Journal of Theoretical and Applied Information Technology
30th June 2017. Vol.95. No 12
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE REQUIREMENT REUSE MODEL BASED ON LEVENSHTEIN DISTANCES

[1]WONG PO HUI, [2,*]WAN MOHD NAZMEE WAN ZAINON

School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
[1]wphui.ucom12@student.usm.my, [2]nazmee@usm.my

ABSTRACT

Software reuse has always been one of the popular topic in software engineering community. It refers to the development of software by reusing components previous software development. Reusing components during the development of software reduces time and cost which could also enhance reliability and quality of the concept of reuse. In this paper, a software reuse model is established where it shows the overall process of retrieving the relevant use cases. A database is built and a prototype is developed based on the model. Levenshtein Distance algorithm is applied in computing the similarity score. A list of relevant use cases is displayed as a result. Evaluation criteria such as recall and precision are used to evaluate the result. Based on the results, the enhancement of the software reuse model has been proposed

Keyword- Software Reuse, Use Case Diagram, Software Requirements

1. INTRODUCTION

Software reuse concept was first introduced by Mcllroy at NATO Software Engineering Conference in 1968 [2]. Mcllroy had proposed of building a complex system by reusing the building blocks available in a library of reusable software components. Recently, the concept of software reuse had gained a lot of attention. This is due to the high competition existed among software developers where they need to figure out ways to reduce the cost for the production of software in order to achieve competitive advantage in software market [4]. In addition, software development process has become more complex [5]. This is because software that needs to be developed is complex and huge to satisfy users need by fulfilling their requirements. Therefore, software reuse is believed as one of the solutions to cope with the situations mentioned above.

Software reuse promotes countless benefits. However, it had also caused failure in the development of software. There are some barriers identified such as organizational barriers, economic barriers, administrative impediments, political impediments and psychological impediments [8]. Therefore, some developers feel reluctant to apply the concept of reuse in the development of software through the software reuse.

There are some guidelines proposed by some researchers in reusing the software in order to prevent the failures. One of these guidelines include creating and maintaining a central library where all new components developed from scratch and previously developed projects are available in the library. Code review should also be used as reference to determine if the reusability is feasible or not. Analysis on cost reduction should also be carried out to confirm that the cost could be reduced by reusing the software [8].

Failures in software reuse could also be reduced by reusing the early-stage artefacts. Reusing early-stage software artefacts can actually maximize the benefits gained from software reuse where those later-stage software artefacts that are related to early stage artefacts could also be reused [10].

Basically, in software reuse, all the reusable software components are stored in repository or library. The size of software library would increase as more and more software components are stored [11]. This could increase complexity in the process of searching for the most suitable reusable software components which might lead to the decrease of the benefits in reducing the complexity of software development process.

Besides, retrieval of reusable software components is also an essential process in software reuse. The failure to retrieve the most suitable software components might cause the development of software to fail. This might happen when the software components retrieved and the desired software components are totally different or unrelated. So, in other words, the application of appropriate retrieval technique is important during the process of retrieving reusable software artefacts.

The focus of this paper is the reuse of software requirement specification in terms of use case which act as early-stage software artefacts in the development process. The use of early-stage software artefacts would affect the development life cycle where it acts as a support to the development with software by reusing software components. Requirement engineering process can also be improved.

The main concern of this paper would be retrieval of the use cases where the related use cases would be retrieved based on the keywords and the domain as requested by users. All the use cases would be stored as table form.

A database would be developed where software requirement specification from different domain which are in the form of use cases would be stored after they are collected. Retrieval techniques proposed by other researchers would be analysed. Evaluation would be performed where comparison would be carried out between the retrieved software requirements and the query from users.

2. RELATED WORKS

Software reuse approaches can be divided into three categories such as component-based software reuse, domain engineering & software product and architecture based software reuse [11].

In component-based reuse, a library or repository would be built where components previously developed would be stored in it. Searching for a best match for the components to be reused in the library is the first and most important step in software reuse. Therefore, a searching mechanism would also be developed in order to find a suitable component to be reused.

Domain engineering involves identification of common features of the system developed previously where a new system would be constructed with the reuse of those features. There are two stages included in domain engineering which are domain analysis and domain implementation. In domain analysis, related systems would be studied and the universalities of system features would be the identified. Domain implementation involves development of reusable features based on the universalities of the domain and the features developed would be further used to construct a new system.

