The Use of Ontology in Case Based Reasoning for Reusable Test Case Generation

Rui Li, Shilong Ma
State Key Lab. Of Software Development Environment
School of Computer Science & Engineering, Beihang University
Beijing 100191,China
{lirui, slma}@nlsde.buaa.edu.cn

***Abstract*—As we know, software testing is an important part of software development lifecycle. More than 50% of the whole system development work and total cost were spend on the software testing. And it's estimated that almost 60% of the total test time and cost were spent on the design of test cases. In recent years, the automated testing gradually replaces the traditional manual testing to become an important branch of software testing. In automated testing, the most important thing is to design and generate valid test case automatically. In this paper, In order to reduce the workload of testers on the test case generation, improve the test efficiency and makes test experiences be passed on, we design a common description method for the test case based on ontology; propose a semantic similarity measure method to retrieve the usable test cases from test case library based on WordNet; establish the relationship between the test case and test requirement though rules to modify the retrieved test case; generate the final test case sequence. At last, the practical applicability of the approach is evaluated through an experiment.**

Keywords-software testing; ontology; test case generation; semantic similarity measure

I. INTRODUCTION

As we know, software testing is an important part of software development lifecycle. Research shows that approximately 50 percent of the elapsed time and the total cost were spent on the software testing. In some areas which relate to life and property, such as aerospace, national defense, transportation, nuclear energy and so on, it will take a longer time and costs more [1]. In recent years, the automated testing gradually replaces the traditional manual testing to become an important branch of software testing. In automated testing, the most important thing is to design and generate valid test case automatically. Statistically speaking, it's estimated that almost 60% of the total test time spent on the design of test cases [2]. So, how to design and generate test cases efficiently becomes a hot research topic in the area of automated test.

Test case is a set of conditions or variables which used to determine whether an application, a software system or one of its features is working as it was originally established for it to do [3]. In general, the test case design and generate depends on the tester's intuition and personal experience, and the format of test cases varies from different testers, which directly affect the efficiency of software testing. As we know, software reuse has been thought as a key strategy for reducing development costs and improving quality [4]. Some experts have put the concepts of reuse into the practice of software requirements engineering and design engineering, and have achieved remarkable results. Applying this method to software test and making full use of the past outcomes, accumulated knowledge and experiences to design new test case not only can make up the deficiency of testers' experiences, but also can reduce those redundant works in designing similar test cases, which improves the test efficiency and makes test experiences be passed on.

Case-based reasoning is a method for problem-solving. It imitates human thinking trying to make a decision based on earlier experiences. In other words, Case-based reasoning is an approach which can be used to solve new problems by using or reusing that were used to solve similar problem [5]. That is precisely suitable for the requirements of the test case reuse. On the other hand, although there are a large number of test cases can be reused in the field of software testing, the test process will different due to the different operating system, operating environment, hardware, network conditions, user characteristics, etc. This undoubtedly increases the complexity of the test case which determines the test cases reuse need to take various factors into account, and find out the best solution on the basis of previous experience and test cases. That makes case-based reasoning useful and highly valued.

In this paper, to simplify the way of generate reusable test case for various testing, we design a common description method for the test case based on ontology; propose a a semantic similarity measure method to retrieve the usable test cases from test case library based on WordNet; establish the relationship between the test case and test requirement though rules to modify the retrieved test case; generate the final test case sequence. At last, the practical applicability of the approach is evaluated through an experiment.

The rest of this paper is structured as follows. In Section 2 we outline the related work in this area followed by Section 3 that describes the ontology-based test case library and test task along with its constituent concepts and relations. Section 4 outlines a semantic similarity measure method to retrieve the usable test cases from test case library based on WordNet, and a rule-based method to generate the test case sequence. In Section 5, we give an experiment to evaluate this approach. Finally we discuss our conclusions and future work in Section 6.

