ hasMeasurementMetricInput only Test_Report
Ease of Installation	∀ isMeasurementMetricOf only Installability ∀ conductedUsing only Qualification_Testing ∀ hasMeasurementMetricInput only (Operation_Report or Problem_Report)
Ease of Function Learning	∀ isMeasurementMetricOf only Learnability ∀ conductedUsing only (Qualification_Testing or Validation) ∀ hasMeasurementMetricInput only (Test_Report or User_Monitoring_Record)
Failure Resolution	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only Qualification_Testing ∀ hasMeasurementMetricInput only Test_Report
Fault Density	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only (Qualification_Testing or Quality_Assurance) ∀ hasMeasurementMetricInput only (Operation_Report or Problem_Report or Test_Report)
Fault Detection	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only (Joint_Review or Verification) ∀ hasMeasurementMetricInput only Review_Report
Fault Removal	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only (Joint_Review or Verification) ∀ hasMeasurementMetricInput only (Fault_Removal_Report or Test_Report or Review_Report)

Input Output Utilization	∀ isMeasurementMetricOf only Resource_Utilization ∀ conductedUsing only Verification ∀ hasMeasurementMetricInput only Source_Code
Installation Flexability	∀ isMeasurementMetricOf only Installability ∀ conductedUsing only Validation ∀ hasMeasurementMetricInput only (Requirement_Specification or Review_Report)
Mean Time Between Failure	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only Qualification ∀ hasMeasurementMetricInput only Test_report
Message Clarity	∀ isMeasurementMetricOf only Operability ∀ conductedUsing only (Qualification_Testing or Validation) ∀ hasMeasurementMetricInput only (Test_Report or User_Monitoring_Record)
Precision	∀ isMeasurementMetricOf only Accuracy ∀ conductedUsing only (Joint_Review or Validation or Verification or Quality_Assurace) ∀ hasMeasurementMetricInput only (Design or Requirement_Specification or Review_Report or Source_Code or Test_report)
Response Time	∀ isMeasurementMetricOf only Time_Behaviour ∀ conductedUsing only Qualification_Testing ∀ hasMeasurementMetricInput only Test_Report
Restartability	∀ isMeasurementMetricOf only Recoverability ∀ conductedUsing only (Qualification_Testing or Validation) ∀ hasMeasurementMetricInput only Test_Report
Restorability	∀ isMeasurementMetricOf only Recoverability ∀ conductedUsing only (Joint_Review or Qualifiaction_Testing or Validation or Verification) ∀ hasMeasurementMetricInput only (Requirement_Specifiaction or Review_Report or Test_Report or User_Manual)
Test Coverage	∀ isMeasurementMetricOf only Maturity ∀ conductedUsing only (Qualification_Testing or Quality_Assurance or

	Validation) ∀ hasMeasurementMetricInput only (Requirement_Specification or Test_Report or User_Manual)
Undoability	∀ isMeasurementMetricOf only Operability ∀ conductedUsing only (Qualifiactio_Testing or Validation) ∀ hasMeasurementMetricInput only (Test_Report or User_Monitoring_Recoed)
Quality Assurance	∀ invokes only (Audit or Review or Validation or Verification) ∀ enforces only (Reliability or Usability) ∀ produces only Quality_Assurance_Plan ∀ hasInput only (Requirement_Specification or Test_Report or User_Manual)
SW Design Quality Evaluation	∀ invokes only (Inspection or Review) ∀ enforces only (Efficiency or Functionality or Maintainability or Portability or Usability) ∀ uses only (Prototyping or Simulation)
Verification	∀ invokes only (Audit or Review) ∀ enforces only Efficiency ∀ produces only (Test_Report or Verification_Plan) ∀ hasInput only (Design or Requirement_Specification or Review_Report or Source_Code)
Portability	∀ enforcedBy only SW_Design_Quality_Evaluation ∀ measuredBy only (Installability or Portability_Compliance)
Usability	∀ enforcedBy only (Quality_Assurance or SW_Design_Quality_Evaluation) ∀ measuredBy only (Learnability or Operability or Understandability)
Checklist Meeting	∀ usedBy (Audit or Review)
Prototyping	∀ usedBy only (SW_Design_quality_evaluation or Validation)
Simulation	∀ usedBy only SW_Desing_Quality_Evaluation
Use cases	∀ usedBy only Validation
Walk through	∀ usedBy only Review

APPENDIX C: THE OWL CODE OF THE SQA ONTOLOGY

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
<!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
<!ENTITY protege
   "http://protege.stanford.edu/plugins/owl/protege#" >
<!ENTITY xsp "http://www.owl-
   ontologies.com/2005/08/07/xsp.owl#" >
<!ENTITY swrla
   "http://swrl.stanford.edu/ontologies/3.3/swrla.owl#" >
<!ENTITY sqwrl "http://sqwrl.stanford.edu/ontologies/built-
   ins/3.4/sqwrl.owl#" >]>


<rdf:RDF xmlns="http://www.owl-ontologies.com/SQOntology#"
xml:base="http://www.owl-ontologies.com/SQOntology"
xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-
   ins/3.4/sqwrl.owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:swrl="http://www.w3.org/2003/11/swrl#"
xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#
   "
xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#
   ">
<owl:Ontology rdf:about="">
<owl:imports
   rdf:resource="http://sqwrl.stanford.edu/ontologies/built-
   ins/3.4/sqwrl.owl"/>
<owl:imports
   rdf:resource="http://swrl.stanford.edu/ontologies/3.3/swrla.
   owl"/>
</owl:Ontology>
<swrl:Variable rdf:ID="A"/>
<swrl:Variable rdf:ID="C"/>
<swrl:Variable rdf:ID="D"/>
<swrl:Variable rdf:ID="LO"/>
<swrl:Variable rdf:ID="P"/>
<swrl:Variable rdf:ID="R"/>
<Measurement_Metric rdf:ID="Access_Controlability"/>
<Measurement rdf:ID="Accuracy"/>
<Measurement_Metric rdf:ID="Accuracy_to_expectation"/>
<Process rdf:ID="Audit"/>
<Product rdf:ID="Audit_Starategy"/>
<Measurement_Metric rdf:ID="Availability"/>
<Technique rdf:ID="check_list"/>
<Measurement_Metric rdf:ID="Completeness_of_Description"/>
<Method rdf:ID="complexity_analysis"/>
<Measurement_Metric rdf:ID="Computational_Accuracy"/>
<Measurement_Metric rdf:ID="Data_Corruption_Prevention"/>
<Measurement_Metric rdf:ID="Data_Exchangeability"/>
<Method rdf:ID="data_flow_analysis"/>
<Product rdf:ID="Design"/>
<Measurement_Metric rdf:ID="Ease_of_function_Learning"/>
<Measurement_Metric rdf:ID="Ease_of_Installation"/>
<Observable_QA rdf:ID="effeciency"/>
<Measurement rdf:ID="Efficiency_Compliance"/>
<Measurement_Metric rdf:ID="Efficiency_Compliance_Metric"/>
<owl:DatatypeProperty rdf:ID="EndTime">
<rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="enforcedBy">
<rdfs:domain rdf:resource="#Quality_Attribute"/>
<owl:inverseOf rdf:resource="#enforces"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="enforces">
<rdfs:domain rdf:resource="#Process"/>
<owl:inverseOf rdf:resource="#enforcedBy"/>
<rdfs:range rdf:resource="#Quality_Attribute"/>
</owl:ObjectProperty>
<Measurement_Metric rdf:ID="Error_Correction"/>
<Measurement_Metric rdf:ID="Failure_Avoidance"/>
<Measurement_Metric rdf:ID="Failure_Resolution"/>
<Measurement_Metric rdf:ID="Fault_Density"/>
<Measurement_Metric rdf:ID="Fault_Detection"/>
<Measurement_Metric rdf:ID="Fault_Removal"/>
<Product rdf:ID="Fault_Removal_Report"/>
<Measurement rdf:ID="Fault_Tolerance"/>
```

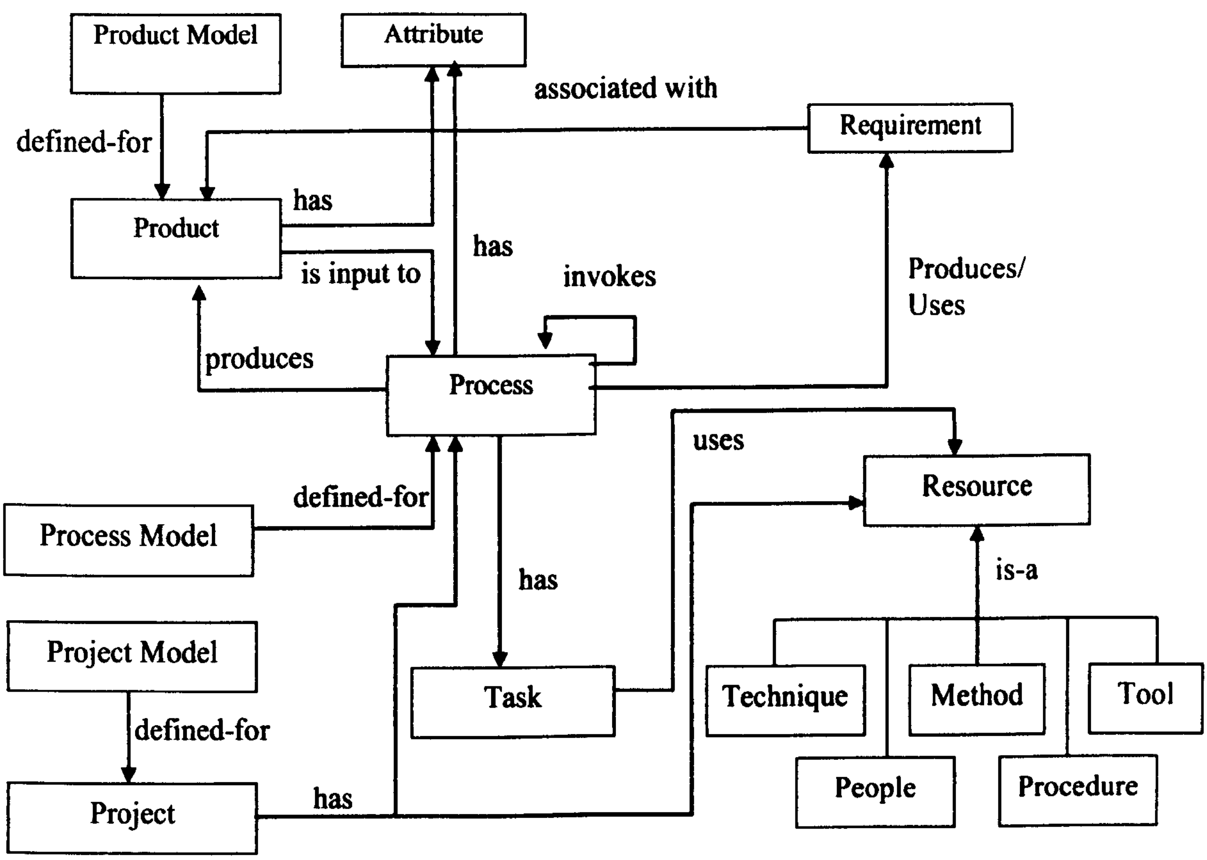
```
<Requirement rdf:ID="Functiona_lRequirement"/>
<Observable_QA rdf:ID="functionality"/>
<owl:ObjectProperty rdf:ID="has">
<rdfs:domain rdf:resource="#SQAConcept"/>
<owl:inverseOf rdf:resource="#isA"/>
<rdfs:range rdf:resource="#SQAConcept"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasInput">
<rdfs:domain rdf:resource="#Process"/>
<owl:inverseOf rdf:resource="#isInputTo"/>
<rdfs:range rdf:resource="#Product"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasMeasurementMetric">
<rdfs:domain rdf:resource="#Measurement"/>
<owl:inverseOf rdf:resource="#isMeasurementMetricOf"/>
<rdfs:range rdf:resource="#Measurement_Metric"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasMeasurementMetricInput">
<rdfs:domain rdf:resource="#Measurement_Metric"/>
<owl:inverseOf rdf:resource="#isInputToMeasurementMetric"/>
<rdfs:range rdf:resource="#Product"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPart">
<rdfs:domain rdf:resource="#SQAConcept"/>
<owl:inverseOf rdf:resource="#isPartOf"/>
<rdfs:range rdf:resource="#SQAConcept"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasPassword">
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasProcess">
<rdfs:domain rdf:resource="#Project"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasProduct">
<rdfs:domain rdf:resource="#Project"/>
<rdfs:range rdf:resource="#Product"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="Id">
<rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<Measurement_Metric rdf:ID="Input_Output_Utilization"/>
<Measurement_Metric rdf:ID="Input_Validity_Checking"/>
<Process rdf:ID="Inspection"/>
<Measurement rdf:ID="Installability"/>
<Measurement_Metric rdf:ID="Installation_Flexibility"/>
<Measurement rdf:ID="Interoperability"/>
<NonObservable_QA rdf:ID="interoperability"/>
<owl:ObjectProperty rdf:ID="invokes">
<rdfs:domain rdf:resource="#Process"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isA">
<rdfs:domain rdf:resource="#SQAConcept"/>
<owl:inverseOf rdf:resource="#has"/>
<rdfs:range rdf:resource="#SQAConcept"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isInputTo">
<rdfs:domain rdf:resource="#Product"/>
<owl:inverseOf rdf:resource="#hasInput"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isInputToMeasurementMetric">
<rdfs:domain rdf:resource="#Product"/>
<owl:inverseOf rdf:resource="#hasMeasurementMetricInput"/>
<rdfs:range rdf:resource="#Measurement_Metric"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isMeasurementMetricOf">
<rdfs:domain rdf:resource="#Measurement_Metric"/>
<owl:inverseOf rdf:resource="#hasMeasurementMetric"/>
<rdfs:range rdf:resource="#Measurement"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPartOf">
<rdfs:domain rdf:resource="#SQAConcept"/>
<owl:inverseOf rdf:resource="#hasPart"/>
<rdfs:range rdf:resource="#SQAConcept"/>
</owl:ObjectProperty>
<Process rdf:ID="Joint_Review"/>
<Measurement rdf:ID="Learnability"/>
<NonObservable_QA rdf:ID="maintainability"/>
<Measurement rdf:ID="Maintainability_Compliance"/>
<Measurement_Metric
   rdf:ID="Maintainability_Compliance_Metric"/>
<Process rdf:ID="Mangement_Review"/>
<Measurement rdf:ID="Maturity"/>
<Measurement_Metric rdf:ID="Mean_Time_Between_Failure"/>
<owl:ObjectProperty rdf:ID="measuredBy">
<rdfs:domain rdf:resource="#Quality_Attribute"/>
<owl:inverseOf rdf:resource="#measures"/>
<rdfs:range rdf:resource="#Measurement"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Measurement">
```

