RESEARCH ARTICLE OPEN ACCESS

An Evolutionary Approach to Software Testing Ontology Development

Nor Adnan Yahaya [1,2], Mansoor Abdullateef Abdulgabber [3], Shadia Yahya Baroud [1]

[1] University Malaysia of Computer Science and Engineering, Selangor, MALAYSIA
[2] Malaysia University of Science and Technology, Selangor, MALAYSIA
[3] Unilangit Academy, Selangor, MALAYSIA

ABSTRACT

Software Testing Ontology (STO) serves as a formal representation of the vocabulary to be used in descriptions pertaining to software testing knowledge area. The need for a standardized STO is crucial in making descriptions of various aspects of software testing body of knowledge and practices not only machine processable, but also can be done in a uniform way. This paper describes the steps we have used to build the base for the standard STO that, in turn, can be extended and further enhanced towards standardization. The evolving STO is envisioned to be an important addition towards enriching the functionality of semantic software testing tools and systems.

Keywords: - **Ontology**, Software Testing, Semantic Web, Software Testing Ontology.

I. INTRODUCTION

The term ontology was first introduced in the field of philosophy where it means a systematic explanation of individual. Several fields of study have now used the term with interpretations that suite their respective interests. In philosophy, ontology is used to understand and distinguish the meaning of things, the changes of their status, and to classify the entities of the world. In scientific fields, ontology is derived from cognitive semantic or the science of being and used to describe semantic constructs based on the meaning of words (as dictionary in linguistic) [1]. According to [2], ontologies are now widely used in various applications such as knowledge management, intelligent integration information, information retrieval, bioinformatics, education, and relatively new fields such as the Semantic Web.

Software testing is a growing discipline that sees the need for a standardized ontology. A software testing ontology (STO) then is a formal and an explicit description of concepts and relationships used to describe various aspects of software testing artefacts. It allows for the formalization of the standardized software testing terms [3] so that reasoning built on them can be automated. The STO can also be useful in documenting and analysing software testing activities and artefacts. It must be built to make some aspects of all these machine processible and more amenable to realistic interpretation by third party (i.e. human, machine, software agent, etc). The STO can also be considered as a semantic repository which manages the storage and query, offers easier integration and dynamic interpretation of software testing test cases and data. The semantic repository approach allows easier changes and automated interpretation of the data compared to the approach used in relational DBMS [4].

Testing ontology builds upon testing terminology which comprises all terms that belong to the testing process in the software engineering body of knowledge or domain. Defining standard terms in any domain will benefit all parties working within that specific field. Machines, through understanding of how to manipulate the process, will also gain the same benefits gained by humans from standardizing the terms in that field. Without a standard testing terminology, different personnel involved in the testing process might use different terms for the same item. For example, one test manager calls an item a "fault", while the tester calls it "error," and the programmer calls it a "bug." All three personnel are actually referring to the same item. However, using a different terminology, each may think the other is referring to a separate item. In another scenario, if this data is input into a machine for an automated testing process, the machine cannot detect that these three terms are actually synonyms referring to the same item, not three separate things. This causes delay in the job or work due to the confusion caused. In a case study that was held in an industrial setting, it was found that the job was delayed due to the terminology confusion [5]. It is foreseen that a shared understanding among different terminologies would definitely overcome overlapping and mismatching of concept interpretations. This will subsequently benefit knowledge integration and raise the potential of reusing resources. Ultimately, it also helps to reduce data inconsistency.

This paper presents an approach to STO development using semantic technological framework. This initiative is founded by the following premises:

1. Software testing ontologies are crucial to test automation and their development can be driven from the perspective of test management that has to deal with extensive testing-related knowledge.
2. The ontologies must be developed using a versatile knowledge management (KM) framework and tool as well as leveraging on semantic technologies.

3. Standardization of STO can be achieved through evolutionary process by making continuous extension, alignment, and merging of ontologies.

