IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 29, 2020, accepted November 6, 2020, date of publication November 10, 2020,
date of current version November 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3037188

Domain Ontology Construction and Evaluation for the Entire Process of Software Testing

ZHE SUN, CHI HU, CHUNLEI LI, AND LINBO WU
Institute of Computer Application, China Academy of Engineering Physics, Mianyang 621900, China
Corresponding author: Chi Hu (huchi16@nudt.edu.cn)

ABSTRACT As an important part of software engineering, software testing is a knowledge-intensive work. In the process of software testing, inconsistent knowledge expression, diverse knowledge carriers, and a few experienced people have mastered most of the knowledge, which hinders the transfer and sharing of domain knowledge. Ontology is widely used in various stages of software engineering to define the semantic relationship between relevant information and knowledge. To solve the problem of knowledge silo in the process of software testing, this article forms an Entire Process Ontology on Software Testing (EPOST). EPOST covers the concepts and relationships of software testing process information, software test object information, and software defect information. The concepts and terms in the ontology are extracted from ISTQB, SWEBOK, IEEE std.829-2008 standard, and IEEE std.610.12-1990 standard. We adopt a comprehensive ontology construction method based on Dev. 101 method and Methontology method. The developed ontology is successfully evaluated by using validation and verification tests. Ontology verification uses an improved FOCA evaluation method by adding a cohesion metric. The evaluation result infers that EPOST has a high quality of ontology and good domain coverage, and achieves the purpose of ontology construction. Finally, we make a case study on the role of EPOST in software testing process. The results show that ontology-based application in the software testing process can promote the sharing and transmission of domain knowledge, and improve the testing process and testing quality.

INDEX TERMS Ontology construction, software testing, domain ontology, knowledge management, ontology evaluation.

I. INTRODUCTION

Software testing is an important means of software quality assurance, and most of the cost of software development and maintenance is test cost [1]. The test objects and their business fields are quite different. Software testers not only need to master a large number of testing technologies and tools but also need to understand the business characteristics of the tested objects, which makes software testing knowledge-intensive work. Given the different personal experiences and abilities of testers, it is prone to incomplete consideration of abnormal situations during software testing requirements analysis, which leads to the problem of missing items in the test criteria and test case design. In the process of test execution, testers use manual or automated methods to execute test cases, diversifying the implementation and organization of test scripts [1]. Test cases are stored in different places in the form of documents, forms, records, etc., which also brings a lot of inconvenience to search. In terms of use case design, tool selection, report preparation, etc., there is no unified process and guidelines to guide, it is difficult to categorize and reuse, resulting in the uneven quality of test results. Therefore, domain knowledge isolation islands are common in the practice of software testing. Knowledge representation lacks a standardized and unified description, knowledge management lacks a unified and effective platform, knowledge search results are inaccurate, and most of the experience knowledge is in the hands of a few people [2]. Software testing domain knowledge involves a large number of concepts and the relationship between concepts. Only efficient and accurate management of knowledge can better improve the various stages of the software testing process [3], [4]. To effectively avoid the problem of knowledge silo in the software testing domain due to the large, scattered, and difficult to obtain test data, it is necessary to organize domain knowledge structurally and then express it formally.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhe Xiao.

The ontology contains rich semantic information, which can effectively reduce the ambiguity in conceptual understanding, and is of great help in improving the performance of many text processing-related tasks. At present, ontology is widely used in many tasks related to text processing, such as information retrieval, information extraction, information integration, data management, information recommendation, text classification and clustering, question and answer systems, etc., and have achieved good results. [5]. According to the currently widely accepted definition of ontology, ontology is a standardized and clear definition of conceptual form and the relationship between concepts [6]. An ontology usually consists of three parts: concepts, relationships between concepts, and axioms based on relationships. By reviewing the literature [3]–[17], it is found that researchers have done a lot of work in the field of software testing knowledge representation and ontology construction, but most domain ontology coverage is not enough and cannot be applied to the whole process of software testing, or can only be applied in a certain test activity.

Based on the previous research work, this article constructs an Entire Process Ontology on Software Testing (EPOST) to improve the knowledge isolation island problem. The EPOST ontology covers the software test objects, test methods, test techniques, test tools, test cases, test reports, software defects, and other information involved in the whole process of software testing. The ontology construction method adopts a comprehensive ontology construction method [18], and uses the Protégé tool for ontology editing [4]. Ontology evaluation uses HermiT [15] and OOPS! [19] to validate context, and uses an improved FOCA method for ontology verification [20], [21].

The rest of this article is divided into the following sections: Related works are presented in section 2. Section 3, describes the construction of EPOST. The ontology evaluation is presented in section 4. Section 5 describes a discussion about the evaluation results. Finally, section 6, shows the conclusion and future work.

II. RELATED WORK

The software testing process is composed of multiple serialized activities. When representing domain knowledge and constructing ontology, the relationship between each process should be considered as much as possible. The test case is a relatively important work content in the software testing process, so most of the literature has built the corresponding domain ontology around test cases [3], [4], [7]–[17]. To better support the running process of software testing, some domain ontologies built by some work also focus on test process, test technology, test activity, and test environment [3]. The content of knowledge representation needs a consistent explanation. The resources cited in the definition of terms such as concepts in the ontology of software testing are mainly International Software Testing Qualifications Board (ISTQB) [21], the Software Engineering Body of Knowledge (SWEBOK) [22]. The current typical ontology construction methods are most closely related to specific ontology design projects [18]. At present, the main ontology construction methods include TOVE method, Methontology method, skeleton method, KACTUS method, SENSUS method, IDEF5 method and Dev. 101 method [19]. These methods are composed of an overall process and operation rules of each step. In addition to the differences of various methods, an ontology construction method should include the following basic steps: first, acquiring knowledge, abstracting and refining knowledge, then expressing it in a way that the computer can understand, and then evaluating the quality of the ontology and maintaining updates. Some auxiliary technologies are usually used in the process of ontology construction. The ontology representation language is OWL, and the ontology editing tool is Protégé [4]. Besides, SWRL is often used to describe ontology reasoning rules, and HermiT is used as ontology reasoning tool [15].

As a knowledge-sharing model, the ontology provides great convenience for the information interaction between people and application systems in a specific field. It is also because of this, combined with knowledge engineering, natural language processing, and other technologies, the introduction of ontology technology in the software testing process can effectively reduce development costs, improve software quality, increase the degree of knowledge reuse, and promote knowledge inheritance [24]–[31]. Many process management factors are involved in software testing, and an effective way of expression and management is required. Introducing knowledge management technology to improve the software testing process, providing a standard structured knowledge expression and query method, can alleviate the inconveniences in the software testing process [29]. The combination of ontology technology and reuse theory can improve the reuse degree and retrieval efficiency of software test cases [10], [25], [27]. Using ontology technology to model knowledge with a hierarchical structure, and establish an ontology-based knowledge-sharing platform to provide semantic query, experts and non-experts can use this platform to improve software testing quality [15]. Combining the knowledge management model and using ontology technology to formally describe the software testing process can promote the sharing and dissemination of software testing knowledge [8], [14], [24], [26], and can also guide software performance testing [4], software GUI test [30].

Most of the above work focuses on the application of ontology technology to optimize a certain activity in the process of software testing. Some are the construction or formal representation of software testing knowledge ontology, and some are aimed at the application of ontology technology in a certain test activity or a specific step in the testing process. They fail to think about the overall situation of software testing and do not give a general plan. The software test ontology constructed in the literature [8], [24], [29], [31] can be used as a commonly controlled dictionary to provide back-bone information for the content of the knowledge base indexed by it, but they had not been evaluated or verified.

