An Ontology-based Knowledge Management System for Software Testing

Shanmuganathan Vasanthapriyan[†], Jing Tian[*], Dongdong Zhao[‡],Shengwu Xiong[§] and Jianwen Xiang[*]
Hubei Key Laboratory of Transportation Internet of Things
School of Computer Science and Technology
Wuhan University of Technology
Wuhan, P.R. China
Email: priyan@appsc.sab.ac.lk , {jtian*,zdd,xiongsw,jwxiang*}@whut.edu.cn

***Abstract*—Software testing is an important activity in quality assurance and it generates large amount of knowledge. Software testers need to gather domain knowledge to be able to successfully conduct a software testing activity. Not having a proper knowledge base within its own context by software testing environments cause software testers to query limited knowledge available or consult peer software testers, which would greatly impact on their decision-making process. Ontologies emerge as one of the more appropriate knowledge management tools for supporting knowledge representation, processing, storage and retrieval. Given great importance to knowledge for software testing, and the potential benefits of managing software testing knowledge, using semantic web technologies, ontology based knowledge management system is developed. A Software testing knowledge sharing ontology is designed to describe software testing domain knowledge. SPARQL is used as the query language to retrieve software testing knowledge from the semantic storage. Both Ontology experts and non-experts evaluated the developed ontology. We believe our software testing ontology can support other software organizations to improve the sharing of knowledge and learning practices.**

***Keywords*—software testing ontology, software testing knowledge, ontology based knowledge management system, knowledge sharing.**

[1] [2]

I. Introduction

Software testing is a sub area of software engineering which is also a knowledge intensive and collaborative activity[1][2]. Knowledge can be applied to different testing tasks and purposes. Since software development is an error prone task, to achieve quality software products, Validation and Verification should be carried throughout the development[3]. As software testers, they would be familiar with the several software testing methods and considerably aware of the software development models, need information relevant to their context. For instance, software testers may either need assistance on a test case design information relevant to a similar project which was handled previously for testing purposes or to design a test case. Moreover, this information would also have a greater impact on their decision-making process. Importantly, during the software testing activities a huge amount of knowledge is being continuously produced and consumed. But, the approaches are limited and less employed to capture this knowledge and manage inside the organization to facilitate software testers. Our previous study results revealed some of the issues such as, outdated knowledge in the repositories, un structured internal documents and varied formats, less accessing facilities and lack of targeted delivery methods. Hence, efficient mechanisms for capturing, representing, reusing, and sharing the software testing knowledge involved are sorely needed.

According to Gruber[4], an ontology is an explicit specification of a conceptualization. Ontology provides a structured view of domain knowledge and acts as a repository of concepts in the domain. Recently, ontologies and Semantic Web technologies have received more attention and been gradually used in the knowledge representation[5]. Given great importance to knowledge for software testing, and the potential benefits of managing software testing knowledge, using semantic web technologies, ontology based knowledge management system is developed.

The context has also been decided to confine the study to a particular Sri Lankan software development company. The key reasons are based on the geographical location of the researcher, practicality and ease of access to those software development companies and comparability of research data due to company's same jurisdiction, same economic and regulatory regimes governing their operation. Further, we briefly explain each type of the high-level concepts based on IEEE 829-2008[6], also known as the 829 Standard for Software and System Test Documentation and ISTQB (International Software Testing Qualifications Board)[7]. Even though the standard specifies the procedures of software testing, we have also included what software company is stipulating in it's practice.

The remainder of this paper is organized as follows. Succinct analysis of related research is presented in the second section. Section 3 discusses software testing ontology in detail. Development of knowledge management portal to manage software testing knowledge is discussed in Section 4. Section 5 discusses the evaluation of the ontology developed under two points-of-view: domain experts and non-experts. Finally, Section 6 concludes the paper and presents directions for future work.