This paper is structured as follows. Section 2 presents the recent trend in software testing. Section 3 describes the main steps involved in building the core STO. Section 4 illustrates the implementation of the ontology in Protégé. This is followed by the evaluation of the STO in Section 5. Finally, Section 6 concludes and summarizes the contribution of this work.

II. RECENT TREND IN SOFTWARE TESTING

Software testing has become a mature discipline, both in theory and practice. With the advent of the new technologies of IR4.0, the trend towards providing more and more automated support to the software testing process is expected to prevail. Central to this is the need to have good automated support for test management activities that relies on efficient and effective management of testing-related knowledge. A study on the most promising trend in software testing was made to help steer the proposed evolutionary approach to software testing ontology development. The amalgamation of knowledge-based testing and the semantic web technologies is found to be most promising in developing the technological framework for the evolving STO.

A. Knowledge Management in Software Testing

The systematic literature review [6] captures the following important views and observations from various researchers that point to the future use of knowledge management in software testing:

1. Knowledge management ideas and methodologies are known to have been used in several software development process phases [7],[8],[9].
2. Since software testing itself is a knowledge intensive process and also an important component of software engineering, it is critical to have automated tools for catching, distributing, evaluating, retrieving, and displaying testing knowledge [10].
3. The software testing community has acknowledged the importance of knowledge management and having to learn from the knowledge management community. Accordingly, various efforts have been made to incorporate knowledge management where knowledge-based software testing is used to generate tests utilizing existing system knowledge as an example [11].
4. The tester's own expertise, as well as knowledge of software testing and application domains can be utilized to produce tests and identify faults [12].

All the above suggest that testing information should be gathered and characterized in a cost-effective and managed manner by employing knowledge management principles.

B. Semantic Web Technologies in Software Testing

Furthermore, the systematic literature review [6] also stipulates that with the advent of semantic web technologies, new approaches to merging software and knowledge engineering have also emerged. The review also claims that a number of researches in software testing that have used knowledge management in activities such as automated test generation have taken advantage of semantic web technologies. The following cited works by the review provides some good insights on the deployment of semantic web technologies in software testing.

1. Increasing test automation can be achieved by providing a fairly formal specification of test process data. This is considered as one of the main challenges in knowledge-based software testing approaches. However, because of their logic-based character, inference capabilities, and machine comprehensibility, semantic web data models and ontologies are strong candidates for supplying this formalism and increasing test automation [13].
2. An ontology can be used to model requirements from a software requirements specification (SRS), where the inference rules can specify test case derivation strategies from that ontology. The resultant meta-model of a requirement comprises requirement conditions and parameters as well as test results and actions to be represented in the ontology [14].
3. Other software testing operations, such as test data generation, test reuse, test oracle can also be supported by semantic web technologies. For example, test data can be generated using the Web of Data, a worldwide dataset holding billions of interrelated and machine processable statements encoded in RDF triples [15].

C. Research in Software Testing Ontologies

A systematic literature review (SLR) on software testing ontologies [16] identified 12 different ontologies that were developed during the period from 2000 to 2011 which were then analyzed to answer their research questions. The SLR paper reveals the following important observations:

1. Software testing domain is highly complex which makes ontology development for it not a straight-forward task.
2. Most of the ontologies that have been developed have very limited coverage.
3. The approaches and techniques being used vary, which have led to heterogeneity and the need for a common reference foundational ontology.

Subsequently, the same group of researchers developed a Reference Ontology on Software Testing (ROoST) which is described in [17].

More recently, another systematic review on the past and current works on software testing ontologies [18] concludes that ROoST is the most formally rigorous testing ontology, well modularized and balanced with respect to taxonomic and non-taxonomic relationships. However, the researchers of the study opine that ROoST does have some limitations too. Furthermore, the same study also identified the problem of limited coverage where most of the analyzed ontologies have a lack of terminological coverage on non-functional requirements (NFRs) and static testing.