Literature [4], [10], [25]–[27], [30] used the ontology to query and reuse test cases, which guided software performance testing or GUI testing, but the scope of a domain covered by ontologies was limited and not comprehensive enough. Software testing is a knowledge-driven work process. Knowledge silo is mainly caused by the inconsistent data format and the lack of effective sharing and query mechanisms. Improving in a specific step does not contribute much to the overall efficiency improvement. Ontology needs to be verified and evaluated on its quality before use to ensure that the content of the ontology is correct and meets application requirements. [4] used the criteria extracted from the Protégé tool to compare ontologies, [14] used ontology competency questions to verify ontology quality, the ontology developed by [15] was evaluated in three methods such as internal, ontology expert method and non-expert methods. But most of them do not evaluate the structure of the ontology.

Based on the above review and analysis, this study constructs a software testing domain ontology, covering the information driving and assisting the software testing process. We use the multiple-criteria based method to evaluate the quality of the ontology, including ontology validation and ontology verification.

III. ONTOLOGY CONSTRUCTION

The ontology construction method is to guide developers to construct ontology according to the required requirements and basic steps, which directly determines the ontology's knowledge representation and logical reasoning. There are two methods to construct the domain ontology of software testing knowledge, one is to build a new domain ontology directly, and the other is to expand the existing domain ontology. When choosing ontology construction methods, we should adopt the most appropriate method according to the actual situation, or integrate the advantages of various methods to improve and optimize the existing construction methods. In view of the existing common ontology construction methods, [18] compared them by some metrics, such as the life cycle, the technology adopted, the details of the method, the characteristics of the method, and the application field of ontology. They considered that the relatively complete and mature methods are Dev. 101 method and Methontology method. Based on the integration of Dev. 101 method, Methontology method, and IDEF5 method, [18] proposed a comprehensive ontology construction method. This article uses the method of [18] to construct ontology. As shown in Figure 1, this method includes six steps: ontology requirement analysis, reusable ontology investigation, core concept establishment, concept classification level establishment, class definition, and attribute creation, ontology evaluation, and evolution.

A. ONTOLOGY REQUIREMENT ANALYSIS

First of all, the goal and scope of ontology construction must be determined. The field of this research is software testing process information. The goal of domain ontology construction is to build a domain ontology that can cover the whole process of software testing, so that the software testing process information becomes more structured, to facilitate the acquisition and management of knowledge.

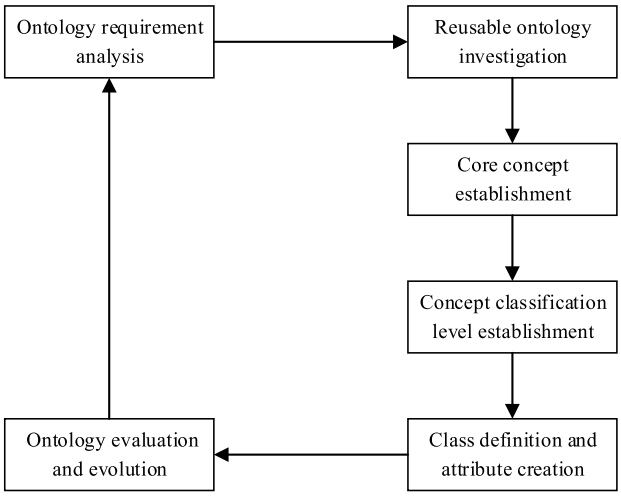

FIGURE 1. Ontology development process.

To determine the scope of the domain, you can use the ontology capability analysis method, that is, to determine the domain boundary through the following Competency Questions (CQs). What fields and stakeholders are involved in the ontology? How to apply the information of the tested object to test? How to structure the knowledge of the software testing process? What activities and artifacts are covered in the software testing process? What resources are needed for software testing, and what technologies and tools are used? How to analyze software defects found in software testing?

The scope of the domain ontology constructed includes all information that can drive and assist the software testing process, that is, the software test object, test method, test technology, test tools, test cases, test reports, software defects, and other information used by software testing engineers in the software testing process.

B. REUSABLE ONTOLOGY INVESTIGATION

One of the features of ontology is sharable and reusable. Checking the possibility of scalable and reusable existing ontology is an important way to ensure that this feature of ontology can be realized. Reusing existing ontology can also improve the efficiency of ontology construction. At the same time, the ontology is also scalable and can be updated at any time. It is also the architecture basis for reusing existing ontology. To find research that has a rich ontological coverage that includes non-functional and functional requirements, [3] conducted a Systematic Literature Review (SLR) to identify, evaluate and summarize the available research about conceptualized software testing ontologies selected. A new ontology TestTDO was developed [3]. Similarly, in order to check whether the existing ontologies can be reused, this article extracts their relevant information and makes a comparative

TABLE 1. Summary of Extracted Information for the Related Conceptualized Software Testing Ontologies in [3].

Conceptualized Ontology (Publish. Year)	Mainly Used Terminology [Glossary, Taxonomy, Ontology]	ST concepts covering?	SF concepts covering?	SUT concepts covering?
STOWS, Zhu et al.[7] (2005)	Not specified	Yes	No	Yes
OntoTest, Barbosa et al. [8] (2008)	Glossary by IEEE 829; ISO/IEC 12207; The Software Process Ontology by Falbo & Bertollo; STOWS;	Yes	No	Yes
TOM, Bai et al [9] (2008)	UML 2.0 Testing Profile (U2TP)	Yes	No	Yes
Cai et al. [10] (2009)	Glossary by SWEBOK, ISO/IEC 9126	Yes	No	No
Sapna et al. [11] (2011)	Glossary by SWEBOK	Yes	Yes	No
Arnicans et al. [12] (2013)	Glossary by ISTQB	Yes	No	No
PTOntology, Freitas et al. [4] (2014)	Glossary by SWEBOK; IEEE Glossary of SE Terminology; IEEE Standard for Software Test Documentation	Yes	No	Yes
Asman et al. [13] (2015)	Glossary by ISTQB SWTOI; OntoTest;	Yes	Yes	No
ROoST, Souza et al. [14] (2017)	Glossary by IEEE 610-1990 & 829-1998; SWEBOK; ISTQB; ISO 29119; SP-OPL; E-OPL; UFO	Yes	No	Yes
Vasanthapriyan et al. [15] (2017)	Glossary by ISTQB; IEEE 829-2008	Yes	No	No
Vasanthapriyan et al. [16] (2017)	Glossary by ISTQB; IEEE 829-2008	Yes	No	Yes
RTE-Ontology,Campos et al. [17] (2017)	PROV-O Ontology	Yes	No	No
TestTDO [3] (2020)	Glossary by ISTQB; ISO 29119	Yes	No	Yes

analysis with our research objectives. As shown in Table 1, all of the them cover the concepts of SoftwareTesting(ST) sub-domain considering terms related to testing activities, artifacts, process, environments, tools, test levels, and techniques(or methods) [3]. Few ontologies consider the terms in the sub-domain of SoftwareFailure (SF) by using the words such as 'defect', 'failure' and 'bug' [3]. On the other hand, some ontologies have one term that refers to the sub-domain of SoftwareUnderTest (SUT). Some of them call it 'Code To Be Tested', 'System Under Test Concept', 'Artifact Under Testing', 'Subject Under Test'(1) and so on [3].

Analyzing the objectives of the selected ontologies, we can conclude that none of them is totally in line with our research objectives.