II. RELATED WORK

In the 1900s, the ideas that apply the concepts of reuse into the software test cases began to sprout. To the best of our knowledge, Mayrhauser is the first researcher who published relevant paper. In his paper, Mayrhauser [6] proposed a new test case generate method to improve the reuse of test cases by domain analysis and domain modeling, and developed a test case generate tool (DBT) based on domain modeling. After that, much research has been devoted to the reuse of the test case. It can be mainly divided into two aspects: the reusable test cases generation and the reusable test case management. Govind Kulkarni [7] discussed the reusability of test case for web application. Renzuo Xu [8] provide a theoretical model for generating and executing pattern, and make the test cases independent of the software to be tested and reach the testing reuse target. Yongbo Wang [9] proposed an approach of test case generation based on ontology. To describe precisely and accurately test case, Shaojie Gao [10] pointed out an ontology-based method which as a basis for the sharing and reuse of knowledge has been widely used in information science. Luxiao L [11] developed a test case library and discussed the model of testing case managing. To support effective test reuse, Z. L. Shao [12] proposed a software test design model based on the analysis of reusable test assets and their relationships. There are many outstanding researches which focus on how to generate or manage test case, however, few study is chiefly concerned with how to retrieve reusable test case from test case library efficiently. In recent years, CBR is considered as an effective approach to improve this problem. And the related research is just beginning. As we know, the case-based reasoning for test case generation involves following steps: (1) retrieving relevant test cases from the test case library; (2) selecting a set of the most suitable test cases; (3) modifying and evaluating the set of test case in testing process; (4) storing the newly test cases in the test case library as a valuable and reusable resource for the future applications or system. In the whole process, it is a key issue that the suitable test cases description and retrieval. The biggest drawback of CBR is that the case adaptation is not easy, which often need an artificial adjustment. But if using ontology to describe the test case, the test case will have semantic capability, which makes itself easy to be modified. Therefore, in the following section, we build ontology-based test case library, and retrieve the reusable test case from this library by calculating the semantic similarity between test case and test requirement.

III. ONTOLOGY-BASED DESCRIPTION

Actually, ontology is a philosophical concept. Over the past twenty years, the ontology is widely used in artificial intelligence. As a vividly description said, ontology like a smart middleware between people and machine. It can not only makes the communication between people and machine become more smooth, but also helps the machine to understand natural language better that machine can make the appropriate changes based on the changes in natural language. In software engineering, the specification is described in natural language at first. The test requirements and the test cases are no exception. For the machine, there is no link between two different sentences with similar meaning which described by natural language. However, ontology can establish a semantic association between these two different sentences which can be understood by machine. Therefore, we use ontology to build the test case library and model the test requirements.

In this paper, all the work is based on the following two assumptions.

Assumption 1: The same or similar test requirements can use the same or similar test case.

Assumption 2: The same or similar test requirements will be repeated.

A. *Ontology-Based Test Case Library*

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

As ontology is used to describ different areas, as no universal definition for ontology exists. We give the definition of ontology-based test case and test case library as follows:

Definition 1: Test Case Ontology. TCO={TCC, R}. TCC (Concepts) and R (Relations) are two sets which don't intersect with each other.

For convenience, following basic relations of ontology are used in the rest of this paper:

Theorem 1: P(x) relation. $\exists C_1 \in$ TCC, if the type of concept "C_1" is P, then it can be described as "P(C_1)". For example: has(C_1): means that C_1 is exist.

Theorem 2: R(x, y) relation. $\exists C_1, C_2 \in$ TCC, if the type of relationship between concept "C_1" and "C_2" is "R" then the relation can be described as R(C_1, C_2). For example: instance(C_1, C_2): describes the instance relationship between two concepts, it means that concept "C_1" is the instance of "C_2".

Definition 2: **Test Case Library.** TCL= {TCO, Rules}. Rules represent a set of inference rules that we will explain in detail below. In this paper, rules are used to modify the test case.

As we mentioned above, a test case in software testing is a set of input, execution condition and expected results for a specific purpose. And in the ontology-based test case library, the precise definition of the terms can be represented by a series of description logic formulas. Therefore, In order to clearly exhibit the process of reusing test cases, a test case is defined as a 7-tuple: {TID, TP, PR, TE, TI, TO, ER}. The 7-tuple is shown in Table 1.