A software system's architecture consists of software components, relationships among the components and external properties. In architecture-based reuse, components such as external properties and relationships are reused in the development of new system.

2.1 Software Requirement Specification (SRS)

Software requirement specification (SRS) is an official statement that contains all the functions of a system [4]. In other words, it shows what a system should do to fulfil the needs of users. It is often used as a reference to design a system through the revision of the requirements available in SRS. It also serves as consensus between users and developers or organization about the requirement of users on the system and effort or ability of developers in fulfilling these requirements. Architectural decisions such as decision on system's structure could be made based on the information available in SRS. These decisions could actually cause a tremendous effect on the quality of the system. In addition, other professionals such as Software Quality Assurance (SQA) team or usability experts are also need this document to ensure that the system works as required. So, it can be concluded that the success of a software rely on the level or standard that it manages to reach in fulfilling the requirement specification. SRS could also be described as a detailed description about the functional and non-functional requirements.

Functional requirement describes functions of a system [14]. It plays a vital role as it defines and describes how system works [26]. Functional requirements should always be completed where all the functions of system as requested by users should be included and defined. Besides, it should also be consistent where there is no contradiction occur in the definition of the requirement.

Basically, the definition of non-functional requirements are related to the terms such as performance, constraints, quality, property or characteristic and attribute [14]. Thus, non-functional requirements involves all the demand on the solution of software. Non-functional requirements are often being not concerned as compared to functional requirements as development of system focus on functionality of system only. However, awareness of the importance of non-functional requirements had been increased recently in developing a software that satisfied by users. This is because the failure to meet a non-functional requirement would cause the whole system to be unusable. Non-functional requirement might affect structures and behaviours of software design as some of the non-functional requirements are part of functional entities while others would be the constraints of design.

2.2 Use Case

In Unified Modelling Language (UML) specification, use case is defined as "the specification of a sequence of actions, including variants that a system (or a subsystem) can perform, interacting with actors of the system" [14]. In other words, it can be defined as interactions among actors and system under consideration. Actors could be either a person or groups of people.

There are few types of relationships available in structuring use cases. Those include generalization, include and extends. Include relationship refers to the behaviour explained in the sub use case is included in base use case. Extends relationship mentions about the conditions that must be fulfilled if there is any extensions occur. Generalization relationship shows child use case had every attributes and behaviours that are defined in parent use case.

Basically, it is one of the most commonly used components during the requirement analysis stage in software development life cycle where it plays vital role in collecting and gathering the requirements of system. Reusing use case diagram could actually help developers in modelling use case diagram within a short period of time.

2.3 Software Retrieval

There are four activities involved in software reuse such as requesting the software components through the query, retrieving the most suitable and similar component based on the query, making changes on the retrieved components and integration of the new system into the library [14]. Retrieving components of software is an essential step in the process of software reuse. This is because it shows how much return or gain is obtained through reuse [12].

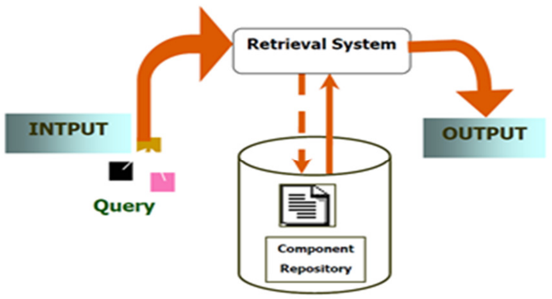


Fig. 1: Software Component Retrieval System

Reusable software components are stored in repository and component library [12]. As more and more software components are kept in the repository, the repository size increases which directly cause an increase in retrieval time. This might increase the development time of software. Figure below shows the software retrieval system proposed.

During the retrieval of software components, the similarity between the components available in the repository and the query that described the required components based to be determine. A similarity metric, in other words comparison functions need to be developed so that users are able to retrieve the desired components.

The software component retrieval is evaluated in order to measure efficiency and effectiveness of the process of retrieval [4]. There are few metrics involved in measuring the process of evaluation such as recall, precision and harmonic mean. Recall refers to the portions of retrieved and related documents to all the relevant documents available in the repository. Precision is defined as the portion of the retrieved and related documents to all the documents that had been retrieved. Harmonic mean is the combination of recall and precision.