C. CORE CONCEPTS ESTABLISHMENT

This step is to determine the concepts involved in the information domain ontology of the software testing process, list all the important terms in the field, collect domain concepts, semantics, attributes, examples, etc., and establish a concept summary table after sorting and refining. The core concept dictionary, as the grading concept of the conceptual model, must meet the requirements of being unambiguous and covering the entire software testing domain knowledge. To prevent the existence of heterogeneity between subdomain ontologies at the same level, the description of the concept refers to ISTQB [22], SWEBOK [23], IEEE std.829-2008 [32], and IEEE std.610.12-1990 [33] standards on software testing.

Through the analysis of the software testing domain, it can be seen that the software testing information mainly includes: the structure, function, scale, module, and other related information of the software under test; test requirements, test plans, test cases, test problem reports and other related document information generated by software testing; information related to the failure type, failure cause, and failure impact generated during the software failure analysis process; related knowledge of testing techniques, methods, and experience used in the testing process.

TABLE 2. Portion of the Core Concept Dictionary.

	Concepts	
SoftwareTesting concepts	TestingActivity	TestingArtifact
	TestingLevel	TestPlanning
	TestingPhase	TestingProcess
	TestingEnvironment	TestingStep
	TestingProcedure	TestingTechnique
	TestingResource	TestingType
SoftwareFailure concepts	FailureAnalysis	FailureContainment
	FailureSeverity	FailureCause
	FailureEffect	FailureMode
	FailureOccurProportion	
SoftwareUnderTest concepts	SoftwareInformation	CriticalLevel
	SoftwareStructure	SoftwareModule

As shown in Table 2, the core concept dictionary table of this field ontology mainly includes information about the software under test, software testing process information, and software defect information.

D. CONCEPT CLASSIFICATION LEVEL ESTABLISHMENT

This step defines the domain concepts sorted out and analyzes the possible relationships between concepts, including explicit relationships and implicit ones. There are the top-down method, bottom-up method, and middle-out method to establish the hierarchical relationship of concept classification. Since we have established the core concept dictionary, this article will use the middle-out method to establish the concept classification hierarchy.

The concepts and related terms in this field are extracted from ISTQB [22], SWEBOK [23], IEEE std.829-2008 [32],

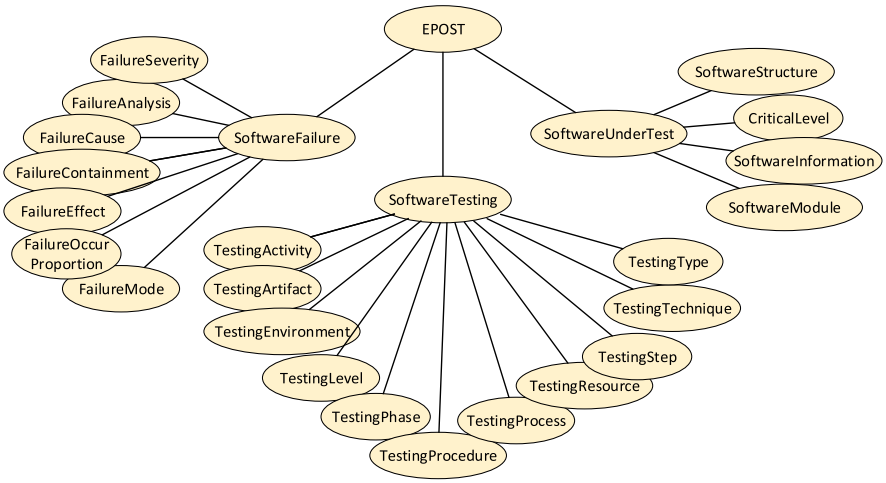

FIGURE 2. Hierarchy of concept classes in EPOST.

and IEEE std.610.12-1990 [33] standard. According to the concept definition and domain in the standard, the core concepts in Table 1 can be organized into a hierarchical structure as shown in Figure 2. Concept hierarchy is the framework of ontology, which needs to be enriched and extended through the attributes and relationships of concepts. The concept hierarchy with relation is a typical structure with semantic ability, which can better organize and express knowledge. With the development of ontology construction, the concept hierarchy may modify, add, and delete the concepts and their relationships in the concept dictionary.

After the hierarchical structure of the concept is defined, we can use 'is-a' or 'part-of' to express the taxonomic relationship between them. Then, as shown in Figure 3, we need to define the other non-taxonomic relationships to make EPOST have strong semantic capabilities. The software testing process is a set of serialized test steps, and each test step executes and maintains related test activities according to the test purpose. A test step can consume or generate some test artifact and require different test resources. The whole test process can be divided into different test stages according to different outputs. The test steps are selected and tailored according to the test requirements and purpose of the software under test. Testing different versions of software need to determine the corresponding test activities and processes. The software structure and software module of the tested software can determine the testing technology and testing tools. The test resources needed in the test environment need to refer to the software environment tested by the software under test. The test type and test level are related to the critical level of the software under test. The quality criteria of software under test is an important input of test criteria. Software test results include various software failures. Software failure analysis needs to understand the software module division, to carry out failure mode analysis, give failure causes, and formulate containment measures according to failure effect analysis.

E. CLASS DEFINITION AND ATTRIBUTE CREATION

A class in ontology is a collection of individuals that share some attributes and belong to the same group. Generally, each knowledge point is set as a separate class. The upper knowledge point is defined as the parent class according to the knowledge level, and the lower knowledge point is taken as the subclass. The highest level class represents the most abstract entity concept. Each subclass inherits the abstract characteristics of its parent class and is a more specific and smaller entity concept than its parent class. In this process, according to the pre-defined upper-level abstract parent class, the next level subclass is specified gradually.

The two attributes of a concept are the numerical attributes describing its structure information and the object attributes describing the relationship between concepts. Each attribute has its attribute name, which determines the class it describes. This process of defining classes and creating attributes is the representation of ontology. Ontology development often uses modeling tools or languages to define classes and create attributes. In this study, Protégé is selected as the ontology modeling tool, and OWL recommended by W3C is used as the ontology description language. The classes and attributes constructed are shown in Figure 4. In Figure 4, the solid line represents the subordination relationship, and the dotted line represents the association relationship.EPOST includes 217 concepts, 53 relational properties, 42 datatype properties, 278 SubClassOf axioms, 40 DisjointClasses axioms, and 241 AnnotationAssertion aximos. There are 143 concepts in SoftwareTesting sub-ontology, 55 concepts in SoftwareUnderTest sub-ontology, and 19 concepts in SoftwareFailure sub-ontology.

IV. ONTOLOGY EVALUATION

After the ontology is initially constructed, it can be evaluated and improved. Ontology evaluation is a process of the comprehensive evaluation of various factors affecting the quality of ontology based on various evaluation indicators

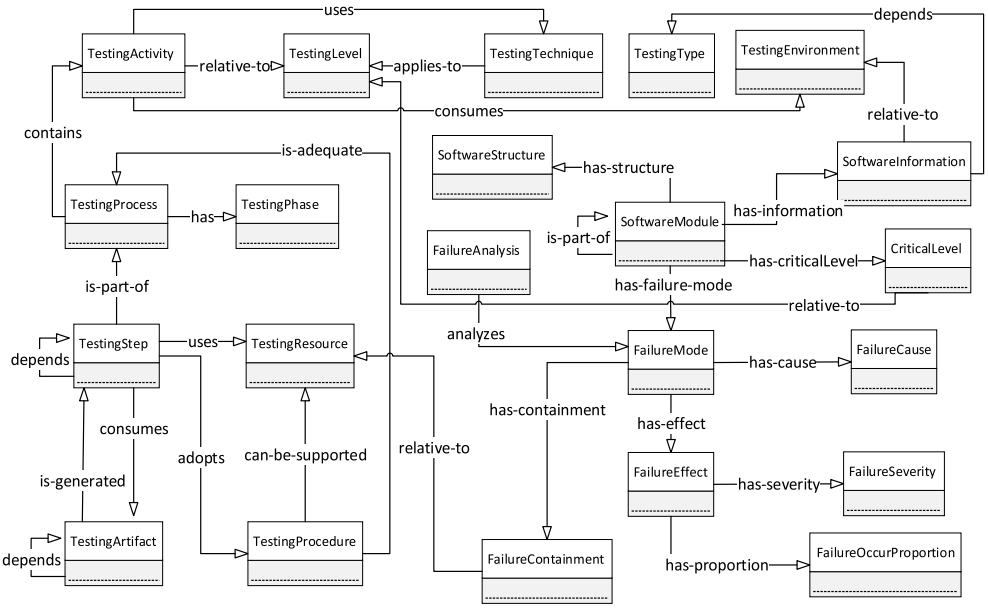

FIGURE 3. Non-taxonomic relationships of concept classes in EPOST.