TABLE I. THE PROPERTIES OF A TEST CASE

Property	Name of Property	Description
TID	Id	Unique identifier of a test case
TP	Test Purpose	The indivisible purpose of the testing
PR	Precondition	The condition need to meet before testing
TE	Test Environment	The environment needed in the testing
TI	Test Input	The input data
TO	Operation	The process of testing
ER	Expected Result	The expected result

Definition 3: Test Case Sequence. The test case sequence is composed of at least one test case. And there is a certain sequence existed between the test cases. It can be defined as follows:

$$TCS: \geq 1 has(TCO) \wedge Pre(tco_i, tco_j) \wedge instance(tco_i, TCO) \wedge instance(tco_j, TCO)$$

For a better description, number restrictions and an R(x, y) relation are used in the formal expression of TCS.

- $\geq n$ R : at least number restrictions, the number of relationship "R" is at least n.
- $\leq n$ R : at most number restrictions, the number of relationship "R" is at most n.
- Pre(x, y): sequence relationship, it means that x is prior to y.

B. *Ontology-Based Test Task*

At the beginning of the software testing, testers must determine the test target. A test target can be represented as a set of test requirements. For example, for the functional coverage testing, each function of the application or system which waits to be tested corresponds to a test requirement. Moreover, testers also need to determine the test environment. Therefore, a test task can be defined as a 2-tuple (TT, TE), and TT refers to test target, while TE means test environment. If using TR to represent the test requirement which cannot be subdivided into some smaller requirements, TT can be defined as follows:

$$TT: \geq 1 has(TR) \wedge Pre(tr_i, tr_j) \wedge instance(tr_i, TR) \wedge instance(tr_j, TR)$$

This definition is similar to the definition of TCS. It also illustrates that at least one test case sequence is required to complete a test target.

Here is a simple example to illustrate the relationship between them which can be seen from Fig.1. Assuming for a specific test target, there is a test requirement set R = {tr_1, tr_2, tr_3, tr_4}, and there exists an order between them, such as $tr_1 \rightarrow tr_2 \rightarrow tr_3 \rightarrow tr_4$.

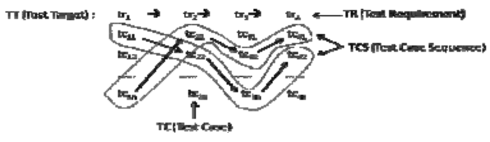


FIGURE I. A SIMPLE EXAMPLE TO ILLUSTRATE THE RELATIONSHIP BETWEEN TEST CASE, TEST CASE SEQUENCE, TEST TARGET AND TEST REQUIREMENT

If we can find test case set {tc_{11}, tc_{12}… tc_{1n}} for tr_1, test case set {tc_{21}, tc_{22}… tc_{2n}} for tr_2, test case set {tc_{31}, tc_{32}… tc_{3n}} for tr_3, and test case set {tc_{41}, tc_{42}… tc_{4n}} for tr_4. In the actual testing, the number of test cases in each test case set for each test requirement is not equal. If the output of tc_{11} meets the precondition of tc_{22}, then we can add them into a same test case sequence. And the rest can be done in the same manner. Finally, the test case sequence can be found for the specific test target.

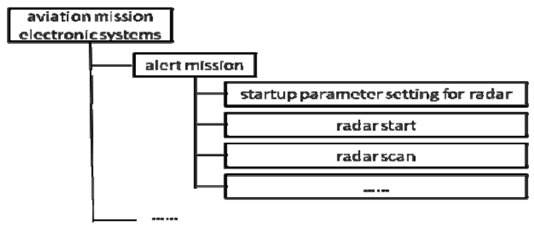
IV. TEST CASE RETRIEVAL AND ADAPTATION

If the test requirement (TR) is regarded as query case, and the test case library is viewed as case base, the reusable test case generation might be considered as searching the cases which has the highest degree of matching with query case from case base, analyzing and rewriting the cases according to the actual conditions. In this section, we will discuss the two steps respectively.