[1]DOI Reference Number: 10.18293/SEKE2017-020
[2]*Corresponding Authors

II. Related Work

A number of overviews of work on knowledge management in software testing have previously been published. The research carried out by Wei and Ying[8] discussed the proposal of implementing test knowledge management framework in iDEN phone software system testing and how the knowledge management approach can benefit the testing team in terms of cost and productivity. A reusable test case knowledge management model is proposed by Li and Zhang [9] to support the knowledge reuse based on the ontology representation of reusable test cases so that the test engineers can retrieve and reuse test cases flexibly. Douglas [10] proposed to investigate an open-standard based approach to the sharing of test results in the form of digital objects. Such an approach would not only reduce the needless replication of tests that occurs when there are no public records of previous tests conducted, but it would also allow the accumulation of a good deal of evidence to support certain usability design patterns and guidelines. Reference Ontology on Software Testing (ROoST) which was developed managing relevant knowledge to reuse is difficult and it requires some means to represent and to associate semantics to a large volume of test information[11].Bajnaid[12] proposed an SQA ontology that represents both domain and operational knowledge which provides consistent support to communicate between people and software agents but does not eliciting anything related to software testing process.

III. Designing of Software Testing Domain Ontology

A. Describing the Software Testing Process

Software testing process contains *TestPlanning* for planning tests, *TestCaseDesign* for test case construction, and test execution *TestExecution* for execution of test cases and producing *ActualResult*, and *TestResultAnalysis* for analysis and evaluate the test results [13]. In addition, our software testing process includes *Test Design Techniques*, *Test Levels*, *Artifacts*, *TestEnviorment* (includes hardware, software and Human resources) and *Static Testing Techniques*.

A *TestPlan* is produced during the *TestPlanning* activity. TestCaseDesign activity targets to design a *TestCase*. During the *TestCaseDesign*, if a *Test Design Techniques* is used then the following axiom can be defined for *TestCaseDesign* to design any *TestCase* as follows.

$\forall$ *tc:TestCase, tdt:TestDesignTechnique , tcd:TestCaseDesign hasDesignAccordingTo(tc,tdt) and isGeneratesTestCase(tcd,tc)*$\longrightarrow$ *isAdopts (tcd,tdt)*

Several Artifacts have been used to derive test cases in software testing which describes the functionalities, architecture, and design of software. Such Artifacts are used as *TestCaseDesignInput* during the software *TestCaseDesign* activity and the output is test cases. Test cases can be documented as described in the IEEE 829-2008 documentation. The document that describes the steps to be taken in running a set of tests (and specifies the executable order of the tests) is called a test procedure in IEEE 829. Besides, Test case contains a set of input values (*TestCaseInput*), execution preconditions, expected results (*Expected Result*) and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

That is, a test case targets to test a *Code To be Tested* . *Code To be Tested* can be any programs, modules, and the whole system code. Furthermore, *Test Code* is a portion of code that is to be run for executing a given set of test cases, contain three subtypes such as *Test Script, Driver* and *Stub*.

Test Execution activity executes any Test case. Test Execution requires as input the *Test Code* to be run and the Code To Be Tested. Notably, both *Test Code* and *Code To Be Tested* are needed for Test Execution activity. This can be illustrated in an axiom as follows.

$\forall$ *te:TextExecution, tc:TestCase hasExecutesTestCases(te,tc)* $\longrightarrow$ *(* $\exists$ *tcode:TestCode, CodeTobeTest: CodeToBeTested) uses (te,tcode)* $\bigwedge$ *isContainedOf(tcode,tc)* $\bigwedge$ *uses (te,CodeTobeTest)* $\bigwedge$ *hasCodeTobeTested(tc,tcode)*

The output of the *Test Execution* activity is the *Test Result*. Each of the *Test Result* is particularly related to a TestCase. *Test Result* may be related to an *Actual Result*, for a particular *TestCaseInput* and *ExpectedResult*. *Test Execution* also can be elaborated necessary and sufficient axioms in Protégé 5.1 as follows.

hasExecutesTestCases only (TestCase and (hasTestCaseExpectedResult some ExpectedResult))

A test execution can run and achieve a result (*Actual Result*), but it can also fail, and generating a *Test Issue* . Thus, a Test Result contains either an *Actual Result*, or *Test Issue* or both. This can be expressed using the following axiom.

$\forall$ *TestR:TestResult* $\longrightarrow$ *(*$\exists$ *ActResult:ActualResult, Issue: TestIssue) (ActualResult(ActResult)* $\bigvee$ *(TestIssue(Issue))* $\bigwedge$ *isGenarates(ActResult,Issue)*

Further, *Test Issue Report*, could report this event in detail, which requires investigation. Finally, during a Test Result Analysis, *Test Results* are analyzed and a *Test Analysis Report* is produced.