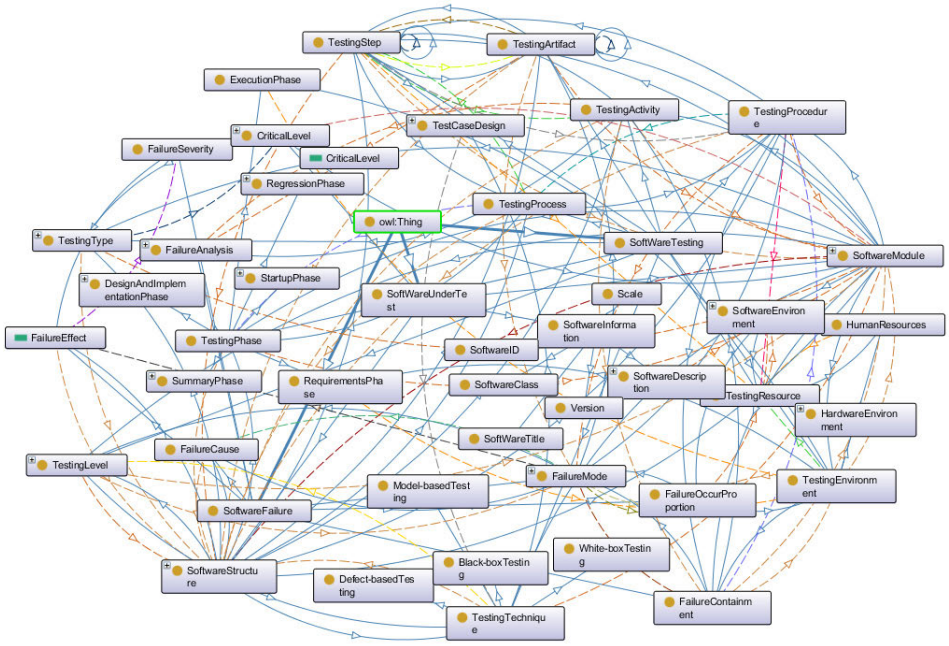

FIGURE 4. Part of EPOST on Protetge OntoGraf.

and scientific evaluation methods [34]. The content of ontology evaluation includes checking whether the objectives set in the requirement analysis stage are met, and verifying whether the ontology constructed is correct. Ontology evaluation activities involve three aspects: evaluation methods, evaluation indicators, and evaluation tools. The widely accepted ontology evaluation is the five criteria proposed by Gruber: clarity, consistency, extensibility, minimum coding preference, minimal ontology commitment. The main ontology evaluation methods include evaluation methods based on metric systems, evaluation based on tasks or applications, evaluation based on statistical analysis, evaluation based on logic or rules, and ''Golden-Standard'' evaluation methods. At present, the commonly used ontology evaluation tools are Ontoclean, Core, OOPS!, TEXCOMON, etc. [34].

A. INTERNAL CONSISTENCY CHECK

Semantic-based ontology reasoning tools use the initial set of axioms to infer the consistency of the logical sequence of the content, including consistency checking, concept satisfiability, classification, and realization [4]. To get an accurate reasoning result, different reasoning tools use different methods and strategies. During the construction of EPOST, we used the built-in reasoning tool HermiT1.4.3.456

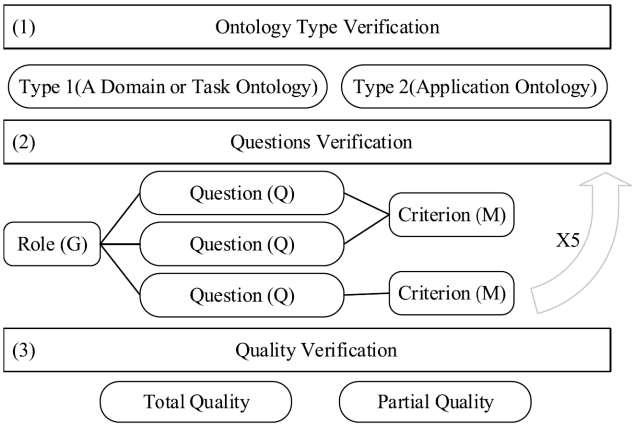

FIGURE 5. FOCA method.

in Protégé 5.5.0 to check internal consistency and reasoning results. After ontology reasoning, the reasoner shows no circularity errors, no partition errors, and no semantic errors in EPOST.

B. CONTEXT PITFALL DETECTION

OOPS! [19], which stands for Ontology Pitfall Scanner!, can help us to detect some of the most common pitfalls appearing when developing ontologies. OOPS! is an online automatic detection tool for ontology defects, and lists 29 ontology pitfalls. This tool evaluates ontology from six dimensions: human understanding, logical consistency, modelling issues, ontology language specification, real world representation, and semantic applications, and divides the results into three levels: critical, important, and minor according to the degree of impact. As shown in Table 3, the evaluation results of EPOST using OOPS! include the occurrence frequency, specific description, and modification suggestions of ontology pitfall. We have modified and improved the EPOST ontology according to the modification suggestions given in Table 3.

C. ONTOLOGY VERIFICATION

Among the criteria-based evaluation methods, FOCA [20] is a relatively mature method for evaluating the quality of ontology. FOCA includes determining the type of ontology, a questionnaire to evaluate the components, a framework to follow, and a statistical model that calculates the quality of the ontology. FOCA goes through three verification steps, as shown in Figure 5 [20]. To get a more accurate evaluation result, [21] added an ontology cohesion metric to the FOCA evaluation metrics system to investigate the closeness of ontology concepts. We use the ontology verification framework in [21] to evaluate the quality of EPOST.

1) ONTOLOGY TYPE VERIFICATION

The FOCA defines two types of ontology, one is a domain or task ontology, and the other is application ontology. The questions that need to be answered are also different. The EPOST focuses on information that can drive and assist the software testing process, is a domain ontology (Type 1). The proposed ontology describes, respectively, the vocabulary of a genetic domain. Therefore, Question 4 should not be verified for Goal 2 shown in Table 4, because it only asks for specific domains.

2) QUESTIONS VERIFICATION

In this step, it needs to answer the questions following the goal/question/metric (GQM) approach for the improved FOCA methodology is shown in Table 4. When a cohesion metric is added, this method contains 5 goals, 14 questions, and 6 metrics [21]. Because the EPOST is a type 1 ontology, 13 out of 14 questions (should answer Q5 instead of Q4) should be answered. These 13 questions serve for 5 goals, and each question fulfills one of the ontology quality criteria. Each question has a description that explains how to verify it, and each answer has a grade ranging between 0 and 100. Depending on the answers to these if questions regarding the developed ontology, the EPOST received a grade for each metric, as shown in Table 4 [35]. Therefore, the mean of the grades from each goal can be calculated.