A. *WordNet-Based Test Case Retrieval*

The test case retrieval process is to find the test case which TP match the TR from the test case library. Because the different people have different expression for the same requirement in software engineering, the string comparison method which only judge whether all strings are equal is not applicable at here. In this paper, we introduce an ontology similarity calculation method to get a test case set in which the TP of each test case match the TR semantically in varying degrees. The ontology similarity calculation is based on the WordNet which is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations [13]. Our approach distinguishes the part of speech the word belongs to. Different parts of speech have different weights in the calculation.

As it mentioned above, TP refers to the indivisible purpose of the testing. Each test case can be modeled as a node in ontology. TP which should include one verb and several nouns is a property of the node. Fig.2 shows an example of test purpose for aviation mission electronic systems.



aviation mission electronic systems
alert mission
startup parameter setting for radar
radar start
radar scan
……
……

FIGURE II. AN EXAMPLE OF TEST PURPOSE FOR AVIATION MISSION ELECTRONIC SYSTEMS

The alert function of aviation mission electronic systems has several realization steps. Such as set the startup parameter for radar, start radar, radar scanning, judge the object which is scanned, and so on. As "set the startup parameter for radar" for example, it can be expressed as a verb and a set of nouns. TR is similar to TP. It also can be expressed as a verb and a set of nouns. We calculate the similarity between them according to the WordNet. The working principle of the ontology similarity calculation can be seen from the Fig.3.

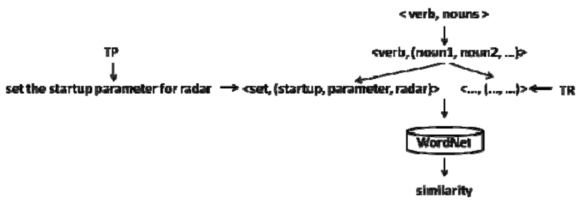

< verb, nouns >
TP
<verb, (noun1, noun2, ...)>
set the startup parameter for radar
<set, (startup, parameter, radar)>
<..., (..., ...)>
TR
WordNet
similarity

FIGURE III. THE WORKING PRINCIPLE OF THE ONTOLOGY SIMILARITY CALCULATION.

There are many methods used to measure ontology semantic similarity on the basis of WordNet [14][15][16][17]. In general, these methods can be divided into the following categories: (1) based on the path length of concept; (2) based on information; (3) based on features; (4) other comprehensive calculation method. Since we need to find as many test cases as possible for consideration in the reusable test case retrieval phase. So, when calculating the similarity of two concepts, we only consider two basic factors: the concept path length and the coincide path length which defined as follows:

Definition 4: Concept Path Length (h_w). The path length from concept to the root.

Definition 5: Coincide Path Length (h). The length of the overlap path from two concept to the root.

Assume that there are two concepts W_i and W_j. The concept path length of W_i is h_{wi}, and the concept path length of W_j is h_{wj}. The length of coincide path between them is h.

If we define the contact ratio (rc):

$$rc = \begin{cases} \dfrac{h}{(h_{wi}+h_{wj})/2} & h_{wi} \neq h_{wj} \\[2ex] \dfrac{h}{h_{wi}} & h_{wi} = h_{wj} \end{cases}$$

Then the similarity between W_i and W_j can be calculated as follows:

$$sim(w_i, w_j) = \frac{e^{rc} - e^{-rc}}{e^{rc} + e^{-rc}}$$

And the similarity between TP and TR can be obtained by the following formula:

$$sim(TP, TR) = \alpha \cdot sim(verb_{TP}, verb_{TR}) + \beta \cdot \underset{(i,j)}{Max}\, sim(noun_{TP}, noun_{TR})$$

In the above formula, α and β are the factor parameters whose value ranges in (0, 1), and $\alpha + \beta = 1; \alpha \geq \beta$.

For the two concepts W_1 and W_2, their location relation in WordNet may have the following situations which can be seen in Fig. 4.

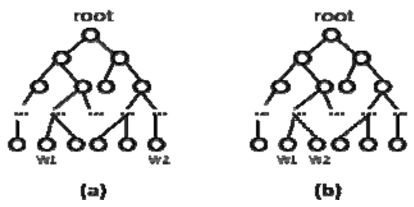


FIGURE IV. THE POSITION RELATION OF W1 AND W2 IN WORDNET.