B. Ontology Engineering Approach

Ontology engineering approach investigates the principles, methods and tools for initiating, developing and maintaining ontologies[14]. In literature, many methodologies have been proposed until now to build an ontology[15], [16], [17], but we considered the Grüninger and Fox methodology[18], which considered a formal approach to design ontology as well as providing a framework for evaluating the adequacy of the

developed ontology. This methodology focuses on building ontology based on first-order logic (FOL) by providing strong semantics. In our scenario, we introduce Description Logic(DL) which is a decidable fragment of FOL since we are designing with OWL 2 Web Ontology Language[19] for sematic web. The widely-used Protégé system (http://protege.stanford.edu) has recently been extended with support for the additional constructs provided by OWL2[20].

C. Contextual Information

We describe "context specific"[21] to the software testers belonging to a leading software company in Sri Lanka and the approach which will be used to design the ontology to provide context-specific information and knowledge to software testers. To identify software tester's context clearly, we have extracted domain specific knowledge of software testing ontologies from existing literature and interviewed the software testing experts belonging to a particular company.

D. Competency Questions(CQs)

Competency Questions (CQ) are a set of questions that the ontology must be capable of answering using its axioms[18]. Thus, these CQs work as requirement's specification of the software testing ontology. With a set of CQs at hand, it is possible to know whether an ontology was created correctly, if it contains all the necessary and sufficient axioms that correctly answer the CQs. Some of the CQs used are shown in Table III.

TABLE I
TESTER'S INFORMATION NEEDS IN CONTEXT (I.E. COMPETENCY QUESTIONS)

Tester's Information Needs in Context (i.e. Competency Questions)
List out the human resources available in a testing activity?
Suitable test case design techniques for a given scenario?
What are the Test Levels described in software testing?
To which Test levels a particular TestDesignTechnique could be applied?
What kind of test automated software tools are available for use in your testing process?
Which Suitable Hardware Resources available for a given TestPlan?
What is the project name in which particular testing activity occurred?
What are the testing artifacts used in a testing activity?
What are the testing artifacts produced in a testing activity?
What are the testing objects used in a testing activity?

E. Ontology Components

At the first instance, high-level ontology concepts, their properties and their relationships should be identified. The basic high level ontology concepts are identified as Test Environment ,Testing Activities, Static Testing Techniques, Test Design Techniques, Test Objects, Test Level, Testing Artifacts and Organizational Team. For example, the concept Organizational Team having the properties of TeamID, TeamName, Team Size and Team Type. Secondly, sub classes of the high-level ontology concepts, their properties and relationships are also defined. For example, Test Environment has Human Resource, Software Tools and Hardware as it's sub classes. These sub classes are related to their superclass by is_ a relation. Artifact concept is mapped to disjoint and equivalent in OWL subclasses. During the modeling of software testing domain ontology, some special types of axioms such as Instantiation, Assertion, Subsumption, Domain, Range and Disjointness are included. The classes have been created in Protégé 5.1 can be described by necessary and sufficient conditions as illustrated in Table II.

TABLE II
NECESSARY AND SUFFICIENT CONDITIONS WRITTEN IN PROTÉGÉ 5.1

Testing Activities	Axioms written in Protégé 5.1 for Equivalent Classes
TestCaseDesign	(isGeneratesTestCase only TestCase) and (isGeneratesTestCase min 1 TestCase)
TestExecution	hasExecutesTestCases only (TestCase and (hasTestCaseExpectedResult some ExpectedResult))
TestPlanning	(hasCreatesTestPlan some TestPlan) and (hasCreatesTestPlan only TestPlan)
TestResultAnalysis	hasEvaluatesTestCases some (TestCase and (hasTestActualResult some ActualResult))

IV. DEVELOPMENT OF SOFTWARE TESTING KM PORTAL

As mentioned before, ontologies are powerful mechanism for representing knowledge presented in semantic web. The ontology and semantic web technologies provide powerful reasoning capabilities. In this section, we describe the ontology based knowledge management system to manage software testing knowledge, built upon the Java J2EE distributed component environment.