Akadej, Nakornthip & Pizzanu [4] had done a research in retrieving software requirements by using use case terms. There are three main processes during the retrieval of use case which include storage, retrieval and evaluation. Storage process involved the collection use case from example and then transformed into indices and weighted value. In retrieval process, user would generate weighted value and queries which then transformed into indices. Use case would be then retrieved from the repository based on the similarity as compared with query generated by user. Evaluation of retrieved use case would be carried out so as to determine the relevance of retrieved use case to user's query.

Theories of information storage and retrieval are applied which include team weighting system, automatic indexing and similarity computation. Term weighting system involves assigning weight to each team which reflects the importance for identification of content. Automatic indexing includes the task of assigning index to components stored in repository. Similarity computation is a process of determining effective approach in retrieving the use case was carried out by comparing the approach of retrieving use case by use case keywords and retrieving use case by use case structure. The experiment result shows the approach of retrieving use case by use case structure is more effective than the approach of retrieving use case by keywords.

3. SOFTWARE REUSE MODEL

Fig.2 shows the software reuse model that has been proposed. From this model, it shows clearly that all the details that had been filled in by users in the form prepared would be store into a table. Keyword get from users would be compared with all keywords available in the database. If similar keyword is found then those use cases with similar keywords would be retrieved. However, similar keyword is unavailable, use cases with same domain would be retrieved.

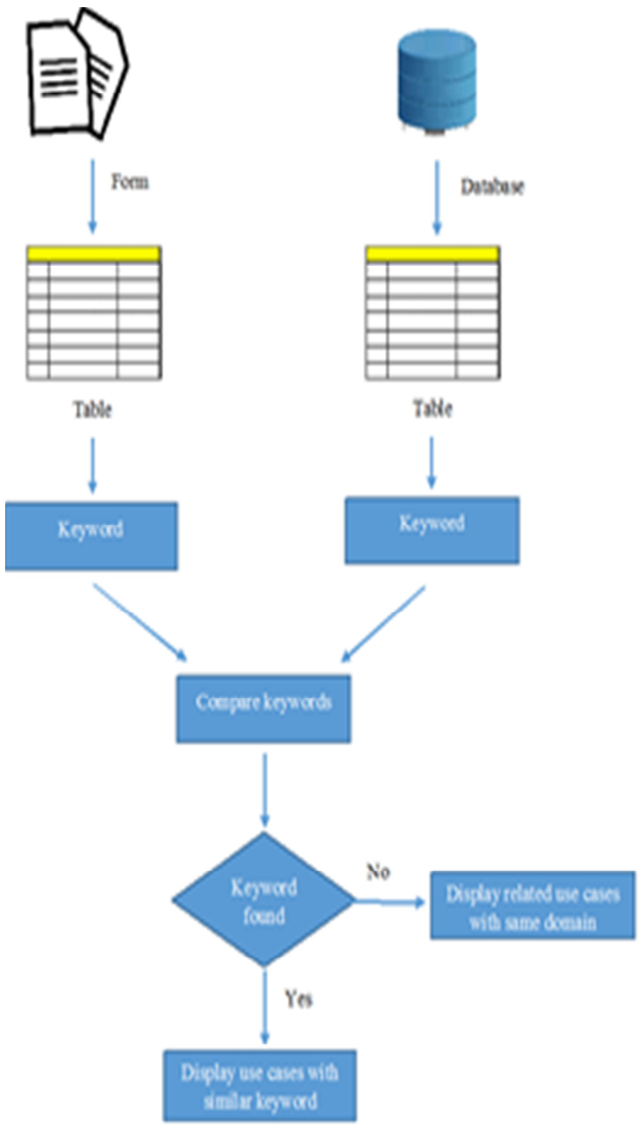

Form
Database
Table
Table
Keyword
Keyword
Compare keywords
Keyword found
No
Display related use cases with same domain
Yes
Display use cases with similar keyword

Fig 2: Software Reuse Model Proposed

A. Evaluation Criteria

There are few evaluation criteria that would be applied in measuring the performance of the retrieval where ratio between the retrieved use case and the desired use case would be identified. The evaluation criteria applied include recall and precision

Recall refers to the percentage of the relevant software components that had been retrieved. The formula of recall is shown as below:

$$Recall = \frac{number\ of\ relevant\ record\ retrieved}{total\ number\ of\ relevant\ record\ in\ database}$$

Precision refers to the percentage of software components that had been retrieved is relevant. The formula of precision is shown as below:

$$Precision = \frac{number\ of\ relevant\ record\ retrieved}{total\ number\ of\ relevant\ and\ irrelevant\ record\ retrieved}$$

B. Data Collection

Experimental data is required to be collected so as to evaluate the effectiveness of the technique selected. Data collected would be the use case diagram. Those use case diagrams stored in database would be in table form. All those use case diagrams would be collected from some academic projects done previously and examples available from text books. Those use case stored in the repository would be used as cases for retrieval.