As can be seen from Fig.4., the location position relation between two concept W_1 and W_2 can be classified into two categories: (1) the coincide path length is 0 (h=0); (2) the coincide path length is greater than 0 (h>0). According to the information theory [18], in the first case, there is almost no similarity between W_1 and W_2. And in the second case, the greater the ratio of the coincide path length in the concept path length, the less the similarity between W_1 and W_2.

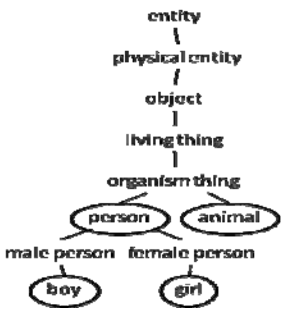


FIGURE V. THE PART OF THE HIERARCHICAL SEMANTIC STRUCTURE IN WORDNET

As {person, animal, boy, girl} for example to prove the validity of the formula which we proposed above. Fig.5 shows the part of the hierarchical semantic structure in WordNet which contains the four nouns. The similarity calculation results are shown in Table 2.

TABLE II. THE SIMILARITY CALCULATION RESULTS OF THE FOUR NOUNS.

(w_1, w_2)	**h**	**h_{w1}**	**h_{w2}**	**rc**	**Sim(w_1,w_2)**
(person, animal)	4	5	5	4/5	0.6640
(person, boy)	5	5	7	5/6	0.6823
(boy, girl)	5	7	7	5/7	0.6134
(boy, animal)	4	7	5	4/7	0.5164

The results shows that the similarity between person and boy is greater than person and animal. And the similarity between boy and animal is the smallest in the results. This is not only consistent with the facts but also consistent with the information theory. Therefore, the formula can be used to retrieve similar test case from the test case library.

B. *Rule-Based Test Case Adaptation*

Generally speaking, due to the test pre-conditions and test environment have changed, the retrieved test cases are difficult to use in testing directly.

In different test environment, the execution command of test case and the environment variables of test case are different. However, system has stable function implement and business logic which cannot change with the environment. Therefore, we need to modify the related information of test cases which we retrieved from the test case library according to the test environment which specified in the test task.

Operating system is an important element in the test environment. As Windows and Linux for example, we give the rewriting rules of test case as follows.

(1) Rewrite the operating system on which the test case is run when the operating system information of test case is different from that of test task.

testTask(?tt)∧hasOS(?x, ?tt)∧testCase(?tc)∧ hasOS(?y, ?tc)∧different(?x, ?y)→transOS(?x, ?tc)

(2) Modify the script execution command.

Is-OS(?x, Windows)∧ script(?file)→execute("cmd"+?systemDrive+?file)

Is-OS(?x, Linux)∧script(?file)→execute("./" +?file)

(3) Modify environment variables.

Is-OS(?x, Windows)∧path(?path)→execute("cmd set"+?path)

Is-OS(?x, Linux)∧path(?path)→execute("export PATH=$PATH:" +?path)

Other changes which caused by the change of test environment can be written to the similar rules.

As we know, if the expected result of a test case meets the precondition of another test case, then the two test cases can be added into a same test case sequence. However, 100% satisfied situation do not occur as often. In this paper, we discuss three situations.

(1) The expected result of a test case has an intersection with the precondition of another test case. Table 3 shows two test case for example.

TABLE III. THE EXPECTED RESULT OF A TEST CASE HAS AN INTERSECTION WITH THE PRECONDITION OF ANOTHER TEST CASE.