However, with the advent of the Semantic Web initiative, while ontology has been closely associated with semantic technologies, none of the above reviews project out any work on software testing ontology development that leverage on semantic web technologies.

The above scenarios seem to suggest that works in software testing ontology are still in the early stage towards maturity. There is still no clear indication of any movement towards having one unified ontology for software testing. As such, more research and efforts in the development of software testing ontology are still needed, especially in term of leveraging semantic web technologies.

Towards that end, an approach to development of a software testing ontology management system that is built on semantic technological framework is presented. The next section provides a general description of the steps involved in building the base for the standard STO that, in turn, can be extended and further enhanced. The evolving STO is envisioned to be an important addition towards enriching the functionality of semantic software testing tools and systems. The evolutionary approach is expected to help to minimize the current heterogeneity, ambiguity, and incompleteness problems in terms, properties, and relationships as stipulated in [18].

III. BUILDING THE SOFTWARE TESTING ONTOLOGY

Building ontologies requires the selection of a comprehensive guide. The steps involved in developing the proposed STO is based on the guide produced by [19] that emphasizes on domain reusability of any developed ontology. Accordingly, the first step being taken here was to start with a very high-level conceptualization.

The method in [19] was selected not only because its popularity, but it also provides a simple explanation on how to develop and evaluate the first ontology through clearly identified steps. Several challenges were met while building the ontology. In particular, the main one was classifying terms to formalize the conceptualization. This task consumed a lot of effort and required critical decisions. In the aforementioned guide, there are seven main steps to build STO which are presented below.

A. Determine the domain and scope of STO

Several selected questions were used to define the domain and scope of STO, characterized by the following goals: purpose, usage, type of information, and who will need the STO. Table 1 illustrates the questions & answers used.

Table 1 Determine STO's domain & scope

Question	Answer
What is the domain the Ontology will cover?	The purpose of building this ontology is to cover the software testing area as the Domain & it is referred to as STO.
What is STO going to be used for?	The STO is built to be used as part of an infrastructure for Semantic Technology regardless of which application uses it with the intention of focusing on representing test cases for management and reusability.
What type of answers should STO provide?	STO needs to provide an understandable, conceptualized and linked vocabulary required by the Software Testing Domain.
Who will use STO?	The STO end users are identified as third party whether they are machines such as (Semantic Agents, Semantic Desktop, etc) or humans such as (Software Testers, Test Managers Test Case Creators, etc)

B. Consider reusing existing Software Testing Ontologies

There are Software Testing Ontologies which have already been built and published in the literature. Studying some of the existing ontologies was an important process for this step. Table 2 shows the analysed findings of the study.

Table 2 Analysed Findings for Existing STO

Ontology Name	Description	Reference
Ontology of Software Testing OntoTest	Defines software testing concepts in a layered approach. The main layer covers main testing concepts and relations. The sub layers cover Testing Processes, Testing Phases, Testing Artefact, Testing Steps, Testing Procedures and Testing Resources.	[20]
Software Testing Ontology for WS (STOWS)	Defines concepts related to software testing into two groups: the basic concepts include context, activity, method, artefact, and environment; and compound concepts include tester, capability and test task.	[21]
Test Ontology Model (TOM)	Defined to specify the test concepts, relationships and semantics from two aspects: (1)Test Design such as test data, test behaviour and test sases; and (2) Test Execution such as test plan, schedule and configuration.	[22]

Table 2 above implies that there are still some limitations on the domain terms (especially those related to test case as individual) as well as relations between concepts and specific tasks. Therefore, instead of reusing the whole ontology, only some of the concepts' names were adopted while the remaining concepts were introduced from scratch to overcome the aforementioned limitations.