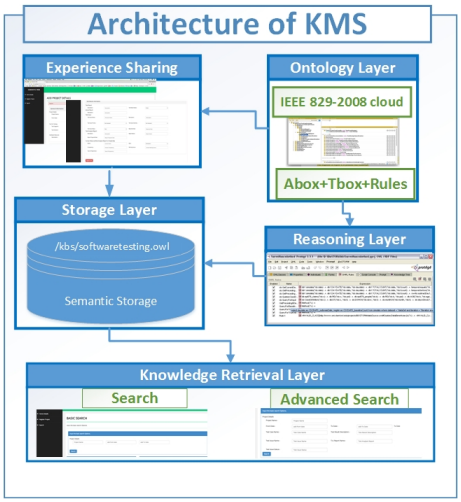

Fig. 1. Design of Knowledge Management Portal

Our KM Portal design consists of Experience Sharing, Ontology, Knowledge Retrieval, Storage and Reasoning Layer and is shown in Figure 1. The class diagram of Knowledge Sharing Portal is shown in Figure 2. We have implemented as a starting point of specifying the Knowledge Management Portal. Importantly, our presented concepts, properties and

relationships here are identified according to the characteristics of the particular organization's software testing environment.

A. Ontology Layer

The ontology layer has our ontology which includes domain rules, axioms etc. Using the Protégé Ontology Editor 5.1, these concepts and their relationships were partly described in section III.

B. Experience Sharing Layer

Through the Experience Sharing layer, software testers can annotate their testing knowledge with the support of ISTQB and IEEE 829-2008 terms. Once the knowledge is shared, such knowledge is transformed by the semantic data generator into the semantic data in a machine-understandable format of triple structure.

C. Storage Layer

We used Triple-store, which stores RDF triples and are queried using SPARQL. Jena TDB[22] has been selected to use in this study because it is a component of Jena for RDF storage and query. Importantly, it supports the full range of Jena APIs and can be used as a high performance of RDF store on a single machine.

D. Reasoning Layer

The Semantic Web Rule Language (SWRL) is based on a combination of OWL with the Rule Markup Language which provides inference capabilities[23] from existing OWL ontology. Rules in SWRL reason about OWL instances in terms of OWL classes and properties. Importantly, such rules express more complex relationships and restrictions between concepts. Software Testing rules were generated with Protégé SWRL Editor that is a plugin in Protégé environment and with the support of the Jess Rule Engine.

E. Knowledge Retrieval Layer

To show how our ontology can be used to share software testing knowledge collected from software testers, the Knowledge Retrieval Layer includes two functionalities that use Semantic Web technologies: (1) basic search, and (2) Advanced Search. SPARQL(SPARQL Protocol And RDF Query Language) has been used as the query language to retrieve software testing knowledge from the semantic data storage. The basic search provides a simple triple pattern matching service, which is one of the most frequently used functions for searching documents in the Semantic Web. Besides, Advanced Search Option includes, logical operators (AND or NOT or OR), so that user can combine different options to retrieve knowledge.

V. Ontology Evaluation

The quality of an ontology should be verified and validated before it is used in practice to avoid defects[24]. Further, such validation process will prevent contain ontology with anomalies or pitfalls, inconsistent incorrect and redundant information. Our evaluation of the ontology is carried out in three methods such as internal(using reasoners and OOPS!), ontology expert method and non-expert methods. Importantly, Protégé inbuilt reasoners such as FaCT++ 1.6.5 and HermiT 1.3.8.413 were used to check the internal consistency and inferences. OOPS!(http://oops.linkeddata.es/) is an online ontology evaluator has been used for our context to detect potential pitfalls that could lead to modelling errors, before ontology has been deployed in the end-user application. This method evaluates human understanding, logical consistency, modelling issues, ontology language specification, real world representation and semantic applications from the developed ontology[24]. Table III summarizes the pitfalls encountered, along with a brief description and the way each one of them was handled. There were three levels such as Critical, Important and Minor. Critical level is very crucial and it must be corrected in order to avoid ontology inconsistency. To make ontology nicer both Minor and Important cases were corrected.