Test case 1	**Test case 2**
…	…
…	PR: $P_i \wedge \ldots \wedge P_j \wedge P_{j+1} \wedge \ldots \wedge P_n$
…	…
ER: $P_1 \wedge \ldots \wedge P_{i-1} \wedge P_i \wedge \ldots \wedge P_j$	…

In table 3, the expected result of test case 1 is P1∧…∧Pi-1 ∧Pi∧…∧Pj, while the precondition of test case 2 is Pi∧…∧Pj∧Pj+1∧…∧Pn. They are not equal, but there exists intersection Pi∧…∧Pj. In this situation, we modify the TI of test case 2 by adding the inputs which satisfy the disjoint parts. The rewriting rules described below.

testCase(?x)∧testCase(?y)∧different(?x, ?y) ∧ hasER(?er, ?x)∧hasPR(?pr, ?y) ∧hasTI(?ti, ?y)∧((has (?P, ?er) ∧¬has (?P, ?pr))∨(¬has (?P, ?er) ∧has (?P, ?pr)))→instance(?ins, ?P)∧add (?ins, ?ti)

(2) The expected result of a test case contains the precondition of another test case which can be seen in Fig.6(a).

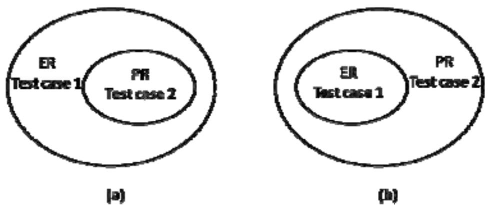


FIGURE VI. THE CONTAIN RELATIONSHIP BETWEEN THE EXPECTED RESULT OF A TEST CASE AND THE PRECONDITION OF ANOTHER TEST CASE

Because ER = PR∪(ER-PR), we can write a new test case for the part (ER-PR) according to the following rule. The new test case is derived from test case 2. So, they have the same TO, but different PR, TI and ER.

testCase(?x)∧testCase(?y)∧different(?x, ?y) ∧ hasER(?er, ?x)∧hasPR(?pr, ?y) ∧hasTI(?ti, ?y) ∧ hasTO(?to, ?y) ∧hasER(?r, ?y)∧contain (?er, ?pr) →newTestCase(?z)∧addPR(?er-?pr, ?z) ∧instance(?ins, ?er-?pr)∧hasTI(?in, ?z)∧add (?ins, ?in) ∧addTO(?to, ?z) ∧ addER(¬?r, ?z)

(3) The expected result of a test case is contained in the precondition of another test case. As shown in Fig.6(b).

In this situation, the expected result of a test case must meet the precondition of another test case. So, there is no need to modify the test case 2.

Actually, at most of the time, the relationship between the expected result of a test case and the precondition of another test case is the combination of the three situations we mentioned above.

V. EXPERIMENT

Our initial implement uses a test case library consists of 200 test case and 23 rules. We use OWL to describe the test case and use SWRL to describe the rewriting rules. And we test our experiment on a machine has a 2.4GHz CPU and 1G RAM.

In our experiment, we decompose a test task into 6 test requirements. Table.4 shows the results.

- The serial number of test requirement TR(column labeled No.)
- Number of the retrieved test cases(column labeled Test Cases)
- Number of the retrieved test cases of which Sim(TP, TR) is greater than 0.5(column labeled Sim(TP, TR)≥0.5)

- The proportion of valid test cases(column Percentage)
- The total time spend on test case retrieve in seconds (column labeled Time)

TABLE IV. RESULTS OF THE EXPERIMENT.

No.	Test Cases	Sim(TP, TR) $\geq$ 0.5	Percentage	Time
1	16	10	0.625	4.01
2	14	8	0.571	3.87
3	10	7	0.7	3.73
4	10	6	0.6	3.73
5	15	9	0.6	3.88
6	23	11	0.478	4.13

The number of generated test case sequence: 4

The results of experiment illustrate that (1) greatly reduce the time which used to generate test cases; (2) the ontology-based similarity calculation approach for test case Can guarantee a certain recall ratio; (3) effectively establish the relationship between different test cases through semantic and generate the test case sequence; (4) although the result of this experiment is not significant, it will improved with the new test case added into the test case library. It can be said that the reusable test case generation method we proposed in this paper is feasible in practical applications.

VI. CONCLUSION

This paper discusses how to generate the reusable test case from test case library based on ontology. Here the knowledge base is test case library which represent what we can learn from a test case, and define the rules that used to adaptation the retrieved test case. We have argued that, although each test case is complexity and seems independent with the other test case, the semantic similarity between them still can be measured. The results of experiment shows that the approach we proposed in this paper can greatly shortens the time spent on the reusable test case generation, reduces the workload of tester, improve the efficiency of the test, and makes test experiences be passed on.