C. Enumerate important terms in the ontology

International Software Testing Qualifications Board is a not-for-profit association founded in Edinburgh in November 2002. One of their missions is to promote a common language for testers globally. They form groups in different areas of software testing and one of the groups is the ISTQB Glossary working group which aims to deliver a standard glossary of testing and related terms. There are various versions of the glossary as the group keeps updating the new terms when necessary. STO was built based on ISQB-glossary Version 3.01 [3] as it presents the most current concepts, terms and definitions of software testing domain and the related artefacts. All terms and concepts presented in the glossary are covered and the taxonomy produced was based on our understanding of the domain. The next step shows how these concepts are classified.

D. Define the concepts and concept hierarchy in the STO

Defining the concepts and their hierarchy concern several approaches identified in the literature as mentioned in the guidelines of the proposed method. The top-down approach is used based on the assumption that it would be more understandable by end users. This definition engaged us with several steps as described briefly below.

1. Categorize the main concepts according to general classifications. This is shown in Table 3 below.

Table 3 Definition of General Classification

General Classification	Definition
Tester	Terms include particular parties that conduct the test activity.
Testing Task	Terms include everyday jobs for performing the testing process.
Artefact	Terms include related pieces to the test and testing process.
Environment	Terms include the surrounding of the testing process and the trait terms from which the process can be described.

2. Identify the sub and sub-sub concepts of the high-level concepts (the general classifications). This is shown in Table 4 below.

Table 4 Identifying the sub-Concepts

Main Concept	Sub Concept	Sub-Sub Concept
Tester	Human	Individual
		Team
	Software_Tool	
Environment	Features	
	Hardware	
	Software	
Artefact	Text	Code
		Document
		Data
		Measurement
	Images	
	Standard	Criteria
		Guide
		Report
		Plan
		Term
TestingTask	Context	Purpose
		Scope
	Activities	Intrinsic
		Extrinsic
	Method	Technique
		Approach
		Practice

3. Classifying the remaining terms in the glossary into these identified concepts.

E. Define the properties of STO concepts

Usually, concepts alone are not enough to give all necessary information. Hence, defining the properties to show the relations between different concepts in the ontology is a necessary step. As shown in Table 5, examples of relations between different concepts are identified. The examples demonstrate a sample view, as STO was built within 59 different types of properties.

Table 5 Examples of Properties

Concept	Object Property	Inverse Property
Team	hasControl	isControlledBy
Software_Tool	hasAutoProcess	isPerformedBy
Code	hasTest	isTestedBy
Measurement	hasMeasurement	isMeasurementOf
Technique	hasTechnique	isTechniqueOf
Approach	has Approach	isApproachOf

F. Define the data properties of STO concepts

This step is required for identifying the data type for each property. The benefit of using data type is the link which can be created between the classes and XML scheme. The STO was built within 32 data type properties. Table 6 shows a sample of the data type. The domain field shows names of concepts that data represent, while the range shows the types of the data.

Table 6 Examples of Data Properties

Data Property	Domain	Range
hasTestID	Individual	string
hasNumberofLine	Code	Integer
hasCreator	Artefact	string
isInfectedCode	Code	Boolean
hasSource	Test Case Suite	string

G. Create instances and individuals of classes

Once the concepts, properties and their data properties were defined, the last step in preparing the STO is to create the instances and individuals of these concepts. Table 7 displays a sample of individuals. With regard to the STO concepts, 106 individuals were built in.

Table 7 Examples of Individuals

Class	Sub Concept/Individual
Images	Call_Graph
	Cause-effect_Diagram
	Control_Flow_Graph
Features	Accuracy
	Complexity
	Maintainability

IV. IMPLEMENTATION OF SOFTWARE TESTING ONTOLOGY WITH PROTÉGÉ 4.0

Protégé 4.0 is an open-source standalone application, written in Java. It provides a plug-and-play environment for the OWL editor that was used to implement the STO. The implementation was performed by following the Protégé guide after getting the STO taxonomy as discussed in the previous section ready. Protégé, with its plug in OWLViz, provides a graphical view for the ontology which makes it easy to understand the relations.