3) QUALITY VERIFICATION

After verifying the questions, it needs to calculate the quality of the ontology. The quality of the ontology (a score in (0, 1)) is calculated by the beta regression models [20], as shown in (1). It can be calculated in two ways, total quality and partial quality. Because most goals are considered in the evaluation, this article used total quality verification.

$$\hat{\mu}_i = \frac{\begin{array}{l}\exp\{-0.44+0.03(Cov_S \times Sb)_i+0.02(Cov_C \times Co)_i \\ +0.01(Cov_R \times Re)_i + 0.02(Cov_{Cp} \times Cp)_i \\ -0.06LExp_i - 25(0.1 \times Nl)_i\}\end{array}}{\begin{array}{l}1+\exp\{-0.44+0.03(Cov_S \times Sb)_i+0.02(Cov_C \times Co)_i \\ +0.01(Cov_R \times Re)_i + 0.02(Cov_{Cp} \times Cp)_i \\ -0.06LExp_i - 25(0.1 \times Nl)_i\}\end{array}} \tag{1}$$

To calculate the total quality: Cov_S is the mean of the grades from Goal 1. Goal 1 contains three sub-questions, therefore the grade of Goal 1 is the mean between the three sub-questions. Finally, Cov_S is the mean between Question 1, Question 2 and Question 3. Cov_C is the mean of the grades from Goal 2. Because the EPOST is a domain ontology, Cov_C is the mean between Question 5, Question 6, and Question 7. Cov_R is the mean of the grades from Goal 3. Cov_{Cp} is the mean of the grades from Goal 4. LE_{xp} is the variable for evaluator experience, with 1 being very experienced and 0 being not experienced at all. N_l is 1 only if some Goal is impossible for the evaluator to answer all the questions. $S_b = 1$, $C_o = 1$, $R_e = 1$, $C_p = 1$, because the total quality considers all the roles.

By substituting these values into equation (1), we can obtain the quality result. The result of the total quality is 0.9881 and it is near to 1, which shows the quality of the EPOST is high. Thus, the proposed EPOST was successfully

TABLE 3. Ontology Pitfall Description and Solution Proposed.

Pitfall	Dimension	Description	Solution Proposed
Creating unconnected ontology elements : 3 cases | Minor	Modelling issues	Ontology elements are created isolated, with no relation to the rest of the ontology.	Removed the isolated class.
Merging different concepts in the same class : 1 case | Minor	Modelling issues, Human understanding	A class whose name refers to two or more different concepts is created.	Removed the ambiguous concept to improve the expressiveness.
Missing annotations : 241 cases | Minor	Human understanding	This pitfall consists in creating an ontology element and failing to provide human readable annotations attached to it.	Included the ontology annotation properties.
Missing domain or range in properties : 62 cases | Important	Modelling issues, Human understanding	Object and/or datatype properties without domain or range (or none of them) are included in the ontology.	Restored missing domains.
Inverse relationships not explicitly declared : 32 cases | Minor	Modelling issues, Human understanding	This pitfall appears when any relationship does not have an inverse relationship defined within the ontology.	Defined symmetric properties using owl:SymmetricProperty.
Using a miscellaneous class : 1 case | Minor	Modelling issues	This pitfall refers to the creation of a class with the only goal of classifying the instances that do not belong to any of its sibling classes.	Removed the miscellaneous class.

TABLE 4. Applying the GQM Approach [26] on EPOST.

Goal	Question	Metric	Grade	Mean
1. Check if the ontology complies with Substitute.	Q1. Were the competency questions defined?	Completeness	100	**100**
	Q2. Were the competency questions answered?	Completeness	100	
	Q3. Did the ontology reuse other ontologies?	Adaptability	100	
2. Check if the ontology complies Ontological Commitments.	Q4. Did the ontology impose a minimal ontological commitment? (**an application ontology**)	Conciseness	-	-
	Q5. Did the ontology impose a maximum ontological commitment? (**a domain or task ontology**)	Conciseness	75	**83.3**
	Q6. Are the ontology properties coherent with the domain?	Consistency	100	
	Q7. Was the ontology cohesion metric value acquired?	**Cohesion**	**75**	
3. Check if the ontology complies with Intelligent Reasoning.	Q8. Are there contradictory axioms?	Consistency	100	**100**
	Q9. Are there redundant axioms?	Conciseness	100	
4. Check if the ontology complies with Efficient Computation.	Q10. Did the reasoner bring modeling errors?	Computational efficiency	100	**87.5**
	Q11. Did the reasoner perform quickly?	Computational efficiency	75	
5. Check if the ontology complies with Human Expression.	Q12. Is the documentation consistent with modeling?	Clarity	100	**75**
	Q13. Were the concepts well written?	Clarity	75	
	Q14. Are there annotations in the ontology that show the definitions of the concepts?	Clarity	50	

validated and verified.

$$\hat{\mu}_i = \frac{\exp\{-0.44+0.03\times(100\times 1)+0.02\times(83.3\times 1)+0.01\times(100\times 1)+0.02\times(87.5\times 1)-0.06\times 1-25\times(0.1\times 1)\}}{1+\exp\{-0.44+0.03\times(100\times 1)+0.02\times(83.3\times 1)+0.01\times(100\times 1)+0.02\times(87.5\times 1)-0.06\times 1-25\times(0.1\times 1)\}} = 0.9881$$

D. ONTOLOGY-BASED APPLICATION

This section will focus on the role of ontology in software testing, and state how an ontology can be utilized for software testing.

1) ROLE OF ONTOLOGY IN SOFTWARE TESTING

The application aims at supporting testers' decisions with domain knowledge of technologies used in testing software, validating and recommending options according with the test environment, goals, tools and activities that tests might present [4]. In Figure 6, the black solid line represents the traditional software testing process, and the blue dotted line indicates the usage of EPOST domain knowledge as an important test support resource. After preprocessing the user's query request, the retrieval module conducts semantic reasoning, transforms the reasoning results into query language for retrieval, sorts them according to the matching degree, and returns the query results to the user as required [36]–[38]. The SoftwareUnderTest domain knowledge in EPOST helps testers analyze the tested object, identify test tasks, define test goals and test activities. Software testers refer to the domain knowledge and related rules in SoftwareTesting and SoftwareFailure, and testers can easily transform the general test objectives into a series of specific test conditions and test cases, and reuse test cases and test questions. For running test cases, tracking issues, and control version systems, ontology can be used to support test integration [29]. After the analysis of test results, it is necessary to evaluate whether there is new

knowledge generated, and update the useful knowledge to the corresponding ontology database or rules [36].

2) TESTING QULITY ENTROPY

EPOST provides knowledge for software testing activities, that is, it provides information for software testing as a source. The amount of information contained in domain ontology knowledge can affect the quality of software testing activities. By the calculation of the information entropy of problem sets in software testing, [36] proposed the concept of testing quality entropy and used it to measure software testing quality.