```
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<owl:Class rdf:ID="Measurement_Metric">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="measures">
<rdfs:domain rdf:resource="#Measurement"/>
<owl:inverseOf rdf:resource="#measuredBy"/>
<rdfs:range rdf:resource="#Quality_Attribute"/>
</owl:ObjectProperty>
<Technique rdf:ID="meeting"/>
<Measurement_Metric rdf:ID="Message_Clarity"/>
<owl:Class rdf:ID="Method">
<rdfs:subClassOf rdf:resource="#Resource"/>
<owl:disjointWith rdf:resource="#Technique"/>
<owl:disjointWith rdf:resource="#Tool"/>
</owl:Class>
<Requirement rdf:ID="NonFunctioanl_Requirement"/>
<owl:Class rdf:ID="NonObservable_QA">
<rdfs:subClassOf rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Observable_QA"/>
</owl:Class>
<owl:Class rdf:ID="Observable_QA">
<rdfs:subClassOf rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#NonObservable_QA"/>
</owl:Class>
<Measurement rdf:ID="Operability"/>
<Product rdf:ID="Operation_Report"/>
<NonObservable_QA rdf:ID="portability"/>
<Measurement rdf:ID="Portability_Compliance"/>
<Measurement_Metric rdf:ID="Portability_Compliance_Metric"/>
<Measurement_Metric rdf:ID="Precision"/>
<Product rdf:ID="Problem_Report"/>
<owl:Class rdf:ID="Process">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="producedBy">
<rdfs:domain rdf:resource="#Product"/>
<owl:inverseOf rdf:resource="#produces"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="produces">
<rdfs:domain rdf:resource="#Process"/>
<owl:inverseOf rdf:resource="#producedBy"/>
<rdfs:range rdf:resource="#Product"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Product">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<owl:Class rdf:ID="Project">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<Technique rdf:ID="prototyping"/>
<Process rdf:ID="Qualification_Testing"/>
<Process rdf:ID="Quality_Assurance"/>
<Product rdf:ID="Quality_Assurance_Plan"/>
<owl:Class rdf:ID="Quality_Attribute">
```

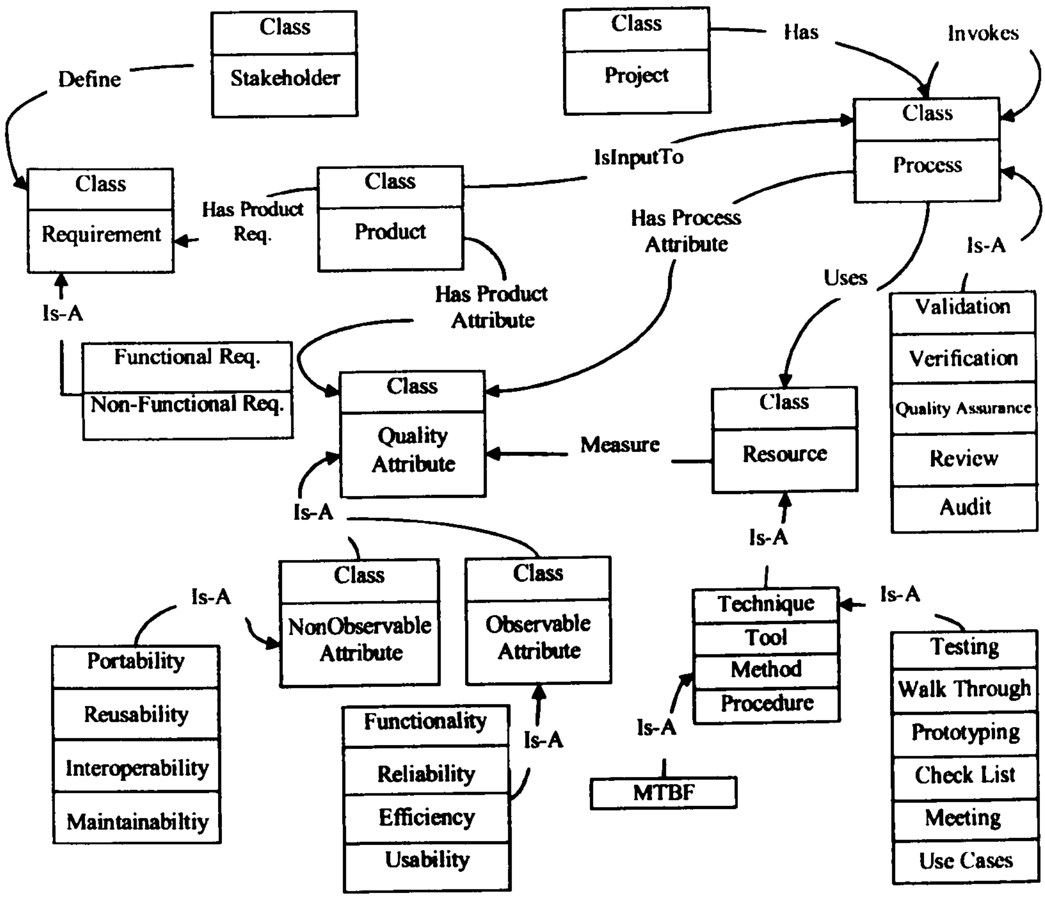
```xml
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Requirement"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<Measurement rdf:ID="Recoverability"/>
<Observable_QA rdf:ID="reliability"/>
<owl:Class rdf:ID="Requirement">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Resource"/>
</owl:Class>
<Product rdf:ID="Requirement_Specification"/>
<owl:Class rdf:ID="Resource">
<rdfs:subClassOf rdf:resource="#SQAConcept"/>
<owl:disjointWith rdf:resource="#Measurement"/>
<owl:disjointWith rdf:resource="#Measurement_Metric"/>
<owl:disjointWith rdf:resource="#Process"/>
<owl:disjointWith rdf:resource="#Product"/>
<owl:disjointWith rdf:resource="#Project"/>
<owl:disjointWith rdf:resource="#Quality_Attribute"/>
<owl:disjointWith rdf:resource="#Requirement"/>
</owl:Class>
<Measurement rdf:ID="Resource_Utilization"/>
<Measurement_Metric rdf:ID="Response_Time"/>
<Measurement_Metric rdf:ID="Restartability"/>
<Measurement_Metric rdf:ID="Restoreability"/>
<NonObservable_QA rdf:ID="reusability"/>
<Process rdf:ID="Review"/>
<Product rdf:ID="Review_Report"/>
<Measurement rdf:ID="Security"/>
<Technique rdf:ID="simulation"/>
<Product rdf:ID="Source_Code"/>
<owl:Class rdf:ID="SQAConcept"/>
<owl:DatatypeProperty rdf:ID="StartTime">
<rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>
<Process rdf:ID="SW_Design_Quality_Evaluation"/>
<Process rdf:ID="Technical_Review"/>
<owl:Class rdf:ID="Technique">
<rdfs:subClassOf rdf:resource="#Resource"/>
<owl:disjointWith rdf:resource="#Method"/>
<owl:disjointWith rdf:resource="#Tool"/>
</owl:Class>
<Product rdf:ID="Test_Cases"/>
<Measurement_Metric rdf:ID="Test_Coverage"/>
<Product rdf:ID="Test_Report"/>
<Product rdf:ID="Test_Specification"/>
<Measurement rdf:ID="Time_Behavior"/>
<owl:Class rdf:ID="Tool">
<rdfs:subClassOf rdf:resource="#Resource"/>
<owl:disjointWith rdf:resource="#Method"/>
<owl:disjointWith rdf:resource="#Technique"/>
</owl:Class>
<Measurement rdf:ID="Understandability"/>
<Measurement_Metric rdf:ID="Understandable_Input_Output"/>
<Measurement_Metric rdf:ID="Undoability"/>
<Observable_QA rdf:ID="usability"/>
<Technique rdf:ID="use_cases"/>
<owl:ObjectProperty rdf:ID="usedBy">
<rdfs:domain rdf:resource="#Resource"/>
<owl:inverseOf rdf:resource="#uses"/>
<rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
<Product rdf:ID="User_Manual"/>
<Product rdf:ID="User_Monitoring_Record"/>
<owl:ObjectProperty rdf:ID="uses">
<rdfs:domain rdf:resource="#Process"/>
<owl:inverseOf rdf:resource="#usedBy"/>
<rdfs:range rdf:resource="#Resource"/>
</owl:ObjectProperty>
<Process rdf:ID="Validation"/>
<Product rdf:ID="Validation_Plan"/>
<Process rdf:ID="Verification"/>
<Product rdf:ID="Verification_Plan"/>
<Technique rdf:ID="walk_through"/>
</rdf:RDF>
```

Appendix D: Previous Versions of the SQA Ontology

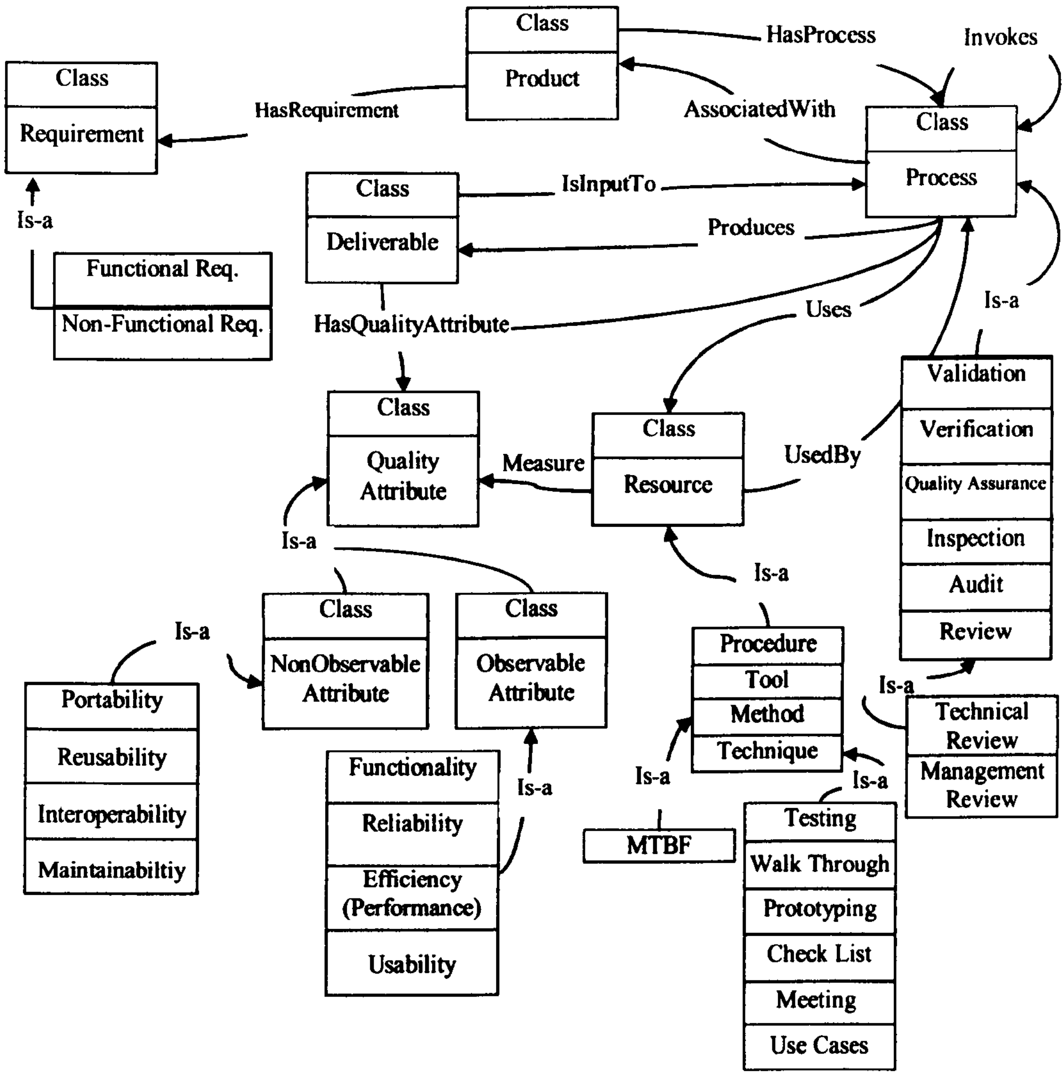
Ontology development is an iterative process where a preliminary ontology prototype is built and then polished with time. Here are examples of developed versions of the SQA ontology towards the final conceptual model shown in Fig. 5.3.



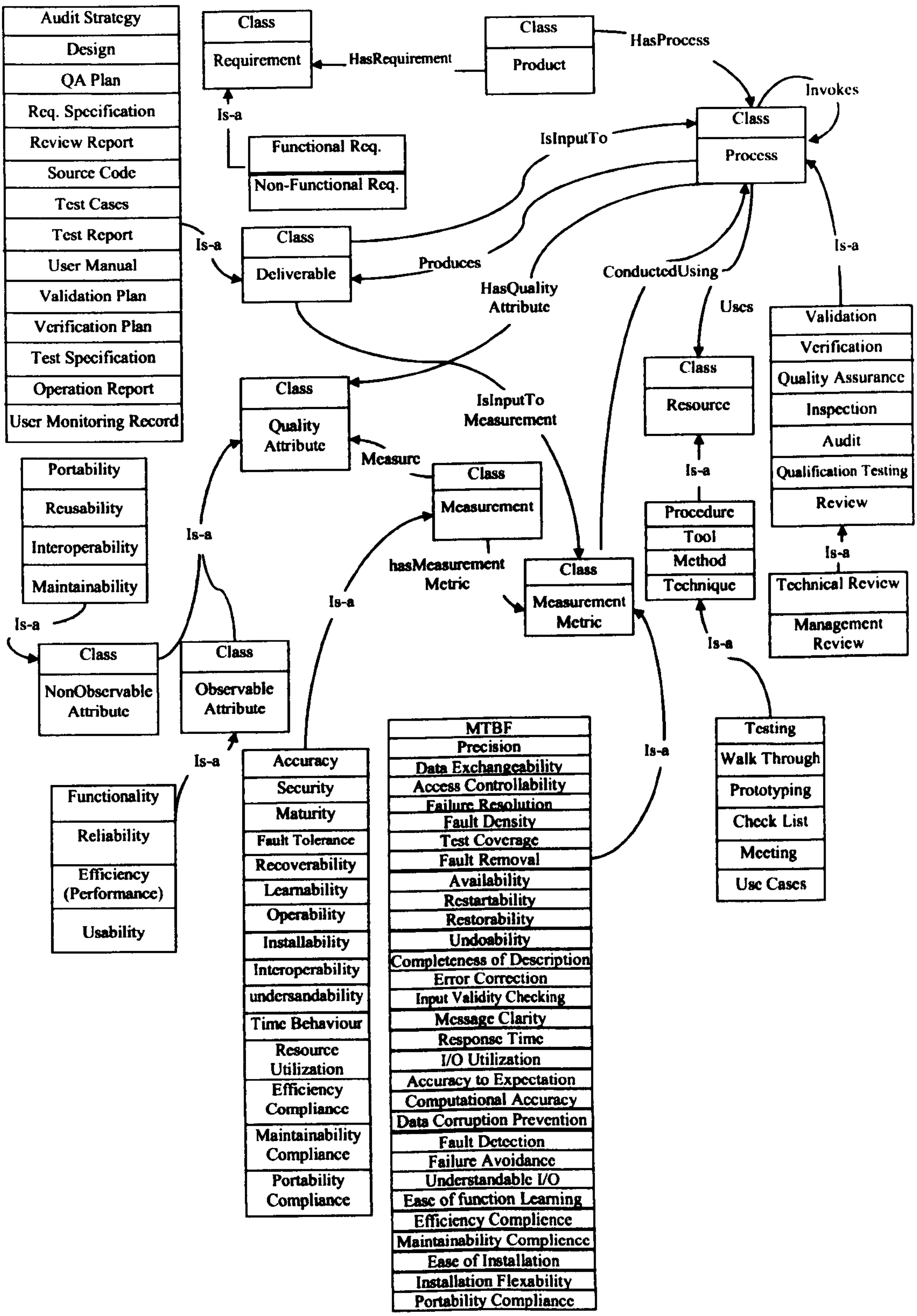
SQA Ontology (2009)



SQA Ontology (Bajnaid et al., 2010)



SQA Ontology (Bajnaid et al., 2011)



SQA Ontology (Bajnaid et al., 2012)

APPENDIX E: COMAPRAISION OF ISO/IEC 25010 AND ISO/IEC 9126

ISO/IEC 25010	ISO/IEC 9126	Notes
Functional suitability	**Functionality**	New name is more accurate and avoid confusion with other meanings of functionality
Functional completeness		Coverage of the stated needs
Functional Correctness	Accuracy	More general than accuracy
Functional appropriateness	Suitability	Coverage of the implied needs
	Interoperability	Moved to comaptability
	Security	Now a characteristic
Performance effieciency	**Efficiency**	Renamed to avoid conflicting with definition of efficiency in ISO/IEC 25062
Time behaviour	Time behaviour	
Resource utilization	Resource utilization	
Capacity		New characteristic (particularly relevant to computer systems)
Compatability		New characteristic
Co-existance	Co-existance	Moved from Portability
Interoperability		Moved from Functionality
Reliability	**Reliability**	
Maturity	Maturity	
Availability		New subcharacteristic
Fault tolerance	Fault tolerance	
Recoverability	Recoverability	
Portability	**Portability**	
Adaptability	Adaptability	
Installability	Installability	
	Co-existance	Moved to comapatability
Replaceability	Replaceability	

ISO/IEC 25010	ISO/IEC 9126	Notes
Security	**Security**	No previous subcharacteristics
Confidentability		
Integrity		
Non-repudiation		
Accountability		
Authenticity		
Usability		Implicit quality issue made explicit
Appropriateness recognizabiluty	Understandability	New name is more accurate
Learnability	Learnability	
Operability	Operability	
User error protection		New subcharacteristic (to achieve freedom from risk)
User interface aesthestics	Attractiveness	New name is more accurate
Accessability		New subcharacteristic
Maintainability	**Maintainability**	
Modularity		New subcharacteristic
Reusability		New subcharacteristic
Analysability	Analysability	
Modifiability	Stability	More accurate name combining changability and stability
Testability	Testability	

APPENDIX F: INTRODUCTORY DOCUMENT TO THE ONTOLOGY ASSESSMENT QUESTIONNAIRE

About a Questionnaire for the Evaluation of the Software Quality Assurance Ontology

Nada Bajnaid
PhD Student at the Faculty of Life and Computing Sciences
London Metropolitan University, UK

Software is a key element of the modern computing systems (from mobile phones to supercomputers) and there is a need for high standards in the educating people who are involved in its development. It becomes especially critical when there are special requirements for high quality software. One problem in the teaching of Software Engineering as a discipline is the use of textbooks as the main source of knowledge. Moreover, the discipline may be studied as separate modules/courses that may be not coordinated in terms of consistency and completeness. This may intern that meaning of terms is inter-related and/or overlapped.