The following three major steps illustrate the major portion of the STO implementation.

A. Building the Classes hierarchy

The STO uses classes (Protégé Guideline) as its basic building block. Classes are then considered as a concrete representation of concepts. Classes and their hierarchy are used to represent the STO taxonomy concepts. Classes are built based on our understanding of the Software Testing Domain. The following steps were followed:

1. Building parent classes to represent the general classification.
2. Building children classes to represent the sub classes.
3. Building grandchildren classes to represent the sub-sub classes.
4. Building the Classes' Individuals to represent the domain objects.

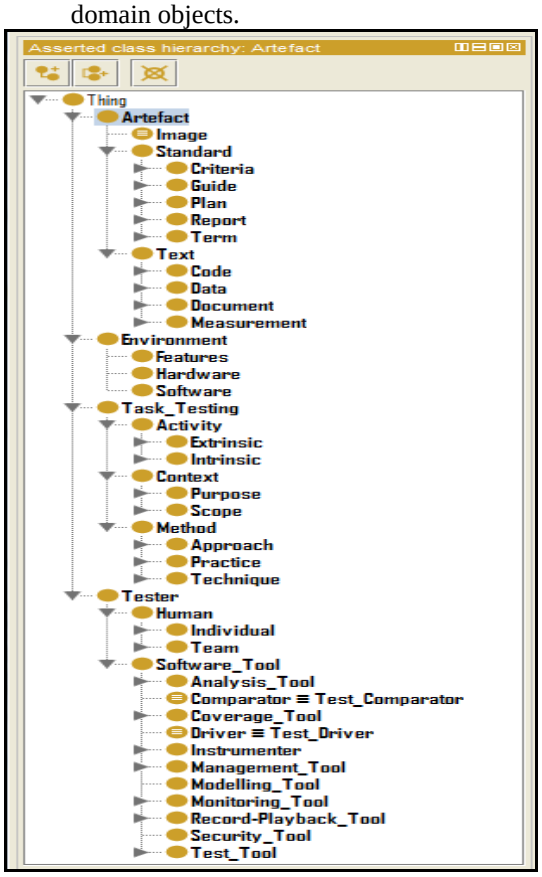


Figure 1 Class hierarchy

Figure 1 shows the hierarchy of the classes. As shown, the STO has four main layers. Each layer is described as follows:

- Tester: This holds the meaning of what/who performs the task of testing. In this layer, Tester is either a person (i.e. human either individual or team) or software (i.e. tools for testing).

- Environment: This holds the meaning of related characteristics to Test. Environment has Features, Hardware and Software as subclasses.

1. Feature Class comprises the behavior terms such as (Pass, Fail and Testability, etc).
2. Hardware Class comprises terms involving hardware such as (Sub, Storage, and Simulator etc).
3. Software Class comprises the software terms such as (Buffer, System and Compiler, etc).

- Artefact: This holds the meaning of objects under the test activities. Text, Image and Standard are created as subclasses of Artefact.

1. Text Class – all included terms describe the Code, Document, Data or Measurement Data.
2. Image Class portrays instances of graphic terms in the domain.
3. Standard Class includes all standards that have been inherited from standard organizations or frameworks. It is classified in Guide, Criteria, Report, Plan or Term classes.

- Task Testing: This defines terms of the main activities in the software testing domain that is in Context, Activity or Method classes.

1. Context Class holds terms describing activities that occur in various software development stages, either for Purpose or Scope.
2. Activity Class includes terms pointing to activities other than testing itself within (Intrinsic) or without (Extrinsic) the system.
3. Method Class takes account of testing activities, whether it is a Technique, Approach or Practice.

Obviously with this simple explanation, the key factor that we depended on in building the general hierarchy classes of the STO is to give effortless meaningful representation for a normal user with basic knowledge in software testing domain. Description Logic specifies hierarchy using restricted set of first-order formulas, and so does OWL reasoning rules. For this purpose, a sub-set of OWL reasoning rules that support our hierarchy classes was derived. For instance, the STO hierarchy class rules are illustrated in Table 8.