TABLE III
Pitfall Description and solution Proposed

Pitfall	Description	Solution Proposed
Missing Annotation (256 cases | Minor)	Ontology terms lack annotation properties that could improve the understanding of the ontology	Included the ontology annotation properties
Missing domain or range in properties (7 cases |Important)	Relationships and (or) attributes without domain or range are included.	Restored missing domains.
Missing Inverse Relationships (14 cases| Minor)	When a relation has non-inverse relationship defined	Included Missing inverse relationships
Defining wrong inverse relationships. (6 cases | Critical)	Relationships are defined as inverse relations when they are not necessarily inverse.	They were removed to improve the expressiveness

Expert evaluation activity is performed by two ontology experts who have a good understanding of software testing. Even though there were many methods discussed to evaluate ontologies, our ontology experts considered Vocabulary, Syntax, Structure, Semantics, Representation and Context to perform evaluation[25]. The main objectives of expert evaluation are (a) whether the software testing ontology meets its requirements, standards, representation of concepts, relationships among them (b) coverage of the software testing domain and (c) checking for internal consistencies. The following suggestions and improvements were highlighted by the ontology experts: (a) Need of Annotations, renaming of concepts to standard methods. (b)Improving the relationship names, removing redundant relationships, some of the relationship were not mentioned, adding of association relationships which were missing. All such suggestions, comments were taken into consideration and implemented in the software testing ontology.

To carry out our first industrial application based evaluation from software testing experts, the developed Knowledge Management Portal was hosted locally inside one software company in Sri Lanka. A separate questionnaire was designed

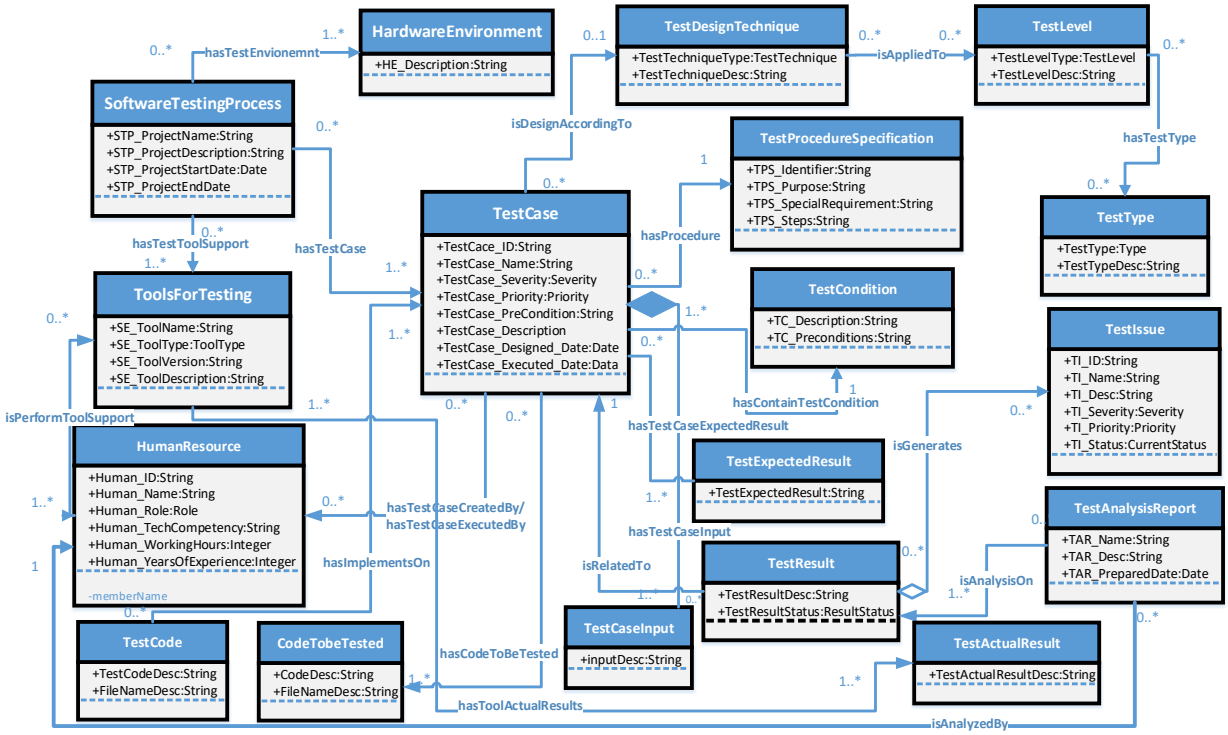

Fig. 2. Class Diagram of Knowledge Management Portal of Software Testing Knowledge