There was an effort by different bodies to develop Software Engineering standards followed by the forming of the ISO/IEC Joint Technical Committee 1 (JTC1) workgroup in order to guarantee consistency and coherency among standards. The IEEE Computer Society and the ISOJTC1-SC7 agreed to harmonize terminology among their standards. However, there is still no single standard which embraces the whole Software Quality Assurance (SQA) knowledge. Because of that, there are various vocabularies to describe the SQA knowledge in learning context including textbooks. In addition, Software Engineering teachers have different backgrounds, use different languages and/or jargons which motivate additional research related to SQA teaching.

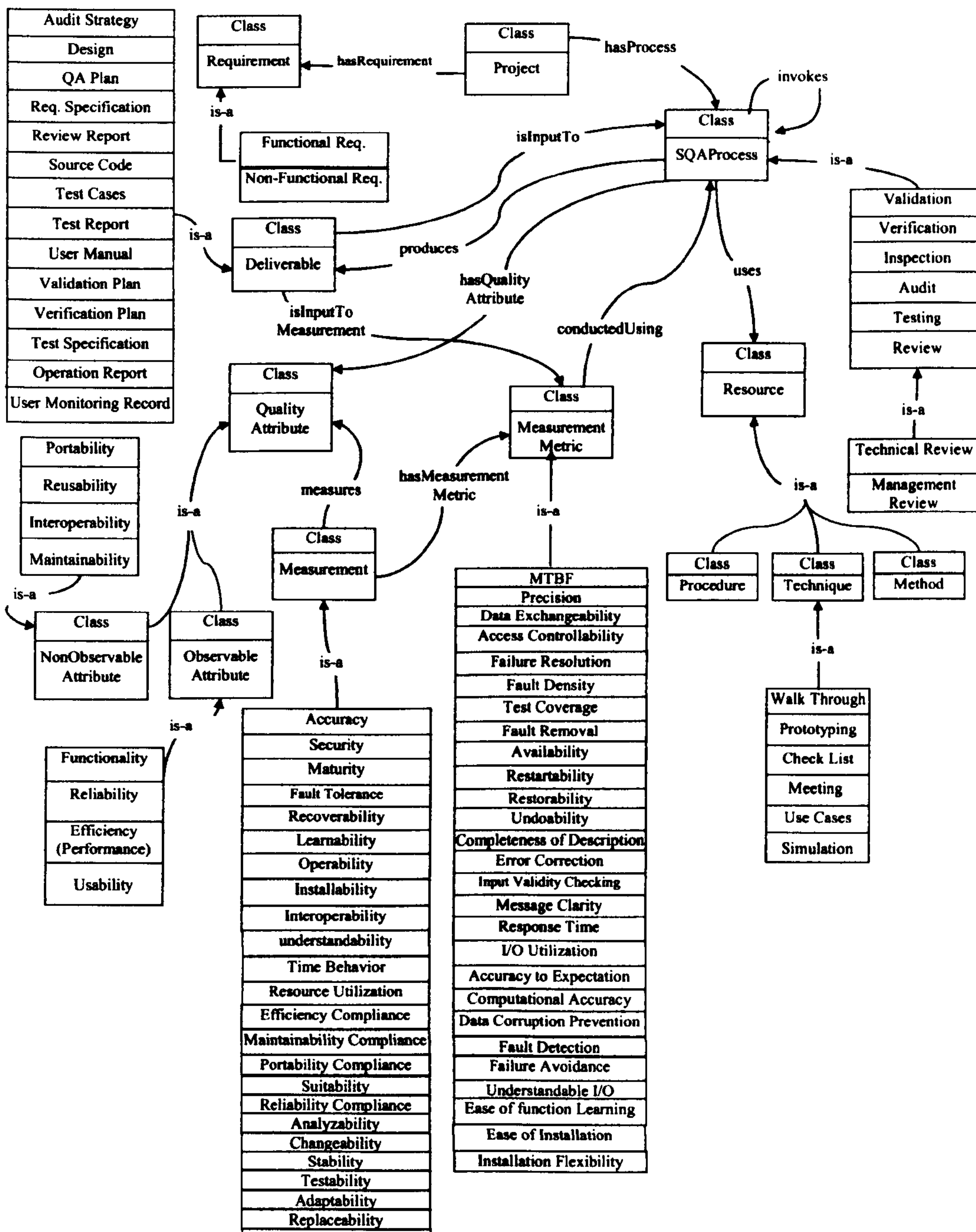
With the new technological advances and the use of e-learning techniques, ontologies play key role in supporting semantic knowledge representation and thus enhancing e-learning experience. It allows structural annotation of electronic resources with semantic information providing machine-understandable contents.

Application-Based ontology evaluation was used where an ontology-based context-aware prototype of SQA e-learning system was designed and implemented to guide students and practitioners about a process of development of the SQA compliant software. The system can sense the learner's current stage of the SQA process and show relevant Learning Objects (LOs) that deal with SQA aspects. There are 200 LOs available to the learner. The system filters out LOs based on the individual learner's usage of the system (profile) and ontology-based reasoning. The Application-Based ontology evaluation is used to measure practical aspects of ontology deployment.

The primary source of the SQA ontology given below is the SWEBOK guide (2004), in addition to that, ISO and IEEE standards (ISO 9126, IEEE 12207, IEEE 610.12, IEEE 00100, SWEBOK 2004, PMBOK 2008, CMMI v1.2) were used and from them relevant terminology was extracted. The following figure[1]illustrates the formal

[1]Bajnaid N., Benlamri R. and Cogan B. (2012), "An SQA e-Learning System for Agile Software Development", Proc. of the Fourth International Conference on Networked Digital Technologies, Dubai, UAE, April 24-26, 2012. Communications in Computer and Information Science(CCIS 7899) Series of Springer LNCS – in press.

structure and the various relationships used to define all SQA processes in the software development process. The figure shows the main SQA concepts as OWL classes where the arrows represent relationships (OWL object properties) between domain classes (the head of the arrow) and range classes (the tail of the arrow). The is-a property relates an SQA concepts with its instances (OWL individuals).



In the figure we eliminate the number of instances of the SQA measurements and metrics for simplicity. While in the OWL ontology model we try to cover almost all SQA measurements and metrics. Applicable measurements and metrics may be not limited to the ones listed in the ontology. Other metrics for particular purposes may be added.The aim of this questionnaire is to validate the ontology quality and usefulness. The ontology validation ensures consistency by avoiding contradictory information. In addition, ontology clarity is to be validated by referring to how well the proposed meanings are communicated. Finally, the ultimate goal was to develop an ontology that faithfully models the SQA discipline as practiced in the software development life cycle, with further emphasis on SQA measurements and metrics.

References

[1] Bajnaid N., Cogan B. and Al-Nuaim H. (2008), "Software quality ontology for teaching: a development methodology's issues". Proc. 5th International Innovations in Information Technology (IIT 2008), pp. 352–356 (2008) Indexed by IEEEXplore 10 Feb 2009, ISBN 978-1-4244-3396-4
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04781698

[2] Bajnaid N., Benlamri R. and Cogan B. (2010), "Ontology-Based E-Learning System for SQA Compliant Software Development", Proc. of the 2nd Int. Conf. on the Application of Digital Information and Web Technologies – ICDIWT'2010, Istanbul, Turkey, pp.85-91, July 12-14, 2010.

[3] Bajnaid N., Benlamri R. and Cogan B. (2011), "Context-Aware SQA E-learning System", Proc. of the Sixth International Conference on Digital Information Management ICDIM 2011, Melbourne, Australia, 26-28 Sept., 2011.

[4] Boehm B., Chulani S., Verner J., and Wong B., (2009). Seventh workshop on Software Quality,.ICSE-Companion 2009. 31st International Conference on Software Engineering – Companion, 16-24 May 2009, pp.449-450.

[5] Calero, C., Ruiz, F. and Piattini, M.: Ontologies in Software Engineering and Software Technology, Springer (2006).

[6] Wille C., Dumke R., Abran A., and Desharnais J. (2004). E-learning infrastructure for Software Engineering educations: steps on ontology modeling for SWEBOK, Proceedings of the IASTED International Conference on Software Engineering, 2004, pp. 520-525.

[7] Saiedian H. and Weide B., (2005). The new context for Software Engineering education and training, The Journal of Systems and Software 74, pp. 109-111.

[8] IEEE standard glossary of software engineering terminology: ANSI/IEEE std 610.12-1990. In: IEEE Software Engineering Standards, vol. 1, Customer and Terminology Standards. IEEE, Inc. 1999.

[9] ISO/IEC 12207, IEEE Std 12207-2008: System and Software Engineering – Software Life Cycle Processes

[10] ISO/IEC 9126:1991, Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use.

[11] Mendes, O., and Abran, A., (2004). Software Engineering Ontology: A Development Methodology, Position Paper, Metrics News 9:1, pp. 68-76.

[12] PMBOK, A Guide to the Project Mangement Body of Knowledge, Third Edition, ANSI/PMI 99-001-2004.

[13] SWEBOK, (2004). Guide to the Software Engineering Body of Knowledge, ed. Bourque P., and Dupuis R. IEEE Computer Society Press, 2004. Available at: http://www.swebok.org

APPENDIX G: THE ONTOLOGY ASSESSMENT QUESTIONNAIRE

The aim of this survey is to evaluate a Software Quality Assurance (SQA) ontology model that has been created in a PhD project. The SQA ontology was developed based on international standards (ISO 9126, IEEE 12207, IEEE 610.12, IEEE 00100, SWEBOK 2004, PMBOK 2008, CMMI v1.2, and ANSI/ISO/ASQ Q9000-2000). The results of the survey will be used to assess and evaluate the developed ontology in this research. It is assumed that respondent has some knowledge in the SQA domain.

A. About respondent

Respondent expertise: please rate your expertise in the SQA domain

1. Null 2. Poor 3. Average 4. Above average 5. Excellent

Respondent expertise: please rate your expertise in the ontology domain

1. Null 2. Poor 3. Average 4. Above average 5. Excellent

Are you now (or ever been) involved into the teaching of Software Engineering?

1. Yes 2. No

Do you think ontology can be useful for teaching SQA?

1. Strongly disagree
2. Disagree
3. Borderline
4. Agree
5. Strongly agree

B. On a scale of 1-5 (where 5 = strongly agree and 1 = strongly disagree), please indicate how will you agree in the following statements about the developed SQA conceptual model

Quality Criteria	Statement	1	2	3	4	5
Completeness	The model covered the major concepts of the SQA domain					
Structure	The taxonomy ("is-a" relationship) is presented correctly in the model					
	Other relationships among the SQA concepts are presented correctly in the model (invokes, produces, measures, uses, ensures...etc.)					