Table 8 STO hierarchy class rules

Rule	Description
subClassOf	*(?Individual rdfs:subClassOf ?Human)*
	(?Human rdfs:subClassOf ?Tester) _
	(?Individual rdfs:subClassOf ?Tester)
disjointWith	*(?Individual owl:disjointWith ?Team)*
	(?Inspector rdf:type ?Individual)
	(?Change_Control_Board rdf:type ?Team)
	(?Inspector owl:differentFrom ? Change_Control_Board)

B. Building the Object & Data Properties

Object Properties are binary relations between the classes. After finishing building all classes, the possible relations (Object Property) between these classes were created. Data properties describe relationships between classes and data values. Some STO classes can be represented by data values. A sub-set of OWL reasoning rules that support data properties was derived for this purpose. For instance, hasText & hasTest properties are illustrated in Table 9.

Table 9 STO property rules

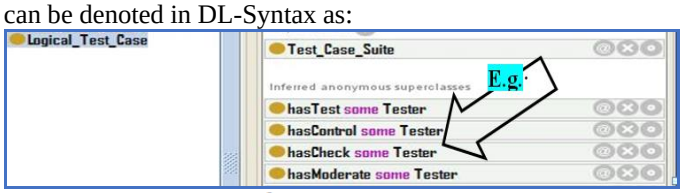
Rule	*Description*
subPropertyOf	*(?hasDocument rdfs:subPropertyOf ?hasData)*
	(?hasData rdfs:subPropertyOf ?hasText) _ (?hasDocument rdfs:subPropertyOf ?hasText)
inverseOf	*(?hasTest owl:inverseOf ?isTestedBy)*
	(?Tester ?hasTest ?Code) _ (?Code ?isTestedBy ?Tester)

C. Building OWL Restrictions Rules

A restriction describes a class of individuals based on the relationships that members of the class participate in. STO restrictions are of two types:

1. Property Restrictions which consist of:

a. someValuesFrom: Existential Restrictions are also known as Some Restrictions, or as some values from restrictions. It can be denoted in DL-Syntax as:



∃ hasTest Tester

b. allValuesFrom: Universal Restrictions are also known as all values from restrictions. It can be denoted in DL-Syntax as:

∀ is CheckedBy Human or Software_Tool

2. Data Restrictions A datatype property can also be used in a restriction to relate individuals to members of a given datatype. For instance, the Code class which has a Boolean data type to check if infected with bugs, has a String data type to carry the name of the code creator and Integer data type to store the number of codes.

V. EVALUATION OF SOFTWARE TEST ONTOLOGY

The STO was evaluated by using reasoning service offered by reasoners plugged in Protégé. The main benefits of the services are computing the classes' hierarchy and logical consistency checking. The STO verification process started at the early stages of the development to ensure the correctness and avoid propagation errors. The two reasoners were used to verify the STO.

1. FaCT++: the first reasoner was used as it is shipped with Protégé. The inferred hierarchy is the automatically computed

class hierarchy by the reasoner. Figure 2 presents the inferred hierarchy graph showing the "no exists" of the inconsistent class. In case of inconsistencies, Protégé would highlight them in red. Meanwhile, the class "Nothing" is to identify the inconsistent classes if any exist.