Testers design test cases with the help of software domain knowledge, complete the test and detect problems. For the testing problem set Q, there are detected problems and undetected problems. The two kind problems are independent of each other, so they can be described by a probability matrix, as shown in (2). In (2), the number of detected problems is q_1, and its occurrence probability is $p(q_1)$; the number of undetected problems is q_2, and its occurrence probability is $p(q_2)$. Q is the state space of the testing problem, $\sum p(q_i) = 1$.

$$\begin{bmatrix} Q \\ P(q) \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \\ p(q_1) & p(q_2) \end{bmatrix} \tag{2}$$

For any kind of testing problem q_i, the testing quality index $I(q_i)$ is defined as the negative value of its probability's logarithm. If the occurrence probability of testing problem q_i is $p(q_i)$, then its quality index can be calculated by (3).

$$I(q_i) = -\log P(q_i) \tag{3}$$

On the problem set Q, the mathematical expectation of testing quality index is taken as the measurement of software testing quality. This is the expression of entropy, which is called testing quality entropy [36]. According to the problem set Q in (2), its testing quality entropy can be calculated as shown in (4).

$$H(Q) = E\left[I(q_i)\right] = -\sum_{i=1}^{2} P(q_i) \log P(q_i) \tag{4}$$

By calculating the entropy of the state space of the problem set, the quantitative evaluation of the software testing quality is realized [36]. In fact, the number of undetected problems is difficult to figure out. Therefore, we postulate some conditions in the engineering application. The best case for the test team is to find all the problems, and the worst case is to find *50%* of the problems [36]. So, the numerical value of testing quality entropy is *0* for the best case, and *1* for the worst case.

3) EXPERIMENT SETUP

In this section, we design an experiment to verify the application effect of EPOST based software testing. For the experiment, testers carry out the activities in the traditional testing process and EPOST based testing process as shown in Figure 6. By measuring the quality of testing process and testing results, we quantify the software testing quality [36]

To reduce the influence of knowledge level and test ability on experimental results, we randomly selecte 6 testers who have been engaged in testing for 3 years and divided them into two test teams, group A and group B. They independently carry out test tasks, generate test artifacts according to the process requirements, record the test results. In order to collect statistical test data clearly and effectively, each test case is designed to detect only one problem at a time. In addition, the number of failed test cases is not merged with similar problems. In the experiment, group A use EPOST domain knowledge as support resources to carry out the test work, and group B carry out the test according to the traditional test process.

This article takes ATM software as the test object. ATM software is a non-embedded application software running on the desktop. The user account registration and login operation can be completed according to the interface prompt. After login, the functions of deposit, withdrawal, transfer, deposit query, withdrawal query, transfer query, balance query, password modification and other functions can be completed.

4) EXPERIMENT RESULTS

As shown in Figure 7 and Figure 8, the two test teams are quite different in terms of safety, performance, strength, reliability, availability and other key test types. We can infer that with the increase of test knowledge, the more test items and test cases are obtained, and the more comprehensive the analysis of the test types which are more difficult and have a greater impact on the software test quality level. It can be seen that knowledge will affect the quality of testing process.

After the test, 44 test problems were found by group A and 31 problems by group B. The comparison of test problems with different critical levels between the two test teams is shown in Figure 9. In terms of the total number of testing problems detected, group A is more than group B. So, the proportion of problems detected by using EPOST is higher, indicating that the size of knowledge affected the quality of testing results. In terms of the number of fatal problems and serious problems, the difference between the two groups is far greater than the proportion of total problems, which indicates that knowledge can support the test of finding key problems, and more knowledge support is needed for the test with high difficulty and high requirement.

In order to evaluate the software testing quality, the influence of knowledge on the test quality level can be quantitatively analyzed by calculating the testing quality entropy [36]. Then, we use (2), (3), and (4) to calculate the testing quality of the two testing teams in the best and the worst case respectively.

In the best case, the number of problems found in group A is set to the total number of problems in ATM software. When calculating the entropy, the total number of problems detected by group A is taken as the denominator of distribution probability. The calculation results are shown in Table 5. The testing entropy of group A is *0*, and that of group B is *0.876*. When $S_A > S_B$, $H_A(Q) < H_B(Q)$. The results show

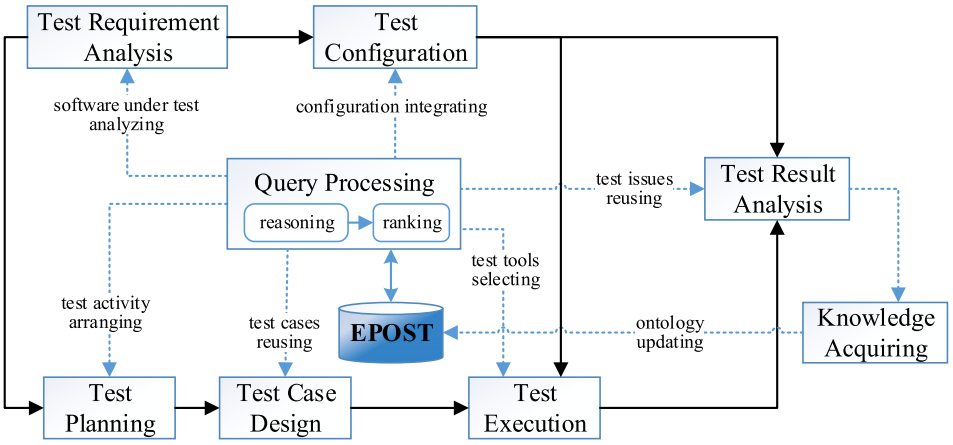

FIGURE 6. Role of EPOST in software testing.

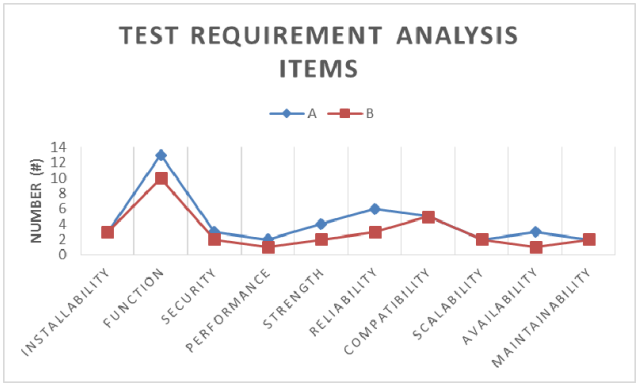

FIGURE 7. Number of test requirement analysis items.

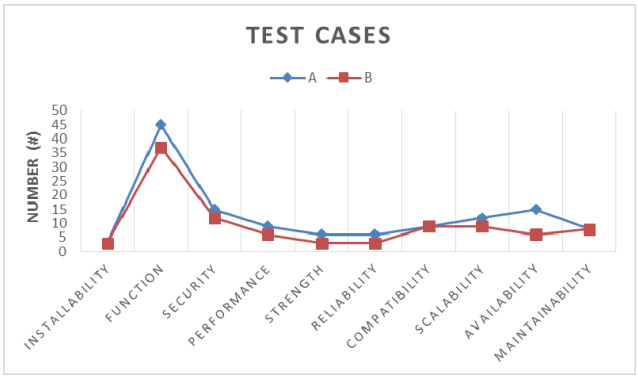

FIGURE 8. Number of test cases.

TABLE 5. Value of Testing Entropy in the Best Case.

Testing Problems	Group		Value of the testing entropy	
	A	B	A	B
Detected Problems	44	31	**0.000**	**0.876**
Undetected Problems	0	13		

that the more knowledge, the lower testing quality entropy, and the more stable testing results.

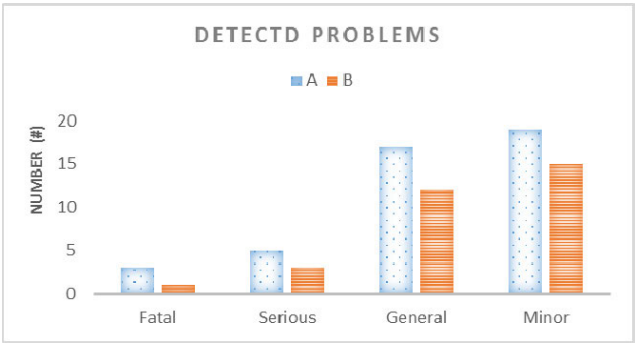

FIGURE 9. Number of detected problems with different critical level.