Clarity	There are some redundant concepts in the model					
	There are some ambiguous concepts in the model					
Consistency	The ontology is logically consistent Ex: X instance of classes A and B, but A and B are disjoint This is a contradiction					
Extendibility	New terms can be introduced without the need to revise existing structure of the model					

C. **Non-relevant concepts/terms to be removed from the model if any. Why you think it should be removed?**

D. **Suggested concepts/terms to be added to the model if any. Where?**

Thank you for your time in completing this questionnaire

APPENDIX H: INDIVIDUAL PARTICIPANTS' RESPONSES TO THE QUESTIONNAIRE

Individual Participants' Responses to Parts I and II of the Questionnaire

Respondent	Responses to Part II Statements as in Table 6.4							Responses to Part I Statements			
	2.1	2.2	2.3	2.4	2.5	2.6	2.7	1.1 Experience on SQA	1.2 Experience on Ontology	1.3 Involved in Teaching SQA	1.4 Usefulness of ontology in teaching SQA
1	5	3	3	3	3	4	4	Excellent	Average	No	Strongly agree
2	4	4	5	3	3	4	5	Excellent	Above average	Yes	Agree
3	4	5	5	2	1	5	5	Above average	Average	Yes	Agree
4	4	5	4	3	4	4	5	Above average	Average	Yes	Agree
5	5	5	5	5	5	5	5	Above average	Above average	Yes	Strongly agree
6	4	3	4	4	3	3	4	Average	Above average	Yes	Agree
7	4	4	3	4	4	3	4	Above average	Poor	No	Borderline
8	3	5	4	4	4	4	5	Excellent	Above average	No	Agree
9	3	3	3	4	4	3	2	Above average	Average	Yes	borderline
10	4	4	3	3	3	3	3	Average	Average	No	Agree
11	4	4	4	1	2	4	4	Average	Average	No	Agree
12	4	3	3	3	5	4	4	Poor	Average	Yes	Borderline
13	4	3	4	3	1	5	2	Excellent	Average	Yes	Agree
14	4	4	4	2	2	3	4	Above average	Average	Yes	Agree
15	4	4	4	3	3	4	4	Above average	Average	Yes	Agree

Individual Participants' Responses to Part I, III, and IV of the Questionnaire

Resp.#	Comments to Part III	Comments to Part IV	Responses to Part I Statements as in Table 6.5			
			1.1	1.2	1.3	1.4
1	Redundant terms: - Use single term for Precision/ Accuracy to Exception/ Computational accuracy - Use one term for Ease of installation and flexibility of installation Revise the order of stages in the domain class that starts with Audit strategy. Design should come after Review Report.	We recommend to refer to FURPS model which was developed by Robert Grady at Hewlett Packard. The FURPS model includes the following domain classes: - Functionality It includes auditing, licensing	Excellent	Average	No	Strongly Agree
2	Measurement and measurement metrics are too detailed I think	Maybe a software quality plan, It might be there but I couldn't see it	Excellent	Above average	Yes	Agree
3	Null	Null	Above average	Average	Yes	Agree
4	-	Issues: 1. In the deliverable, you may consider adding installation manual, configuration management plan (or project plan) 2. It is not clear the difference between Measurement Metrics and Measurement 3. You may consider adding Tools to the procedure, techniques and method. 4. You may consider adding benchmarking to the process	Above average	Average	Yes	Agree
5	-	-	Above average	Above average	Yes	Strongly agree
6	-	-	Average	Above	Yes	Agree

				average		
7	-	-	Above average	Poor	No	Borderline
8	Given that the model does not contain all SQA metrics, it is really hard to provide a feedback on non-relevant concepts. However, if the evaluation procedure is planned to be user-centered, then non-observable attributed such as maintainability should be omitted from the model because only experts (e.g. developers) are able to evaluate that kind of attributes. Besides, there is a redundancy between the name of the attribute and a measurement (e.g. interoperability).	Although the aim of the model is to be applicable to all software domains, it should be extended in order to be comprehensive and thus cover characteristics of all software types. In that respect, I suggest that you add a class related to the type of the software platform (e.g. web, desktop, mobile). In addition, you should consider to add attributes related to the information quality (e.g. ISO/IEC 25012, 2008) as well as attributes related to both quality and quality in use evaluation as proposed in ISO/IEC 25010 (2011). Finally, you should add a class related to the software life cycle.	Excellent	Above average	No	Agree
9	There is overlap/ redundancy between the items listed for Measurement and Measurement Metric.	Software Quality Assurance is a very broad area, and this ontology can be improved both in clarity and breadth. Testing related concepts should be clearer - black box and white box, unit and system level, regression and enhancement tests, and other attributes. Also, there are a range of quality improvement related investment areas that are either missing or not clear in this model: Organizations can invest in	Above average	Average	Yes	borderline

		requirements reviews and various means for reducing ambiguity, architectural analysis, design reviews, code walkthroughs, static code analysis/ smell detection as well as various testing activities - many of these seem absent in the model. Finally, there are issues around development processes (agile vs. waterfall/ plan based) and their practices that have SQA implications. I would recommend reading the concepts in Daniel Galin's book "Software Quality Assurance - From Theory to Implementation".				
10	-	It's stated that the deliverables are the input to the Software Quality Assurance process which also produce the deliverables. Maybe, it can be written which particular deliverables are inputs and which are the outputs.	Average	Average	No	Agree
11	-	-	Average	Average	No	Agree
12	without context I find it hard to distinguish between, say, observable and non-observable attributes. Why "usability" is an observable attribute, and 'interoperability" is not? Why "installability" or "adaptability" are on the list of measurement's attributes, and not, for instance, quality ones? Relationships seem to be somewhat arbitrary, e.g. I cannot vouch, for the validity of this	Where is software configuration management in all this? Version control, for example? Consistency of the artifacts comprising a system? Where is documentation? What about architecture - software architecture, system architecture and enterprise architecture?	Poor	Average	Yes	Borderline

	statement that follows from the model: "management review is audit,testing and validation"..					
13	For teaching purpose, lower level concepts should be dropped. Current model, in a way, represents whole software engineering.	There are different views on SQA. E.g., Pressman mentions 7 aspects of SQA, and some are missing in the model.	Excellent	Average	Yes	Agree
14	My proposals: 1. To use the term "quality characteristic" instead of "quality attribute" because it is used in ISO standards. 2. To use the term "measure" instead of "measure metric" because: In 2002, the ISO/IEC JTC1 sub-committee SC7 – Systems and Software Engineering – replaced the term "metric" by "measure" to align its vocabulary with the one used in metrology. This thesis will use the term measure whenever possible. (See "Software Quality Model Requirements for Software Quality Engineering"//Marc-Alexis Côté M., Witold Suryn, Elli Georgiadou - http://profs.etsmtl.ca/wsuryn/research/SQE-Publ/Quality%20model_requirements.%20SQM2006.pdf). Such terms are used in the new standards from ISO 25000 series. W. Suryn is secretary of technical committee JTC 1/SC 7 Software and systems engineering.	Why your "Quality Attributes" are other than "Quality Characteristics" in ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.	Above average	Average	Yes	Agree
15	-	-	Above average	Average	Yes	Agree

APPENDIX I: SAMPLE USER PROFILE IN XML

"Testing"

Appendix J: Java Code of SQAES

Login.html