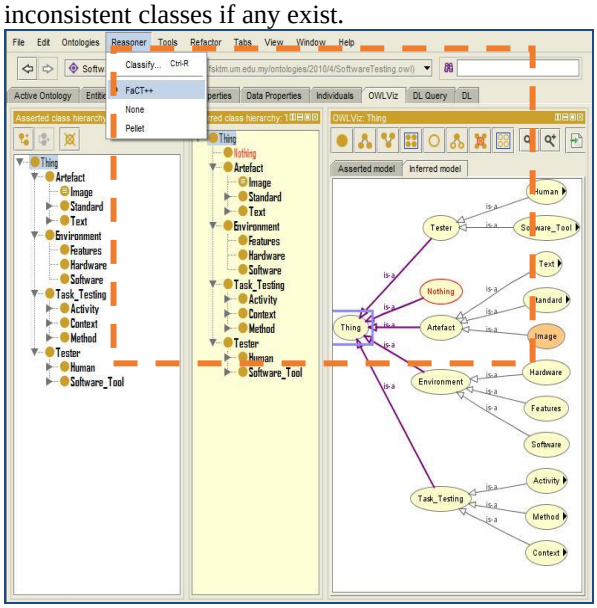


Figure 2 FaCT++ "Nothing" class shows the "no exists" of Inconsistent Class

2. Pellet: the complete OWL-DL reasoner [23]. Protégé allows Pellet plug-in to be installed and be used to compute the OWL. Hence, the computation of the STO via Pellet serves as the second evaluation. Figure 3 presents the inferred hierarchy graph showing the "no exists" of inconsistent class.

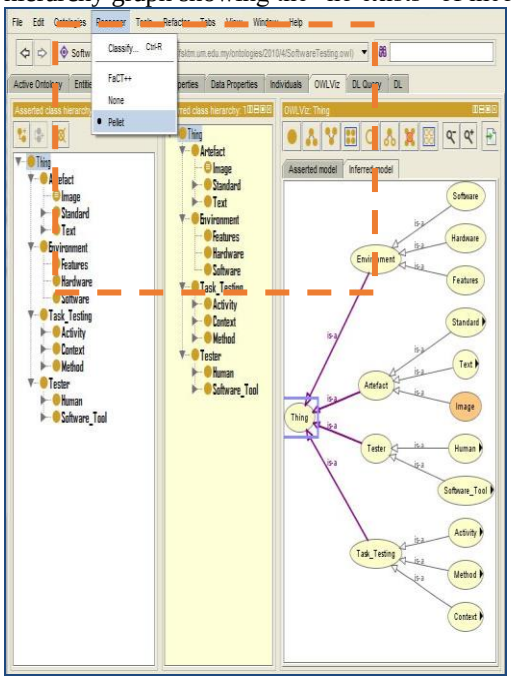


Figure 3 Pellet reasoner shows the "no exists" of Inconsistent Class

VI. CONCLUSIONS

This paper has illustrated the steps involved in creating the base for a standard software testing ontology (STO). The illustration, which is supported by tables and figures taken from [24], has also given some insights on the utility of the approach based on the guide [19] in developing the base ontology which, in turn, can be made to evolve from there. Since semantic technological framework is used from the outset (illustrated through the use of Protégé), the evolving STO can serve as an important addition towards enriching the functionality of semantic software testing tools and systems. The base STO covers the terminology found in standard testing glossary which can be expanded to cover the whole hundreds of concepts related to software testing based on the previously mentioned standard testing glossary. This is large enough to supply accurate reasoning terms for semantic systems such as Semantic Test Case Management System or any other that concerns software testing.

REFERENCES

[1] H. K. Seung and K. L. Sim, "Ontology revision on the semantic Web: Integration of belief revision theory," *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, 2007, doi: 10.1109/HICSS.2007.410.

[2] A. Gómez-Pérez, M. Fernández-López and O. Corcho, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, Springer Verlag, 2004.

[3] I. Software and T. Qualifications, "Standard glossary of terms used in Software Testing International Software Testing Qualifications Board," vol. 3, pp. 1-53, 2014.

[4] J. Davies, R. Studer, and P. Warren, *Semantic Web Technologies: Trends and Research in Ontology-based Systems*. 2006.

[5] T. Parveen, S. Tilley, and G. Gonzalez, "A case study in test management," in *Proceedings of the Annual Southeast Conference*, 2007, vol. 2007, doi: 10.1145/1233341.1233357.