TABLE 6. Value of Testing Entropy in the Worst Case.

Testing Problems	Group		Value of the testing entropy	
	A	B	A	B
Detected Problems	44	31	**0.869**	**1.000**
Undetected Problems	18	31		

According to the worst case, the number of problems found in group B is set to 50% of the total number of problems in ATM software. When calculating the entropy, the distribution probability takes 2 times of the total number of problems detected by group B as the denominator. The calculation results are shown in Table 6. The testing entropy of group A is *0.869*, and that of group B is *1*. Similarly, when $S_A > S_B$, $H_A(Q) < H_B(Q)$. We can also conclude that the more knowledge, the lower testing quality entropy, and the more stable testing results.

V. DISCUSSION

Ontology evaluation is to evaluate the performance and applicability of an ontology in a specific application domain or environment. It can help users choose the ontology that best meets the application requirements, and it can also

help the ontology constructor to check the effectiveness of the ontology, adjust the irrationality in time, and ensure that the user's requirements are met to the maximum extent. The evaluation method is the core of ontology evaluation, the evaluation metric system is the basis of ontology evaluation, and the evaluation tool is the realization method of ontology evaluation. The organic combination of these three aspects can make the assessment activities more comprehensive and effective.

This study used an ontology evaluation method based on FOCA [20], which was a multiple-criteria evaluation method. This type of evaluation method first determined a set of evaluation metrics based on the factors that affect the quality of the ontology, then designed a criteria measurement method for each evaluation metric, and finally combined the weight of each metric to obtain the final evaluation result. According to the evaluation method of [21], EPOST's ontology quality score was 0.9881, which was close to 1, indicating that the ontology quality was high and the design requirements and purposes of ontology. Although this type of evaluation method had good scalability and operability, and could better reflect the quality of the ontology, it was difficult to develop a comprehensive and reasonable evaluation index system. In terms of the evaluation metrics system, there was currently no unified authoritative standard. Ontology evaluation activities involved the design and calculation of evaluation metrics that would change with the research object, making the evaluation of the ontology more complicated and more research perspectives. The evaluation metrics covered in this study included 8 quality indicators: completeness, competence, adaptability, cohesion, consistency, clarity, computational efficiency, and coherence. There were risks of incomplete coverage of the metric system, such as not considering accuracy, organizational fitness, coupling, etc. To fully evaluate the quality of the ontology, this study used the built-in tool HermiT1.4.3.456 [15] to reason about the consistency of the ontology content and used OOPS! [15], [19] to detect ontology pitfalls.

From the comparative analysis of the above experimental results, it infers that the knowledge quantity can affect the quality of the testing process, and the knowledge quantity is directly related to the testing quality level. The test with higher requirements and more difficulty requires more knowledge support. It is very important to establish a knowledge base of standard, and ontology has a great advantage to support that. Using EPOST domain knowledge as support resources can reduce the influence of human factors on the quality of testing process and testing quality, and make the test task more targeted. The improvement of software quality level by EPOST based testing process is better than the traditional testing process.

VI. CONCLUSION AND FUTURE WORK

To solve the problem of knowledge silo in the software testing process, this article used ontology to organize the knowledge in the entire software testing process, developed a comprehensive, accurate, and structured domain knowledge ontology EPOST. The EPOST ontology covered the concepts and relationships of software testing process information, software testing object information, and software failure information. Extracting concepts and related terms from ISTQB [22], SWEBOK [23], IEEE std.829-2008 [32], and IEEE std.610.12-1990 [33] standards, the concept description in domain knowledge was more accurate and comprehensive. We used a comprehensive ontology construction method [18] to construct ontology. Ontology evaluation used HermiT [15] and OOPS! [15], [19] to perform context validation, and used an improved FOCA method [21] for ontology verification. The evaluation results shown that EPOST has a good domain coverage, was being formally rigorous, implemented also non-taxonomic relations, and achieved the purpose of ontology construction, and met the needs of ontology construction. Ontology application effect evaluation used test cases and test results to qualitatively analyze the test process quality, and quantitatively analyze the quality of test results by calculating testing quality entropy [36]. The experimental results showed that, compared with the traditional software testing process, using EPOST domain knowledge as the support resources to carry out the testing work, testers can have more comprehensive test requirements analysis, better coverage of test cases, higher probability of detecting problems, and better quality of test results.

Domain ontology also has a life cycle [18]. After the construction of ontology, it will be a long-term ontology operation stage and maintenance stage. For future work, we will focus on the continuous improvement of EPOST.

REFERENCES

[1] Z. Sun, Y. Zhang, and Y. Yan, "A Web testing platform based on hybrid automated testing framework," in *Proc. IEEE 4th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Chengdu, China, Dec. 2019, pp. 689–692, doi: 10.1109/IAEAC47372.2019.8997684.

[2] L. Xue-Mei, G. Guochang, L. Yong-Po, and W. Ji, "Research and implementation of knowledge management methods in software testing process," in *Proc. WRI World Congr. Comput. Sci. Inf. Eng.*, Los Angeles, CA, USA, 2009, pp. 739–743, doi: 10.1109/CSIE.2009.360.

[3] 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. Tech.*, vol. 123, pp. 1–23, Mar. 2020, doi: 10.1016/j.infsof.2020.106298.

[4] A. Freitas and R. Vieira, "An ontology for guiding performance testing," in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. (WI), Intell. Agent Technol. (IAT)*, Warsaw, Poland, Aug. 2014, pp. 400–407, doi: 10.1109/WI-IAT.2014.62.

[5] R. Fei-Liang, S. Ji-Kun, S. Bin-Bin, and Z. Jing-Bo, "A review for domain ontology construction from text," *Chin. J. Comput.*, vol. 42, no. 3, pp. 654–676, Mar. 2019, doi: 10.11897/SP.J.1016.2019.00654.

[6] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993, doi: 10.1006/knac.1993.1008.

[7] H. Zhu and Q. Huo, "Developing software testing ontology in UML for a software growth environment of Web-based applications," in *Software Evolution With UML and XML*. Hershey, PA, USA: IGI Global, 2005, pp. 263–295.

[8] E. F. Barbosa, E. Y. Nakagawa, A. C. Riekstin, and J. C. Maldonado, "Ontology-based development of testing related tools," in *Proc. 20th Int. Conf. Softw. Eng. Knowl. Eng.*, San Francisco, CA, USA: SEKE, 2008, pp. 697–702.

[9] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of Web services," in *Proc. IEEE Int. Conf. Web Services*, Beijing, China, Sep. 2008, pp. 465–472, doi: 10.1109/ICWS.2008.111.

[10] L. Cai, W. Tong, Z. Liu, and J. Zhang, "Test case reuse based on ontology," in *Proc. 15th IEEE Pacific Rim Int. Symp. Dependable Comput.*, Shanghai, China, Nov. 2009, pp. 103–108, doi: 10.1109/PRDC.2009.25.

[11] P. G. Sapna and H. Mohanty, "An ontology based approach for test scenario management," in *Proc. 5th Int. Conf. Inf. Intell. Syst. Technol. Manag.*, vol. 141. Berlin, Germany: Springer, 2011, pp. 91–100.