```
<html>

<head>
<title> SQA System: User Access </title>
</head>

<bodybgColor="Gainsboro">
<formaction="SQASystem"method="get">

<Center>

<h2> SQAES: Your SQA E_Learning System <br><br></h2>
<h4> Please Enter Your Access Information  </h4>

        User Name :
<inputtype="text"name="userName"size=20><br>
        Password    :
<inputtype="password"name="passWord"size=20><br><br>
<inputtype="submit"value="Sign In"size=40>
</Center>

</form>
</body>
</html>
```

Exit.html

```
<html>

<head>
<title> Exit SQA System </title>
</head>

<bodybgColor="Gainsboro">

<Center>
<br><br><br><br>
<h3> Thank you for using <br><br></h3>
<h2> The Personalized SQA Learning System  </h2>
</center>

</body>
</html>
```

SQAServlet.java

```
package SQA.Learning.System;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import java.util.Iterator;
importjava.util.Iterator;
importjava.util.List;
import java.util.Vector;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import RelatedLOs.UpdateXML;


@WebServlet ("/SQASystem")
//@WebServlet(urlPatterns={"/SQA"})

publicclassSQAServletextends HttpServlet {
   String userName = null;
  ReadingOntology model = new ReadingOntology();
  intsessionNum;
@Override

protectedvoid doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
        String passWord = null;

try
        {
userName = request.getParameter("userName");
```

```
                passWord =
request.getParameter("passWord");
if (userName != null)
          {
                     PrintWriter out =
response.getWriter();
                     response.setContentType("text/html");

                     out.println("<html>");
                     out.println("<META http-
equiv='Content-Style-Type' content='text/css'>");


                     out.println("<body  bgColor =
'Gainsboro'>");
// the header div
                     out.println("<div id = 'header' style
= 'background-color:orange; margin-bottom:0; margin-
top:0; margin-left:0; margin-right:0'>");
                     out.println ("<Center><h3
color:Black>SQAES: Your SQA Learning
System</h3></Center>");

                                 out.println("<p><align
='right' font-size = '10px'><a href=\"Exit.html\">Sign
Out</a></p><br>");
                     out.println("</div>");

if (model.userExist(userName, passWord)){

                        out.println ("<html bgColor =
'Gainsboro'>");
out.println ("<Center><h4>Welcome " + userName +
"</h4>");              out.println ("");
out.println ("Please type your query: ");

                        response.getWriter().print("<form
name = 'searchform' action = './SQASystem' method =
'post'>");
                                       out.println ("<input type = 'text'
name = 'query' size = 25></br>");
                                       out.println ("<input type =
'submit' value = 'Submit Query'></Center>");
                                       out.println ("<br><br><br><center>
You may query an SQA process, quality attribute,
deliverable documents,");
                                       out.println ("<br>quality
measurement, metric, or qulaity supported
resource</center>");
                                       out.println("</form>");
                                       out.println ("</html>");

                                    }
else {
                                       out.println ("<html>");
                                       out.println("<h4>Wrong username or
password!!</h4>");
                                       out.println ("</html>");
                                    }
                                    out.println("</div>");
                                    out.println("</body>");
                                    out.println("</html>");
                              }
                        }

catch (Exception e){
               response.getWriter().println ("Error in
Entry");
               e.printStackTrace();
          }
     }  // doGet

//==============================

protectedvoid doPost (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {

 String name= userName;
Enumeration parmNames = request.getParameterNames();
```

```
        String  parmName;     // to hold a single
parameter name
booleanenumEmpty = false;
        response.setContentType("text/html");
//send data to the client through a printwriter
        java.io.PrintWriter out =
response.getWriter();

        out.println("<html>");

        out.println("<body>");
        out.println("<META http-equiv='Content-Style-
Type' content='text/css'>");
        out.println("<body  bgColor = 'Gainsboro'>");


        out.println("<div id = 'container' style =
'background-color:Gainsboro>");    //the first div for
the whole page

while (parmNames.hasMoreElements())
        {
            parmName = (String)
parmNames.nextElement();


            String queryStr =
request.getParameter("query");
    queryStr =
org.apache.commons.lang.StringUtils.replace(queryStr,"
","_");


            UpdateXML userProfile = new UpdateXML
(name);
sessionNum = userProfile.addSession();
            userProfile.addConcept(sessionNum,
queryStr);
            userProfile.writeToFile(name);
model.coreLearningObjects(queryStr, name);
Vector coreLOs = model.getCoreLOs();
Vector URLs = model.getURLs();
Vector ConsumedLOs = model.getConsumedLOs();
```

```
Vector ConsumedURLs = model.getConsumedURLs();
Vector relatedQAs = model.getRelatedQAs();
Vector isQAof = model.getIsQAof();
Vector invokedProcess = model.getInvokedProcess();
Vector usedResources = model.getUsedResources();
Vector usedByProcess = model.getUsedByProcess();
VectorrelatedConcepts = model.getRelatedConcepts();
Vector inputs = model.getInputs();
Vector isInputTo = model.getIsInputTo();
Vector deliverables = model.getDeliverables();
Vector producedBy = model.getProducedBy();
Vector associatedProcess =
model.getAssociatedProcess();
Vector measures = model.getMeasures();
Vector measuredBy = model.getMeasuredBy();
Vector measurements = model.getMeasurements();
Vector metrics = model.getMetrics();
Vector measurementInputs =
model.getMeasurementInputs();
Vector inputToMeasurements =
model.getInputToMeasurements();

// the header div
            out.println("<div id = 'header' style =
'background-color:orange; margin-bottom:0; margin-
top:0; margin-right:0; margin-left:0'>");
            out.println ("<Center><h3
color:Black>SQAES: Your SQA Learning
System</h3></Center>");
            out.println("<p><align ='right' font-size
= '10px'><a
href='javascript:document.form['searchform'].submit();
'>New Search</a>");
            out.println("<a href=\"Exit.html\">Sign
Out</a></p>");
            out.println("</div>");  // the header div

// the left div
            out.println("<div id = 'left' style =
'font-size:18px; font-family:verdana; width:450px;
float:left'>");
```

```java
if (coreLOs.isEmpty())
                out.println("Sorry! No learning
resources found for your query");
else
          {
 out.println ("<font color =
'brown'><left><strong><i>" + "Suggested Learning
Resources About " +queryStr+
"</i></strong><br></font>");
Iterator itr1 = coreLOs.iterator();
Iteratoritr3 = coreLOs.iterator();
Iterator itr2 = URLs.iterator();

while (itr1.hasNext())
                {
    String location = (String)itr2.next();
                String lo =
itr1.next().toString();
                out.println("<a target = '_blank'
href=\"http://localhost:8080/SQAES/consumedLO?name="+n
ame+"&lo="+lo+"&location="+location+"&sessionNum="+Int
eger.toString(sessionNum)+"&Query="+queryStr+"\">"+lo+
"</a><br>");
                }
          }

if (! ConsumedLOs.isEmpty())
          {
 out.println("<br>");
 out.println ("<font
color='green'><left><strong><i><u>You have visited the
following learning
resources</u></i></strong></font><br>");
Iterator itr1 = ConsumedLOs.iterator();
Iteratoritr3 = coreLOs.iterator();
Iterator itr2 = ConsumedURLs.iterator();

while (itr1.hasNext())
                {
 String location = (String)itr2.next();
                String lo =
itr1.next().toString();
                out.println("<a target = '_blank'
href=\"http://localhost:8080/SQAES/consumedLO?name="+n
ame+"&lo="+lo+"&location="+location+"&sessionNum="+Int
eger.toString(sessionNum)+"&Query="+queryStr+"\">"+lo+
"</a><br>");
                }
          }
          out.println("<br><font color =
'brown'><Strong><i>" + " You may also read about" +
"</Strong></i></font>");
if (! inputs.isEmpty())
          {
Iterator itr = inputs.iterator();
                out.println("<br><Strong><i>" +
"Inputs required by " + queryStr +
"</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
     buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
     out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
          }
if (! measurementInputs.isEmpty())
          {
Iterator itr = measurementInputs.iterator();
                out.println("<br><Strong><i>Inputs
required by " + queryStr + "</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
```