[6] M. Dadkhah, S. Araban, and S. Paydar, "A systematic literature review on semantic web enabled software testing," *J. Syst. Softw.*, vol. 162, p. 110485, 2020, doi: 10.1016/j.jss.2019.110485.

[7] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and Software Technology*, vol. 50, no. 11. 2008, doi: 10.1016/j.infsof.2008.03.006.

[8] I. Rus and M. Lindvall, "Knowledge management in software engineering," *IEEE Software*, vol. 19, no. 3. 2002, doi: 10.1109/MS.2002.1003450.

[9] Vasanthapriyan, J. Tian, and J. Xiang, "A Survey on Knowledge Management in Software Engineering," 2015, doi: 10.1109/QRS-C.2015.48.

[10] J. Andrade et al., "An architectural model for software testing lesson learned systems," in *Information and Software Technology*, 2013, vol. 55, no. 1, doi: 10.1016/j.infsof.2012.03.003.

[11] É. F. De Souza, R. D. A. Falbo, and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," in *Information and Software Technology*, 2015, vol. 57, no. 1, doi: 10.1016/j.infsof.2014.05.016.

[12] J. Itkonen, M. V. Mäntylä, and C. Lassenius, "The role of the tester's knowledge in exploratory software testing," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, 2013, doi: 10.1109/TSE.2012.55.

[13] J. J. Gutiérrez, M. J. Escalona, and M. Mejías, "A Model-Driven approach for functional test case generation," *J. Syst. Softw.*, vol. 109, 2015, doi: 10.1016/j.jss.2015.08.001.

[14] V. Tarasov, H. Tan, M. Ismail, A. Adlemo, and M. Johansson, "Application of inference rules to a software requirements ontology to generate software test cases," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*), 2017, vol. 10161 LNCS, doi: 10.1007/978-3-319-54627-8_7.

[15] L. Mariani, M. Pezzé, O. Riganelli, and M. Santoro, "Link: Exploiting the web of data to generate test inputs," 2014, doi: 10.1145/2610384.2610397.

[16] É. F. Souza, R. A. Falbo, and N. L. Vijaykumar, "Ontologies in software testing: A Systematic Literature Review," *CEUR Workshop Proc.*, vol. 1041, no. October, pp. 71–82, 2013.

[17] É. F. De Souza, R. De Almeida Falbo, and N. L. Vijaykumar, "ROoST: Reference ontology on software testing," *Appl. Ontol.*, vol. 12, no. 1, pp. 59–90, 2017, doi: 10.3233/AO-170177.

[18] G. Tebes, D. Peppino, P. Becker, G. Matturro, M. Solari, and L. Olsina, "Analyzing and documenting the systematic review results of software testing ontologies," *Inf. Softw. Technol.*, vol. 123, p. 106298, Jul. 2020, doi: 10.1016/J.INFSOF.2020.106298.

[19] N. F. Noy and D. L. McGuiness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Stanford Knowledge Systems Laboratory*, doi: 10.1016/j.artmed.2004.01.014.

[20] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado, "Towards the establishment of an ontology of software testing," *18th International Conference on Software Engineering and Knowledge Engineering, SEKE 2006*, May 2014, pp 522-525.

[21] [1] H. Zhu, "A framework for service-oriented testing of web services," in Proceedings - International Computer Software and Applications Conference, 2006, vol. 2, doi: 10.1109/COMPSAC.2006.95.

[22] [2] X. Bai, S. Lee, W. T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," 2008, doi: 10.1109/ICWS.2008.111.

[23] [3] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," Web Semantics, vol. 5, no. 2 2007, doi: 10.1016/j.websem.2007.03.004

[24] M. A. A. Abdulhak, "An ontology-based approach for test case management system using Semantic Technology," PhD thesis, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia, 2013.