[12] G. Arnicans, D. Romans, and U. Straujums, "Semi-automatic generation of a software testing lightweight ontology from a glossary based on the ONTO6 methodology," *Front. Art. Intell. Appl.*, vol. 249, pp. 263–276, Sep. 2012.

[13] A. Asman and R. M. Srikanth, "A top domain ontology for software testing," M.S. thesis, Jönköping Univ., Stockholm, Sweden, 2015.

[14] E. F. Souza, R. A. Falbo, and N. L. Vijaykumar, "Using ontology patterns for building a reference software testing ontology," in *Proc. 17th IEEE Int. Enterprise Distrib. Object Comput. Conf. Workshops*, Vancouver, BC, Canada, Sep. 2013, pp. 21–30, doi: 10.1109/EDOCW.2013.10.

[15] S. Vasanthapriyan, J. Tian, D. Zhao, S. Xiong, and J. Xiang, "An ontology-based knowledge sharing portal for software testing," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Prague, Czech Republic, Jul. 2017, pp. 472–479, doi: 10.1109/QRS-C.2017.82.

[16] S. Vasanthapriyan, J. Tian, and J. Xiang, "An ontology-based knowledge framework for software testing," in *Proc. Commun. Comput. Inf. Sci.*, 2017, pp. 212–226, doi: 10.1007/978-981-10-6989-5_18.

[17] H. de S. Campos Junior, C. A. de Paiva, R. Braga, M. A. P. Araújo, J. M. N. David, and F. Campos, "Regression tests provenance data in the continuous software engineering context," in *Proc. 2nd Brazilian Symp. Systematic Automated Softw. Test. (SAST)*, 2017, pp. 1–6, doi: 10.1145/3128473.3128483.

[18] Z. Wen-Xiu and Z. Qing-Hua, "Research on construction methods of domain ontology," *Library Inf.*, no. 1, pp. 16–19, Jan. 2011.

[19] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation," *Int. J. Semantic Web Inf. Syst.*, vol. 10, no. 2, pp. 7–34, Apr. 2014.

[20] J. Bandeira, I. Ibert Bittencourt, P. Espinheira, and S. Isotani, "FOCA: A methodology for ontology evaluation," 2016, *arXiv:1612.03353*. [Online]. Available: http://arxiv.org/abs/1612.03353

[21] X. Hu and J. Liu, "Ontology construction and evaluation of UAV FCMS software requirement elicitation considering geographic environment factors," *IEEE Access*, vol. 8, pp. 106165–106182, 2020, doi: 10.1109/ACCESS.2020.2998843.

[22] ISTQB. *International Software Testing Qualifications Board, Standard Glossary of Terms Used in Software Testing, Version 3.2*. Accessed: 2019. [Online]. Available: https://www.istqb.org/

[23] IEEE. *The Institute of Electric and Electronic Engineers: Computer Society, SWEBOK. A Guide to the Software Engineering Body of Knowledge*. Accessed: 2014. [Online]. Available: http://www.computer.org/portal/web/swebok

[24] H. Jian-Ying, Y. Hai-Hua, and L. Chao, "Study of ontology-based software testing knowledge management model," *Comput. Sci.*, vol. 34, no. 10, pp. 281–283, Oct. 2007.

[25] S. Guo, J. Zhang, W. Tong, and Z. Liu, "An application of ontology to test case reuse," in *Proc. Int. Conf. Mech. Sci., Electr. Eng. Comput. (MEC)*, Jilin, China, Aug. 2011, pp. 775–778, doi: 10.1109/MEC.2011.6025579.

[26] X. Li and W. Zhang, "Ontology-based testing platform for reusing," in *Proc. 6th Int. Conf. Internet Comput. Sci. Eng.*, Zhengzhou, Henan, Apr. 2012, pp. 86–89, doi: 10.1109/ICICSE.2012.18.

[27] Z. Ying, "The research and application of software testing method based on domain knowledge," M.S. thesis, Softw. Technol. Inst., Dalian Jiaotong Univ., Dalian, China, 2014.

[28] É. F. de Souza, R. D. A. Falbo, and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," *Inf. Softw. Technol.*, vol. 57, pp. 378–391, Jan. 2015, doi: 10.1016/j.infsof.2014.05.016.

[29] S. Sharma, L. Raja, and D. P. Bhatt, "Role of ontology in software testing," *J. Inf. Optim. Sci.*, vol. 41, no. 2, pp. 641–649, Feb. 2020, doi: 10.1080/02522667.2020.1733196.

[30] H. Li, F. Chen, H. Yang, H. Guo, W. C.-C. Chu, and Y. Yang, "An ontology-based approach for GUI testing," in *Proc. 33rd Annu. IEEE Int. Comput. Softw. Appl. Conf.*, Seattle, WA, USA, Jul. 2009, pp. 632–633, doi: 10.1109/COMPSAC.2009.92.

[31] H. Xian-Yu, "Construction of software testing information domain ontology," *Softw. Guide*, vol. 12, no. 9, pp. 29–31, Sep. 2013.

[32] *IEEE Standard for Software and System Test Documentation—Redline*, IEEE Standard 829-2008, 2008.

[33] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standard 610.12-1990, 1990.

[34] X. Lei, "Research advances in ontology evaluation," *J. Chi. Soc. Sci. Tech. Inf.*, vol. 35, no. 7, pp. 772–784, Jul. 2016.

[35] H. Alrumaih, A. Mirza, and H. Alsalamah, "Domain ontology for requirements classification in requirements engineering context," *IEEE Access*, vol. 8, pp. 89899–89908, 2020, doi: 10.1109/ACCESS.2020.2993838.

[36] C. Yang, "Reliability knowledge-based software testing model and its application," Ph.D. dissertation, Wuhan Univ. Tech., Wuhan, Hubei, China, May 2017.

[37] Y. Yuehua, D. Junping, and P. Yuan, "Ontology-based intelligent information retrieval system," *Ruan Jian Xue Bao/J. Softw.*, vol. 26, no. 7, pp. 1675–1687, Jul. 2015, doi: 10.13328/j.cnki.jos.004622.

[38] L. Xiaoqi, Y. Genxing, C. Lizhi, and Z. Juan, "Ontology description and retrieval of test case based on reuse behaviour," *Comput. App. Softw.*, vol. 28, no. 10, pp. 65–68, Oct. 2011.

ZHE SUN received the B.S. degree in software engineering from Xidian University (XDU), Xi'an, China, in 2009, and the M.S. degree in software engineering from the University of Science and Technology of China (USTC), Hefei, China, in 2012. He is currently an Engineer with the Institute of Computer Application, China Academy of Engineering Physics, Mianyang, China. His current research interests include ontology modeling, software testing, and software reliability.

CHI HU was born in China in 1988. He received the B.S. degree in computer science and technology from the University of Electronic Science and Technology of China (UESTC), in 2010, and the M.S. degree from the College of Information and Software Engineering, UESTC, in 2013. He is currently pursuing the Ph.D. degree with the National University of Defense Technology. He is also an Engineer of the China Academy of Engineering Physics. He has three years of experience in CPS system safety and security evaluation. His research interest includes orientation is software testing and verification.

CHUNLEI LI received the B.S. degree in intelligent monitoring and control from the Nanjing University of Aeronautics and Astronautics, China, and the M.S. degree in 2007. He is currently an Engineer with the Institute of Computer Application, China Academy of Engineering Physics, Mianyang, China. His current research interests include embedded system distributed co-simulation, software security reliability simulation, and testing.

LINBO WU was born in 1983. He is currently a Senior Engineer. His current research interests include software modeling and verification, and simulation verification technology. He was a member of China Computer Federation.