In this paper, we have also proposed an approach for calculating the ontology semantic similarity between test case and test requirement based on the WordNet. However, this approach is too coarse at this stage and we will take more effort to elaborate it in the future.

The experiment we introduce in section 5 is quite simple; it did not consider the coverage of the generated test case sequence, and the test case sequence generation process still requires manual intervention, it unable to achieve fully automatic. In the following work we will evaluate and improve our approach by more valuable case. And to make our work more persuasive, the credibility measure model is necessary. These are within our future work.

ACKNOWLEDGMENT

We would like to thank students of the same laboratory for their comments on the paper and also for proofreading the text. This work is partially supported by National Science and Technology Support Program (2011BAH14B04).

REFERENCES

[1] DAVID S., The economics of software quality assurance[J]. National Computer Conference, 1976

[2] Glenford J Myers, Tom Badgett. The art of software testing[M]. 2nd ed.

[3] http://en.wikipedia.org/wiki/Test_case

[4] Frakes W. Systematic Software Reuse: A Paradigm Shift. In Proceedings of Third international Conference on Software Reuse: Advances in Software Reuse. Los Alamitos, California: IEEE Computer Society Press, 1994

[5] Avramenko, Yuri, Kraslawski, Andrzej. Case Based Design: Applications in Process Engineering. Studies in Computational Intelligence, Vol.87. 2008.Springer

[6] Mayrhauser A v, Mraz R T, Walls J, et al. Domain Based Testing: Increasing Test Case Reuse. Proceedings of ICCS'94: Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computer & Processors, Washington, DC, USA: IEEE Computer Society, 1994. 484-491

[7] Kulkarni. Reusable Test Cases How Can It Facilitate Web Testing. Proceedings of 3rd Annual International Software Testing 2001 Conference in India,2001

[8] XU Renzuo, CHEN Bin, CHEN Bo, WU Minquan, XIONG Zhongwei. Investigation on the pattern for Construction of Reusable Test Cases in Object-Oriented Software. Journal of wuhan university. 2003, 49(005): 592-596

[9] Yongbo Wang, Ontology-Based Test Case Generation for Testing Web Service. Autonomous Decentralized Systems, 2007. ISADS '07. Eighth International Symposium.

[10] Shaojie Guo, Weiqin Tong, Juan Zhang, Zongheng Liu. An Application of Ontology to Test Case Reuse.2011 International Conference on Mechatronic Science, Electric Engineering and Computer. August 19-22, 2011, Jilin, China

[11] LU Xiao-Li, GE Wei, CHEN Xin-Li, HAO Ke-Gang. Designing a test case library system of supporting sharing and reusing. Journal of computer science. 33(5): 290-291, 2006

[12] Z.L.Shao, X.Y.Bai, and C.C.Zhao. Research and implementation of a reuse-oriented test design model. Journal of mini-micro systems. 27(11):2150-2155, 2006

[13] WordNet. Princeton University. 2010. http://wordnet.princeton.edu/

[14] A. Budanitsky and G. Hirst, Semantic Distance in WordNet: An Experimental, Application-Oriented Evaluation of Five Measures. Proc. Workshop WordNet and Other Lexical Resources, Second Meeting of the North Am. Chapter of the Assoc. for Computational Linguistics, 2001.

[15] A. Budanitsky , G. Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. Volume 32. Computational Linguistics, 2006.

[16] Nuno Seco, Tony Veale, Jer Hayes. An intrinsic information content metric for semantic similarity in WordNet. In Proceedings of ECAI'2004, the 16th European Conference on Artificial Intelligence., Valencia, Spain, 2004.

[17] M. McHale, A Comparison of WordNet and Roget's Taxonomy for Measuring Semantic Similarity. Proc. COLING/ACL Workshop Usage of WordNet in Natural Language Proceeding Systems, 1998.

[18] WIEMER-HASTINGS, P. Adding syntactic information to Isa. In Proceedings of the 22nd Annual Conference Cognitive Science Society. 989-993. 2000.