```
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }
if (! deliverables.isEmpty())
            {
Iterator itr = deliverables.iterator();
                out.println("<br><Strong><i>" + "
Deliverables produced by "+ queryStr
+"</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+concept+"</a><br>");
                }
            }
            out.println("</div>");    //the left div

// the right div
            out.println("<div id = 'right' style =
'type:text/css; font-family:verdana; font-size:18px;
width:450px; float:right'>");
if (! invokedProcess.isEmpty())
            {
Iterator itr = invokedProcess.iterator();
                out.println("<br><Strong><i>" +
"Processes invoked by " +queryStr  +
"</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }
if (! usedResources.isEmpty())
            {
Iterator itr = usedResources.iterator();
                out.println("<br><Strong><i>" +
"Resources used by " +queryStr + "</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }


if (! producedBy.isEmpty())
            {
Iterator itr = producedBy.iterator();
                out.println("<br><Strong><i>" +
"Processes that may produce " + queryStr + "
</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
```

```java
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! isInputTo.isEmpty())
            {
Iterator itr = isInputTo.iterator();
                out.println("<br><Strong><i> Processes
that require " + queryStr + " as input
</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! measures.isEmpty())
            {
Iterator itr = measures.iterator();
                out.println("<br><Strong><i> Quality
attributes that are measured by " + queryStr +
"</Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! associatedProcess.isEmpty())
            {
Iterator itr = associatedProcess.iterator();
                out.println("<br><Strong><i>Processes
used to conduct " + queryStr + " </Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! inputToMeasurements.isEmpty())
            {
Iterator itr = inputToMeasurements.iterator();

out.println("<br><Strong><i>Measurements require " +
queryStr + " as input </Strong></i><br>");
while (itr.hasNext())
                {
 String concept = (String)itr.next();
 String buffer = concept;
```

```
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                    }
              }
if (! relatedQAs.isEmpty())
              {
Iterator itr = relatedQAs.iterator();
                 out.println("<br><Strong><i>Quality
attributes that are enforced by " + queryStr+
"</Strong></i><br>");
while (itr.hasNext())
                    {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                    }
              }

if (! isQAof.isEmpty())
              {
Iterator itr = isQAof.iterator();
                 out.println("<br><Strong><i>Processes
used to ensure " + " The " +queryStr+" quality
attribute  </Strong></i><br>");
while (itr.hasNext())
                    {
 String concept = (String)itr.next();
 String buffer = concept;

        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                    }
              }

if (! measuredBy.isEmpty())
              {
Iterator itr = measuredBy.iterator();

out.println("<br><Strong><i>Measurements used to
measure the " + queryStr + "quality attribute
</Strong></i><br>");
while (itr.hasNext())
                    {
 String concept = (String)itr.next();
 String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                    }
              }

if (! usedByProcess.isEmpty())
              {
Iterator itr = usedByProcess.iterator();
                 out.println("<br><Strong><i>Processes
use "+queryStr+" as a resource </Strong></i><br>");
while (itr.hasNext())
                    {
 String concept = (String)itr.next();
 String buffer = concept;
```

```
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
    out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! measurements.isEmpty())
            {
Iterator itr = measurements.iterator();
              out.println("<br><Strong><i>Metrics of
the " +queryStr + " measurement </Strong></i><br>");
while (itr.hasNext())
                {
  String concept = (String)itr.next();
  String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }

if (! metrics.isEmpty())
            {
Iterator itr = metrics.iterator();
              out.println("<br><Strong><i>Metrics
used by " + queryStr + "</Strong></i><br>");
while (itr.hasNext())
                {
//out.println ("<small>- " + (String) itr.next() +
"</small><br>");
  String concept = (String)itr.next();
  String buffer = concept;
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
        out.println("<a target='_blank'
href=\"http://localhost:8080/SQAES/LOServlet?queryStr=
"+concept+"&"+"userName="+name+"&sessionNum="+Integer.
toString(sessionNum)+"\">"+buffer+"</a><br>");
                }
            }


//the right div
              out.println("</div>");

        }  // while
        out.println("</div>");    // the first div
        out.println("</body>");
        out.println("</html>");


    } // doPost


} // AccessServlet
```

ReadingOntology.java

```
package SQA.Learning.System;

importjava.io.File;
importjava.io.FileInputStream;
importjava.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
importjava.io.InputStreamReader;
importjava.io.Reader;
importjava.util.Collection;
import java.util.Iterator;
importjava.util.List;
```

```
import java.util.Vector;

import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;
import com.hp.hpl.jena.util.FileManager;

importedu.stanford.smi.protegex.owl.ProtegeOWL;
importedu.stanford.smi.protegex.owl.jena.creator.JenaC
reator;
importedu.stanford.smi.protegex.owl.model.OWLModel;
importedu.stanford.smi.protegex.owl.swrl.*;
importedu.stanford.smi.protegex.owl.swrl.bridge.*;

publicclass ReadingOntology {
 private        OntModel    model;
 final          String      source = "http://www.owl-
ontologies.com/SQOntology#";
 final static StringsourceFile =
"C:/Users/vaio/Desktop/SQA Ontologies/SQOntology.owl";
 final     String       ns = sourceFile + "#";
 final     String       emptyString = "";
 final     String       ontNS = "http://www.owl-
ontologies.com/SQOntology#";
 privateVectorcoreLOs = newVector();
 privateVectorURLs = newVector();
 private  Vector  ConsumedLOs = newVector();
 private  Vector  ConsumedURLs = newVector();
 privateVectorrelatedQAs = newVector();
 private  Vector  isQAof = newVector();
 privateVectorinvokedProcess = newVector();
 private   Vector  usedResource = newVector();
 private   Vector  usedByProcess = newVector();
 privateVectorrelatedConcepts = newVector();
 private   Vector  relatedLOs = newVector();
 private   Vector  deliverables = newVector();
 private  Vector  producedBy = newVector();
private   Vector  inputs = newVector();
private   Vector  isInputTo = newVector();
private  Vector  measures = newVector();
private  Vector  measuredBy = newVector();
private   Vector  associatedProcess = newVector();
private   Vector  metrics = newVector();
private   Vector  measurements = newVector();
private   Vector  measurementInputs = newVector();
private   Vector  inputToMeasurements = newVector();

@SuppressWarnings("deprecation")

public ReadingOntology (){

  InputStream in = FileManager.get().open(sourceFile);

  if (in == null)
   thrownew IllegalArgumentException("File not
Found");

  model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,
null);
  model.read(in, "");

  try {
    in.close();
   } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
   }

    }
 //=============GETTERS====================
 publicVector getMetrics () {
  returnmetrics;
    }

 publicVector getMeasurements () {
  returnmeasurements;
    }
```

```
publicVector getMeasurementInputs () {
  returnmeasurementInputs;
    }

publicVector getConsumedLOs () {
  returnConsumedLOs;
    }

publicVector getConsumedURLs () {
  returnConsumedURLs;
    }

publicVector getInputToMeasurements() {
  returninputToMeasurements;
    }

publicVector getAssociatedProcess () {
  returnassociatedProcess;
    }

publicVector getMeasures () {
  returnmeasures;
    }

publicVector getMeasuredBy () {
  returnmeasuredBy;
    }

publicVector getDeliverables() {
  returndeliverables;
    }

publicVector getProducedBy () {
  returnproducedBy;
    }

publicVector getInputs() {
  returninputs;
    }
```

```
publicVector getIsInputTo() {
  returnisInputTo;
    }

publicVector getRelatedQAs() {
returnrelatedQAs;
    }

publicVector getIsQAof () {
  returnisQAof;
    }

publicVector getUsedResources() {
  returnusedResource;
    }

publicVector getUsedByProcess() {
  returnusedByProcess;
    }

publicVector getRelatedConcepts() {
  returnrelatedConcepts;
    }

publicVector getRelatedLOs() {
  returnrelatedLOs;
    }

publicVector getInvokedProcess () {
returninvokedProcess;
    }

publicVector getCoreLOs() {

returncoreLOs;
    }

publicVector getURLs () {
  returnURLs;
    }
```

```
//===============================================

publicboolean userExist (String userName, String
passWord) {
  boolean exist = false;
        OntClass developer = model.getOntClass(ontNS
+ "Developer");

 Iterator ind = developer.listInstances();

 if (ind != null)
 while (ind.hasNext() && !exist){

        String uri = ind.next().toString();
        String name =
uri.substring(ontNS.length());

 if (name.equalsIgnoreCase(userName)) {

            StmtIterator itr =
model.listStatements();
 while (itr.hasNext() && !exist) {
                Statement stmt =
itr.nextStatement();

                Resource subject =
stmt.getSubject();
                Property property =
stmt.getPredicate();
                RDFNode  object =
stmt.getObject();

                String   subjectName =
subject.getLocalName();
                String   propertyName =
property.getLocalName();
                String   objectName =
object.toString().substring(0, passWord.length());

  if (propertyName.equalsIgnoreCase("hasPassword"))
  if (subjectName.equalsIgnoreCase(name))
 if (objectName.contentEquals(passWord)) {
                    exist = true;
                 }
            }  // while
        } // if
    }  // while

 return exist;

    } //userExist
 //===============================================
 publicboolean foundConcept (String queryStr) {
   boolean found = false;
   OntClass concept = model.getOntClass(ontNS +
"SQAConcept");

Iterator ind = concept.listInstances();

if (ind != null)
while (ind.hasNext() && !found){
          String uri = ind.next().toString();
          String name =
uri.substring(ontNS.length());

if (name.equalsIgnoreCase(queryStr))
 found = true;
        }
   return found;
 }
 //=============================================
 publicvoid consumedLO (String name, String lo)
 {
   Resource s = null;
   RDFNode  o = null;
   Property p = null;

   String buffer = lo;
   buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"
","_");
```

```
  s = model.getResource(ontNS+name);
  p =
model.getObjectProperty(ontNS+"ConsumedLearningObject"
);

  s.addProperty(p, ontNS+buffer);

  try {

   FileWriter fr = new FileWriter (sourceFile);
   model.write(fr, "RDF/XML-ABBREV");
  }
  catch (Exception e) {
   e.printStackTrace();
  }
 }
 ///==========================================
private RDFNode getObject(String lo) {

      OntClass developer = model.getOntClass(ontNS +
"LearningObject");

Iterator ind = developer.listInstances();
while (ind.hasNext()){
 RDFNode currentLO = (RDFNode)ind.next();
 if (currentLO.toString().equalsIgnoreCase(ontNS+lo))
  return currentLO;
      }
      System.out.println("Object not found");
returnnull;
 }
 //==========================================
 private Resource getSubject(String name) {

  OntClass developer = model.getOntClass(ontNS +
"Developer");

Iterator ind = developer.listInstances();
while (ind.hasNext()){
 Resource devloper = (Resource)ind.next();
 if
(developer.toString().equalsIgnoreCase(ontNS+name))
  return developer;
      }
      System.out.println("Subject not found");
returnnull;
 }
 //==========================================
 publicvoid coreLearningObjects (String queryStr,
String userName){

 // createXML.writeXML (query);
        getCoreLOs().clear();
        getURLs().clear();
        getConsumedLOs().clear();
        getConsumedURLs().clear();

        StmtIterator itr = model.listStatements();
 while (itr.hasNext()){
  Statement stmt = itr.nextStatement();
  Resource  subject = stmt.getSubject();
  Property  property = stmt.getPredicate();
  RDFNode   object = stmt.getObject();

  if (property.canAs(ObjectProperty.class))
  {
   String subjectName = subject.getLocalName();
   String propertyName = property.getLocalName();
   String objectName = object.toString();

   if
(propertyName.equalsIgnoreCase("CoreLearningObject"))
    if (subjectName.equalsIgnoreCase(queryStr))
    {
     String lo =
objectName.substring(source.length());

     if (! isConsumed (objectName, userName))
     {
      String buffer = lo;
```

```
        buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
          getCoreLOs().add(buffer);