TABLE IV
SUMMARY OF THE SURVEY AND KEY QUESTIONS

Problem Identified from Survey - in the context of Software Testers (Beginning of the study)	**Feedback from the software testers (After developing Ontology and Implementing KM Portal)**
Is the knowledge in the knowledge repositories precise?	The large majority (66.67%) of the respondents from group A believed (Strongly Agree or Agree) that the provided content in Web Portal is precise. Only 44.44% of the respondents from group B believed (Strongly Agree or Agree) that knowledge in the knowledge repositories is precise.
To what extend is the software testing process incorporated?	Results illustrates that responded software testers have different views about the software testing process.
To what extend Internal documents are categorized using a standardized classification?	More than (75%) of software testers participated believed that the standards have been maintained
To what Extend Search or Retrieval Functionalities of developed KM portal support software testing?	Group A (41.66%) of the respondents believed (Strongly Agree or Agree) that the search or retrieval functionalities are adequate while from group B this value is 55.56%.
To what Extend Knowledge Sharing Functionalities of developed KM portal support software testing?	41.66% respondents from group A believed (Strongly Agree or Agree) that the knowledge sharing functionalities are adequate. 66.67% of the respondents from group B agreed(Agree) with the existing KM portal.
To what extend do you think it useful applying KM Practices in Software Testing?	Both groups (A- 41.67%, B-55.56%) believed with such practices and such results leads to conclude at least the importance and the need of KM practices for their daily activities.

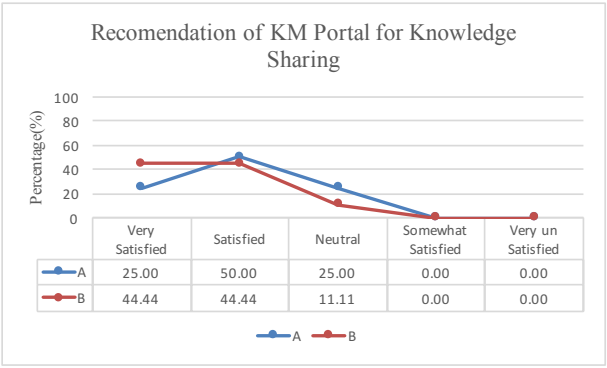

Fig. 3. Recommendation to use Ontoloy-based KM Portal

in English with five-point Likert-type scale to capture respondents' self-reported attitudes where respondents had to make their level of agreement such as; Strongly Agree, Agree, No Idea, Disagree and Strongly Disagree. Scores 5, 4, 3, 2, and 1 were assigned respectively for the above-mentioned categories. The profiles and demographics of the participants (Employed Group, work experience, job description, and qualification) were questioned first and continued with questions focused to check whether developed ontology was able to (a) express software testing knowledge (b) support software testing knowledge sharing (c) support software testing knowledge retrieval and (d)User Satisfaction. We limited the time period to ten (10) days to collect the questionnaire data. This was performed by

two software testing groups working on a similar information system project for comparison and in depth understanding. Importantly, a prior training session was conducted through recorded video to make software testers familiar with the Knowledge Management Portal.

The results are summarized in Table IV. When asked to evaluate the content of the KM Portal, Group B believes that correct contents have been included than the Group A. All of the software testers participated believed that the standards have been maintained, but a few have included some extra comments, such as inclusion of some vocabularies, parameters in Test Level and Test Types. Moreover, when asked about their satisfaction on the factors related to user manipulation, personalization and knowledge community, both groups have mostly responded neither satisfied nor unsatisfied. Notably, a very few have been very much satisfied with the user manipulation or knowledge community of the KM Portal. To conclude, overall 80.95% (See Figure 3) of the respondents would like to recommend the use of such KM Portal among the software testers for knowledge sharing and knowledge retrieval.