          StmtIterator itr2 = model.listStatements();
          while (itr2.hasNext())
          {
            Statement st = itr2.nextStatement();
            Resource  s = st.getSubject();
            Property  p = st.getPredicate();
            RDFNode   o = st.getObject();

            if (p.canAs(ObjectProperty.class))
            {
              String sName = s.getLocalName();
              String pName = p.getLocalName();
              String oName = o.toString();

              if (pName.equalsIgnoreCase("HasURL"))
                if (sName.equalsIgnoreCase(lo))
                {
                  String url =
oName;//.substring(source.length());
                  getURLs().add(url);
                }
            }
          }  // while
        } // if not consumed
        elseif (isConsumed (objectName, userName))   //
if already consumed
        {
          String buffer = lo;
          buffer =
org.apache.commons.lang.StringUtils.replace(buffer,"_"
," ");
          getConsumedLOs().add(buffer);

          StmtIterator itr2 = model.listStatements();
          while (itr2.hasNext())
          {
            Statement st = itr2.nextStatement();
            Resource  s = st.getSubject();
            Property  p = st.getPredicate();
            RDFNode   o = st.getObject();

            if (p.canAs(ObjectProperty.class))
            {
              String sName = s.getLocalName();
              String pName = p.getLocalName();
              String oName = o.toString();

              if (pName.equalsIgnoreCase("HasURL"))
                if (sName.equalsIgnoreCase(lo))
                {
                  String url =
oName;//.substring(source.length());
                  getConsumedURLs().add(url);
                }
            }
          }  // while
        } // if consumed
      }
      }  // if
    } // if

    relatedConcepts.clear();
          qualityAttributes (queryStr);
          isQualityAttributeOf (queryStr);
          InvokedProcess (queryStr);
          usedResources (queryStr);
          usedBy (queryStr);
          Inputs (queryStr);
          IsInputTo (queryStr);
          Deliverables (queryStr);
          ProducedBy (queryStr);
          measure (queryStr);
          isMeasuredBy (queryStr);
          associatedWith (queryStr);
          isInputToMeasurement(queryStr);
          hasMeasurementInput(queryStr);
```

```
        isMetricsOf(queryStr);
        hasMetrics(queryStr);

    } // coreLearningObjects

 //==========================================

 publicboolean isConsumed (String LO, String userName)
 {
  boolean consumed = false;
  boolean developerFound = false;
  OntClass developer = model.getOntClass(ontNS +
"Developer");

Iterator ind = developer.listInstances();

if (ind != null)
while (ind.hasNext() && !developerFound){

          String uri = ind.next().toString();
          String name =
uri.substring(ontNS.length());

if (name.equalsIgnoreCase(userName)) {
                developerFound = true;
                StmtIterator itr =
model.listStatements();
while (itr.hasNext() && !consumed) {
                  Statement stmt =
itr.nextStatement();

                  Resource subject =
stmt.getSubject();
                  Property property =
stmt.getPredicate();
                  RDFNode  object =
stmt.getObject();

                  String   subjectName =
subject.getLocalName();
                                  String   propertyName =
property.getLocalName();
                                  String   objectName =
object.toString();

if
(propertyName.equalsIgnoreCase("ConsumedLearningObject
"))
if (subjectName.equalsIgnoreCase(name))
if (objectName.equalsIgnoreCase(LO) ||
object.equals(ontNS+LO)) {
 returntrue;
                             }
                      } // while
                  }  // if
              } // while
  returnfalse;
 }

//==============================================
 publicvoid associatedWith (String queryStr) {

  associatedProcess.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
     {
          Statement stmt = itr.nextStatement();
          Resource subject = stmt.getSubject();
          Property property = stmt.getPredicate();
          RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
          {
 //check for the ConductedUsing relation
 if
(property.getLocalName().equalsIgnoreCase("ConductedUs
ing") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
              {
```

```
 getAssociatedProcess().add(object.toString().substrin
g(source.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

                    }
                }
            } // for
    }
//=================================================
 publicvoid isMetricsOf(String queryStr) {

   measurements.clear();
  for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
            Property property = stmt.getPredicate();
            RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
                {
 //check for the isMeasurementMetricOf relation
 if
(property.getLocalName().equalsIgnoreCase("isMeasureme
ntMetricOf") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                    {

 getMeasurements().add(object.toString().substring(sou
rce.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

                    }
                }
            } // for
    }
//=================================================
 publicvoid hasMetrics(String queryStr) {

   metrics.clear();
  for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
            Property property = stmt.getPredicate();
            RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
                {
 //check for the hasMeasurementMetric relation
 if
(property.getLocalName().equalsIgnoreCase("hasMeasurem
entMetric") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                    {

 getMetrics().add(object.toString().substring(source.l
ength()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

                    }
                }
            } // for
    }
//=================================================
 publicvoid hasMeasurementInput(String queryStr) {

   measurementInputs.clear();
  for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
```

```
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
        {
 //check for the hasMeasurementMetricInput relation
 if
(property.getLocalName().equalsIgnoreCase("hasMeasurem
entMetricInput") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
            {

 getMeasurementInputs().add(object.toString().substrin
g(source.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

            }
        }
    }  // for
 }
//============================================
 publicvoid isInputToMeasurement(String queryStr) {

  inputToMeasurements.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
        {
 //check for the isInputToMeasurementMetric relation
 if
(property.getLocalName().equalsIgnoreCase("isInputToMe
asurementMetric") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
            {

 getInputToMeasurements().add(object.toString().substr
ing(source.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

            }
        }
    }  // for
}
//============================================
 publicvoid measure(String queryStr) {

  measures.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
        {
 //check for the Measure relation
 if
(property.getLocalName().equalsIgnoreCase("Measure")
&& subject.getLocalName().equalsIgnoreCase(queryStr))
            {

 getMeasures().add(object.toString().substring(source.
length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

            }
        }
    }  // for
```

```
 }
//==============================================
 publicvoid isMeasuredBy(String queryStr) {

  measuredBy.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
     {
         Statement stmt = itr.nextStatement();
         Resource subject = stmt.getSubject();
         Property property = stmt.getPredicate();
         RDFNode object = stmt.getObject();

  if (property.canAs(ObjectProperty.class))
         {
  //check for the MeasuredBy relation
  if
(property.getLocalName().equalsIgnoreCase("MeasuredBy"
) &&
subject.getLocalName().equalsIgnoreCase(queryStr))
             {

  getMeasuredBy().add(object.toString().substring(sourc
e.length()));

  getRelatedConcepts().add(object.toString().substring(
source.length()));

             }
         }
     }  // for
 }
//==============================================
 publicvoid usedResources(String queryStr) {

  usedResource.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
     {
         Statement stmt = itr.nextStatement();
         Resource subject = stmt.getSubject();
         Property property = stmt.getPredicate();
         RDFNode object = stmt.getObject();

  if (property.canAs(ObjectProperty.class))
         {
  //check for the Use relation
  if (property.getLocalName().equalsIgnoreCase("Use")
&& subject.getLocalName().equalsIgnoreCase(queryStr))
             {

  getUsedResources().add(object.toString().substring(so
urce.length()));

  getRelatedConcepts().add(object.toString().substring(
source.length()));

             }
         }
     }  // for
 }
//==============================================
 publicvoid usedBy(String queryStr) {

  usedByProcess.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
     {
         Statement stmt = itr.nextStatement();
         Resource subject = stmt.getSubject();
         Property property = stmt.getPredicate();
         RDFNode object = stmt.getObject();

  if (property.canAs(ObjectProperty.class))
         {
  //check for the UsedBy relation
  if
(property.getLocalName().equalsIgnoreCase("UsedBy") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
             {
```

```
 getUsedByProcess().add(object.toString().substring(so
urce.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

                }
            }
    }  // for
 }
//==============================================

 publicvoid Deliverables (String queryStr) {

  deliverables.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
            {
 //check for the Produce relation
 if
(property.getLocalName().equalsIgnoreCase("Produce")
&& subject.getLocalName().equalsIgnoreCase(queryStr))
                {

getDeliverables().add(object.toString().substring(sour
ce.length()));

getRelatedConcepts().add(object.toString().substring(s
ource.length()));

                }
            }
    }  // for
 }
//==============================================
privatevoid ProducedBy(String queryStr) {

  producedBy.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
            {
 //check for the producedBy relation
 if
(property.getLocalName().equalsIgnoreCase("producedBy"
) &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                {

 getProducedBy().add(object.toString().substring(sourc
e.length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

                }
            }
    }  // for
 }
 //==============================================
 publicvoid Inputs (String queryStr) {

  inputs.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
```

```
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
        {
 //check for the hasInput relation
 if
(property.getLocalName().equalsIgnoreCase("HasInput")
&& subject.getLocalName().equalsIgnoreCase(queryStr))
            {
getInputs().add(object.toString().substring(source.len
gth()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));

            }
        }
    }  // for
 }
 //==============================================
 publicvoid IsInputTo (String queryStr) {

  isInputTo.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
    {
        Statement stmt = itr.nextStatement();
        Resource subject = stmt.getSubject();
        Property property = stmt.getPredicate();
        RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
        {
 //check for the isInputTo relation
 if
(property.getLocalName().equalsIgnoreCase("IsInputTo")
&& subject.getLocalName().equalsIgnoreCase(queryStr))
            {
getIsInputTo().add(object.toString().substring(source.
length()));

 getRelatedConcepts().add(object.toString().substring(
source.length()));
            }
        }
    } // for
}
 //==============================================
 public String Isa (String queryStr)
    {
        Statement[] statemets = new Statement [50];

 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement s = (Statement)
itr.nextStatement();

            Resource subject = s.getSubject();
            Property property = s.getPredicate();
            RDFNode  object = s.getObject();

 if (property.canAs(ObjectProperty.class))
            {
 if (property.getLocalName().equalsIgnoreCase(""));
 //do something
            }
        }
 returnnull;
    }
 //==============================================
 publicvoid qualityAttributes(String queryStr) {

 relatedQAs.clear();
 for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
```

```
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
            Property property = stmt.getPredicate();
            RDFNode object = stmt.getObject();

    if (property.canAs(ObjectProperty.class))
            {
    //check for the hasQualityAttribute relation
    if
(property.getLocalName().equalsIgnoreCase("HasQualityA
ttribute") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                {

    getRelatedQAs().add(object.toString().substring(sourc
e.length()));

    getRelatedConcepts().add(object.toString().substring(
source.length()));

                }
            }
        }   // for


    }
//=========================================
    publicvoid isQualityAttributeOf(String queryStr) {

    isQAof.clear();
    for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
            Property property = stmt.getPredicate();
            RDFNode object = stmt.getObject();

    if (property.canAs(ObjectProperty.class))
            {
    //check for the isQua;ityAttributeOf relation
    if
(property.getLocalName().equalsIgnoreCase("IsQualityAt
tributeOf") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                {
    getIsQAof().add(object.toString().substring(source.le
ngth()));

    getRelatedConcepts().add(object.toString().substring(
source.length()));

                }
            }
        }   // for

    }
//=========================================
    publicvoid InvokedProcess (String queryStr){

    invokedProcess.clear();
    for (StmtIterator itr = model.listStatements();
itr.hasNext();)
        {
            Statement stmt = itr.nextStatement();
            Resource subject = stmt.getSubject();
            Property property = stmt.getPredicate();
            RDFNode object = stmt.getObject();

    if (property.canAs(ObjectProperty.class))
            {
    //check for the invoke relation
    if
(property.getLocalName().equalsIgnoreCase("Invoke") &&
subject.getLocalName().equalsIgnoreCase(queryStr))
                {

getInvokedProcess().add(object.toString().substring(so
urce.length()));
```

```
getRelatedConcepts().add(object.toString().substring(s
ource.length()));
                    }
                }
            } // for
        }

    //=====================================================
    publicvoid relatedLearningObjects (String query) {