VI. Discussion and Conclusion

This research presents software testing ontology to represent software testing domain knowledge which includes software testing concepts, properties and their relationships. The implemented an ontology-based KMS based on semantic web technologies provides facilities for software testers to share their knowledge and experiences. Basic search and advanced search operations are used to retrieve knowledge. The implemented ontology was validated and evaluated before it is used. Ontology experts opinion was received to improve the ontology. The results from the industrial experimental investigation show that the proposed ontology-based KM Portal is adequate to support knowledge sharing and reuse, allowing: (a) knowledge representation and organization; (b) distributed knowledge inference and retrieval; (c) management of organizational knowledge on software testing. Our Portal addresses the existing key issues such as knowledge is not reaching the software testers due to its unstructured, incomplete, varied formats, and lack of targeted delivery methods. We believe our software testing ontology can support other software organizations to improve the sharing of knowledge and learning practices. In the future work, the reasoning engine with Query-enhanced Web Rule Language (SQWRL) will be incorporated into knowledge searching to support more precise and effective knowledge sharing.

Acknowledgment

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This work was partially supported by the National Natural Science Foundation of China (Grant No. 61672398), the Key Natural Science Foundation of Hubei Province of China (Grant No. 2015CFA069), and the Applied Foundamental Research of Wuhan (Grant No. 20160101010004).

References

[1] I. Rus and M. Lindvall, "Knowledge management in software engineering," *IEEE software*, vol. 19, no. 3, pp. 26–35, 2002.

[2] S. Vasanthapriyan, J. Tian, and J. Xiang, "A survey on knowledge management in software engineering," in *Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on*. IEEE, 2015, pp. 237–244.

[3] V. Santos, A. Goldman, and C. R. De Souza, "Fostering effective inter-team knowledge sharing in Agile software development," *Empirical Software Engineering*, vol. 20, no. 4, pp. 1006–1051, 2015.

[4] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[5] I. Horrocks, "Ontologies and the semantic web," *Communications of the ACM*, vol. 51, no. 12, pp. 58–67, 2008.

[6] S. S. E. Committee *et al.*, "IEEE standard for software and system test documentation," *Fredericksburg, VA, USA: IEEE Computer Society*, 2008.

[7] D. Graham, E. Van Veenendaal, and I. Evans, *Foundations of software testing: ISTQB certification*. Cengage Learning EMEA, 2008.

[8] O. K. Wei and T. M. Ying, "Knowledge management approach in mobile software system testing," in *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2120–2123.

[9] X. Li and W. Zhang, "Ontology-based testing platform for reusing," in *Internet Computing for Science and Engineering (ICICSE), 2012 Sixth International Conference on*. IEEE, 2012, pp. 86–89.

[10] I. Douglas, "Testing object management (TOM): A prototype for usability knowledge management in global software," in *International Conference on Usability and Internationalization*. Springer, 2007, pp. 297–305.

[11] E. F. Souza, R. Falbo, and N. L. Vijaykumar, "Using ontology patterns for building a reference software testing ontology," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International*. IEEE, 2013, pp. 21–30.

[12] N. Bajnaid, R. Benlamri, A. Pakstas, and S. Salekzamankhani, "Software quality assurance ontology from development to evaluation (s)." in *SEKE*, 2013, pp. 689–694.

[13] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[14] Y. Sure, S. Staab, and R. Studer, "Ontology engineering methodology," in *Handbook on ontologies*. Springer, 2009, pp. 135–152.

[15] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," 1997.

[16] Y. Sure, S. Staab, and R. Studer, "On-to-knowledge methodology (otkm)," in *Handbook on ontologies*. Springer, 2004, pp. 117–132.

[17] N. F. Noy, D. L. McGuinness *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.

[18] M. Grüninger and M. S. Fox, "Methodology for the design and evaluation of ontologies," 1995.

[19] W3C OWL Working Group, *OWL2 Web Ontology Language: Document Overview*. W3C Recommendation, 2009, available at http://www.w3.org/TR/owl2-overview/.

[20] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, "OWL2: The next step for OWL," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 4, pp. 309–322, 2008.

[21] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[22] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds, "Efficient RDF storage and retrieval in Jena2," in *Proceedings of the First International Conference on Semantic Web and Databases*. CEUR-WS. org, 2003, pp. 120–139.

[23] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean *et al.*, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member submission*, vol. 21, p. 79, 2004.

[24] M. Poveda-Villalón, M. C. Suárez-Figueroa, and A. Gómez-Pérez, "Validating ontologies with OOPS!" in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2012, pp. 267–281.

[25] D. Vrandečić, "Ontology evaluation," in *Handbook on Ontologies*. Springer, 2009, pp. 293–313.