    //loop through all model's statements and add only
those related to query
        Statement[] statements = new Statement [50];
    int index = 0;

    for (StmtIterator itr = model.listStatements();
itr.hasNext(); )
            {
                String upperClass;
                Statement stmt = itr.nextStatement();

                Resource subject = stmt.getSubject();
                Property property = stmt.getPredicate();
                RDFNode  object = stmt.getObject();

    // check for object peoperties only
    if (property.canAs(ObjectProperty.class))
                {
    //check for the Isa relation
    if
(property.getLocalName().equalsIgnoreCase("hasQualityA
ttribute") &&
subject.getLocalName().equalsIgnoreCase(query))
                    {
    // add to statements
                        String lo = object.toString();
    relatedLOs.add(lo);
                        statements [index] = stmt;
                        index++;
                    }

    elseif
(property.getLocalName().equalsIgnoreCase("isQualityAt
tributeOf") &&
subject.getLocalName().equalsIgnoreCase(query))
                    {
    // add to statements
                        String lo = object.toString();
    relatedLOs.add(lo);
                        statements [index] = stmt;
                        index++;
                    }

    elseif
(property.getLocalName().equalsIgnoreCase("Produce")
&& subject.getLocalName().equalsIgnoreCase(query))
                    {
    // add to statements
                        String lo = object.toString();
    relatedLOs.add(lo);
                        statements [index] = stmt;
                        index++;
                    }

    elseif
(property.getLocalName().equalsIgnoreCase("producedBy"
) && subject.getLocalName().equalsIgnoreCase(query))
                    {
    // add to statements
                        String lo = object.toString();
    relatedLOs.add(lo);
                        statements [index] = stmt;
                        index++;
                    }

    elseif
(property.getLocalName().equalsIgnoreCase("MeasuredBy"
) && subject.getLocalName().equalsIgnoreCase(query))
                    {
    // add to statements
```

```
                    String lo = object.toString();
 relatedLOs.add(lo);
                    statements [index] = stmt;
                    index++;
               }
 elseif
(property.getLocalName().equalsIgnoreCase("Measure")
&& subject.getLocalName().equalsIgnoreCase(query))
               {
 // add to statements
                    String lo = object.toString();
 relatedLOs.add(lo);
                    statements [index] = stmt;
                    index++;
               }

          }
     } // loop through model statements
 // return statements;
   }
 //==========================
 publicvoid ClearAllLOs (String name)
   {
  for (StmtIterator itr = model.listStatements();
itr.hasNext();)
      {
          Statement stmt = itr.nextStatement();
          Resource subject = stmt.getSubject();
          Property property = stmt.getPredicate();
          RDFNode object = stmt.getObject();

 if (property.canAs(ObjectProperty.class))
          {
 //check for the ConsumedLearningObject relation

 if
(property.getLocalName().equalsIgnoreCase("ConsumedLea
rningObject") &&
subject.getLocalName().equalsIgnoreCase(name))
                {
 model.remove(stmt);
                }
              }
          } // for
  try {
     FileWriter fr = new FileWriter
("C:/Users/vaio/Desktop/SQOntology.owl");
     model.write(fr, "RDF/XML-ABBREV");
   }
   catch (Exception e) {
     e.printStackTrace();
   }

   }
}
```

LOServlet.java

```
package RelatedLOs;

import java.io.IOException;
import java.io.PrintWriter;
importjava.net.InetSocketAddress;
import java.util.Iterator;
import java.util.Vector;
importjava.util.concurrent.Executor;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

importcom.sun.net.httpserver.HttpContext;
importcom.sun.net.httpserver.HttpHandler;
importcom.sun.net.httpserver.HttpServer;

@WebServlet ("/LOServlet")
```

```
publicclassLOServletextends HttpServlet {

 String userName;
 String queryStr;
 intsessionNum;
 //SQA.System.ReadOntology model;

 public LOServlet (){
  super();
 }

 @Override
 protectedvoid doGet(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException {
  try
  {
   queryStr = (String)
request.getParameter("queryStr");
   userName = (String)
request.getParameter("userName");
   sessionNum =
Integer.parseInt((String)request.getParameter("session
Num"));
  }
  catch (Exception e)
  {
   System.out.println("Error in initialization");
   e.printStackTrace();
  }

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  out.println("<html>");
        out.println("<body  bgColor = 'Gainsboro'>");
        out.println("<div id = 'header' style =
'background-color:orange; margin-bottom:0'>");
        out.println ("<Center><h3 color:Black>SQAES:
Your SQA Learning System</h3></Center>");
// new search
        out.println("<p align ='right' font-size =
'10px'><a href='javascript:document.form['search
form'].submit();'>New Search</a></p>");
//out.println ("<Center><h3>Personalized SQA Learning
System</h3></Center>\n");
        out.println("<p align =\"right\"><a
href=\"Exit.html\">Sign Out</a></p>");
        out.println("</div>");
        SQA.Learning.System.ReadingOntology model =
new SQA.Learning.System.ReadingOntology();

        UpdateXML userProfile = new UpdateXML
(userName);
        userProfile.addConcept(sessionNum, queryStr);
        userProfile.writeToFile(userName);

        model.coreLearningObjects(queryStr, userName);
Vector coreLOs = model.getCoreLOs();
Vector URLs = model.getURLs();

        out.println ("<left><strong><i><u>" + "Core
Knowledge about " +queryStr+
"</u></i></strong><br><br>");

if (! coreLOs.isEmpty())
        {
 Iterator itr1 = coreLOs.iterator();
Iterator itr2 = URLs.iterator();

while (itr1.hasNext())
          {
 String lo = itr1.next().toString();
 String location = (String)itr2.next();
 out.println("<a target = '_blank'
href=\"http://localhost:8080/SQAES/consumedLO?name="+u
serName+"&lo="+lo+"&location="+location+"&sessionNum="
+Integer.toString(sessionNum)+"&Query="+queryStr+"\">"
+lo+"</a><br>");
          }
        }
```

```
        out.println("<br><br><br>");

        out.println("</body>");
        out.println("</html>");
  }
  //====================================

  @Override
  protectedvoid doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {

  }

}
```

ConsumedLO.java

```
package RelatedLOs;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import SQA.Learning.System.ReadingOntology;

importcom.hp.hpl.jena.ontology.OntModel;
importcom.hp.hpl.jena.rdf.model.Property;
importcom.hp.hpl.jena.rdf.model.Resource;
importcom.hp.hpl.jena.rdf.model.Statement;
importcom.hp.hpl.jena.rdf.model.StmtIterator;

/**
 * Servlet implementation class consumedLO
 */
@WebServlet("/consumedLO")
publicclassconsumedLOextends HttpServlet {
    //private static final long serialVersionUID = 1L;

    String name;
    String lo;
    ReadingOntology model = new ReadingOntology();
    String location;
intsessionNum;
    String concept;

public consumedLO() {
super();

    }

@Override
  protectedvoid doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
    try
    {
     name = (String) request.getParameter("name");
     lo = (String) request.getParameter("lo");
     location = (String)
request.getParameter("location");
     sessionNum = Integer.parseInt((String)
request.getParameter("sessionNum"));
     concept = (String) request.getParameter("Query");
    }
    catch (Exception e)
    {
     System.out.println("Error in initialization");
     e.printStackTrace();
    }

    if ( ! model.isConsumed(lo, name))
     model.consumedLO(name, lo);
    UpdateXML userProfile = new UpdateXML(name);
    userProfile.addLO(sessionNum, concept, lo);
    userProfile.writeToFile(name);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

```
    out.println("<html>");
    out.println("<head>");
    out.println("<title>"+lo+"</title>");
    //out.println ("<meta HTTP-EQUIV=\"REFRESH\"
content=2; url=\""+location+"\">"+"</meta>");
    out.println ("<meta HTTP-EQUIV=\"REFRESH\"
content=\"2; url="+location+"\">"+"</meta>");
    out.println("</head>");
    out.println("<body  bgColor = 'Gainsboro'>");
        out.println("Please wait while we direct you
to "+lo);
        out.print("<br> If you are not redirected in 5
seconds please ");
    out.println("<a href=\""+location+"\">"+" click
here</a>");
    out.println("<go href ='http://www.web-
source.net/html_redirect.htm'>"+"</go>");

    out.println("</body>");
    out.println("</html>");

}

 @Override
 protectedvoid doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
   // TODO Auto-generated method stub
  }

}
```

UpdatedXML.java

```
package RelatedLOs;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Iterator;
import java.util.List;

import org.jdom.Attribute;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;

publicclass UpdateXML {

  public Document doc = new Document();
  public Element sessions;
  publicstaticfinal String DATE_FORMAT_NOW = "yyyy-MM-
dd HH:mm:ss";

  public UpdateXML (String fileName)
  {
   try {
     SAXBuilder builder = new SAXBuilder();
    try {
     doc = builder.build(new
File("c:/UserProfiles/"+fileName+".xml"));
     sessions = (Element) doc.getRootElement();
   } catch (IOException e) {
     e.printStackTrace();
   }
   } catch(JDOMException e) {
     e.printStackTrace();
   } catch(NullPointerException e) {
   e.printStackTrace();
   }
  }

  publicint addSession ()
  {
   Attribute nos =
 sessions.getAttribute("numOfSessions");
   String numSt = nos.getValue();
```

```
  int num = Integer.parseInt(numSt);
  numSt = Integer.toString(num+1);
  sessions.setAttribute("numOfSessions", numSt);
  Element newSession = new Element ("Session");
  newSession.addContent(new Element
("id").addContent(numSt));
  // get date and time
  Calendar cal = Calendar.getInstance();
     SimpleDateFormat sdf = new
SimpleDateFormat(DATE_FORMAT_NOW);

  newSession.addContent (new Element
("time").addContent(sdf.format(cal.getTime())));
  sessions.addContent(newSession);
  return Integer.parseInt(numSt);

 }

 publicvoid addConcept (int sessionId, String concept)
 {
  Boolean notFound = true;
  List sessionList = sessions.getChildren("Session");
  Iterator itr = sessionList.iterator();
  while (itr.hasNext() && notFound)
  {
   Element e = (Element) itr.next();
   String id = (e.getChild("id")).getValue();
   if (sessionId == Integer.parseInt(id))
   {
       notFound = false;
       e.addContent(new Element
("Query").setAttribute("Concept", concept));
   }
  }
 }

 publicvoid addLO (int sessionId, String concept,
String LO)
 {
  Boolean notFound = true;
  Boolean foundConcept = false;
  List sessionList = sessions.getChildren("Session");
  Iterator itr = sessionList.iterator();
  while (itr.hasNext() && notFound)
  {
   Element session = (Element) itr.next();
   String id = (session.getChild("id")).getValue();
   if (sessionId == Integer.parseInt(id))
   {
    Iterator conceptItr =
session.getChildren("Query").iterator();
    while (conceptItr.hasNext() && !foundConcept)
      {
     Element c = (Element) conceptItr.next();
     String conceptName =
c.getAttributeValue("Concept");// .getValue();
     if (conceptName.equalsIgnoreCase(concept))
     {
      foundConcept = true;
      c.addContent (new Element
("Consumed_LO").addContent(LO));
     }
        }
   }

  }
 }

 publicvoid writeToFile (String fileName)
 {
  try {
    FileWriter writer = new
FileWriter("c:/UserProfiles/"+fileName+".xml");
    XMLOutputter outputter = new XMLOutputter();
    Format format = Format.getRawFormat();
    format.setIndent("  ");
    format.setLineSeparator("\n");
    //outputter.setFormat(format);
    outputter.output(doc,writer);
    writer.close();
```

```
  } catch (IOException e) {
   e.printStackTrace();
  }

}
publicvoid display ()
{
  try {
   Format format = Format.getRawFormat();
   format.setIndent("  ");
   format.setLineSeparator("\n");
      XMLOutputter outputter = new XMLOutputter();
      outputter.setFormat(format);

      outputter.output(doc, System.out);
  } catch (java.io.IOException e) {
      e.printStackTrace();
  }
```