C. Similarity Computation

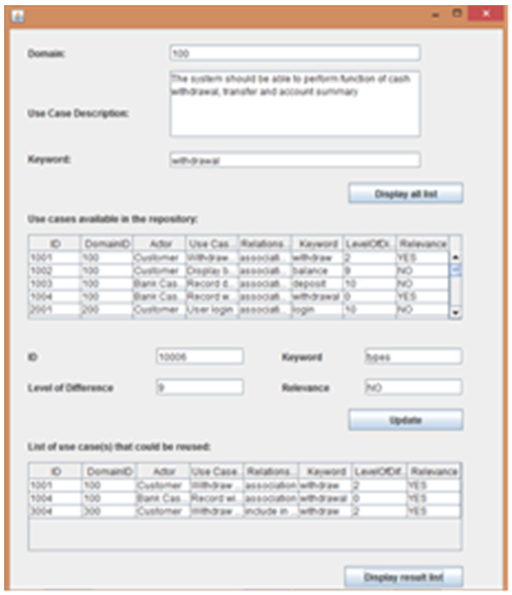
Users would be requested to enter the keyword. Those keywords would be compared with the keyword stored in the repository. So, similarity score would be computed in the process of making the comparison between the keywords. An algorithm named Levenshtein distance would be applied in the process of similarity computation. The shorter the distance, the lower the difference between the keywords. In other words, the level of similarity between both keywords is higher.

This algorithm measures the similarity between two strings which include the source string (s) and the target string (t) and shows the number of character that needs to be edited so that both words would be end up as the same words [16]. Levenshtein distance is usually applied in DNA analysis and detection of plagiarism. So, in this research, keyword entered by user would be the target string whereas the keywords stored in the repository would be the source string. The shorter the distance, the lower the level of difference between the keywords. In other words, both keywords are actually quite similar with each other

4. EVALUATION & RESULTS

Testing is performed after the process of implementation. During testing process, a test document would be prepared where test cases would be listed out. An analysis of determining whether the retrieved use cases fulfilled the requirement from users would then be performed.

The Fig. 3a shows the actual result of test case TC_1. Based on the figure, it shows that the keyword “withdrawal” is available in the repository. So, 1 use case with the same keyword is listed. Keyword “withdrawal” is displayed as well since the level of difference is less than half of the length of “withdrawal”.


Domain: 100

Use Case Description: The system should be able to perform function of cash withdrawal, transfer and account summary

Keyword: withdrawal

Use cases available in the repository:

ID	DomainID	Actor	Use Cas...	Relations...	Keyword	LevelOfDi...	Relevance
1001	100	Customer	Withdraw...	associati...	withdraw	2	YES
1002	100	Customer	Display b...	associati...	balance	9	NO
1003	100	Bank Cas...	Record d...	associati...	deposit	10	NO
1004	100	Bank Cas...	Record w...	associati...	withdrawal	0	YES
2001	200	Customer	User login	associati...	login	10	NO

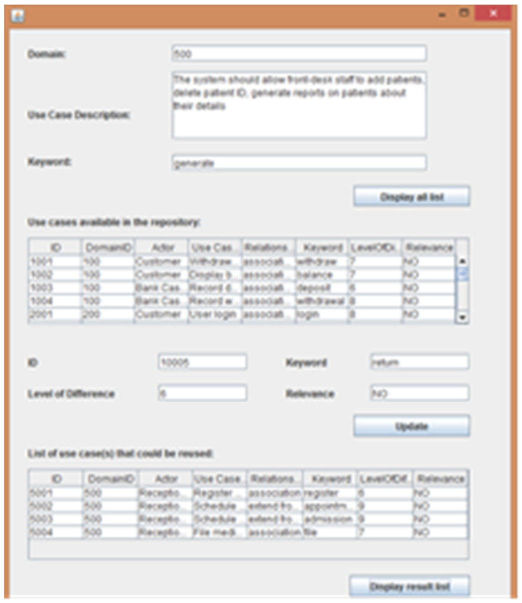
ID: 10006 Keyword: types

Level of Difference: 9 Relevance: NO

List of use case(s) that could be reused:

ID	DomainID	Actor	Use Case...	Relations...	Keyword	LevelOfDif...	Relevance
1001	100	Customer	Withdraw...	association	withdraw	2	YES
1004	100	Bank Cas...	Record wi...	association	withdrawal	0	YES
3004	300	Customer	Withdraw...	include in...	withdraw	2	YES

(a)


Domain: 500

Use Case Description: The system should allow front-desk staff to add patients, delete patient ID, generate reports on patients about their details

Keyword: generate

Use cases available in the repository:

ID	DomainID	Actor	Use Cas...	Relations...	Keyword	LevelOfDi...	Relevance
1001	100	Customer	Withdraw...	associati...	withdraw	7	NO
1002	100	Customer	Display b...	associati...	balance	7	NO
1003	100	Bank Cas...	Record d...	associati...	deposit	6	NO
1004	100	Bank Cas...	Record w...	associati...	withdrawal	8	NO
2001	200	Customer	User login	associati...	login	8	NO

ID: 10005 Keyword: return

Level of Difference: 6 Relevance: NO

List of use case(s) that could be reused:

ID	DomainID	Actor	Use Case...	Relations...	Keyword	LevelOfDif...	Relevance
5001	500	Receptio...	Register...	association	register	6	NO
5002	500	Receptio...	Schedule...	extend fro...	appointm...	9	NO
5003	500	Receptio...	Schedule...	extend fro...	admission	9	NO
5004	500	Receptio...	File medi...	association	file	7	NO

(b)

Fig. 3: Prototype showing result of test case TC_1 & TC_2

The Fig. 3b above shows the actual result of test case TC_2. The figure shows that all the use cases of the related domain is listed. This is because there is no keyword “generate” available in the database. So, all the use cases with related domain displayed act as reference for users in constructing use case diagram.

The actual result and their respective recall and precision are tabulated as Table 2.Based on the result shown in the table, most of the test cases showing 0 for both recall and precision. This might show that the method proposed in retrieving the relevant use cases are improper. However, this is actually depend on number of the use cases available in the repository. The result would be different when number of use cases stored in the repository increases as there are more use cases could be selected and retrieved based on the requirements from user.

Besides, test cases showing 0 is also caused by the keyword entered by user and the keyword stored in the repository. When keyword entered by user is mostly matched with the keyword in the repository or there are countless of keywords are available in the database, then most of the relevant use cases could be retrieved. Thus, the result would be different as compared from the current result obtained.

Users are free to enter any keyword they want. So, there might be cases where the keyword entered by user is actually having the same meaning as some keywords stored in the repository. Those use cases with different keyword but having the similar meaning with the keyword entered by would not be retrieved as result. In other words, those use cases might be left out. This situation might also affect the result obtained.

In addition, the result is also depend on the definition of the relevant record and irrelevant record. In this research, those use cases with same keyword that belong to the same domain or use cases with different keyword but belong to the same domain are defined as relevant records. Other than these, all the other use cases are defined as irrelevant records. Therefore, even though the use cases having the similar keyword as requested by users but belong to different domain, they are still be considered as irrelevant use cases instead of being treated as relevant use cases.

the similarity between the keywords stored in the database and the keyword obtained from user. Test cases has been set and evaluation criteria had been used to measure the system's ability in retrieving the relevant use cases. However, the result obtained is not as accepted as there are actually few factors affect the result which include the keywords available in the database, number of use cases available in the database and definition of relevant and irrelevant records.

Table 2: Result obtained based on recall and precision

Test Case No.		**Actual Result**		**Recall**	**Precision**
	relevant records retrieved	***irrelevant records retrieved***	***relevant records in the database***		
TC_1	2	1	4	0.5	0.67
TC_2	0	0	5	0	0
TC_3	0	1	4	0	0
TC_4	0	2	7	0	0
TC_5	0	1	7	0	0
TC_6	1	0	3	0.3	1.00

5. CONCLUSION

Software reuse has been widely practiced in software engineering community. Therefore, the concept of software reuse would also be applied in developing the requirement specification model which is in the form of database system. Use cases would be collected and stored into database system which act as repository. A retrieval technique would be applied so that the required software components could be retrieved successfully from the database.

The similarity between the retrieved requirements and the desired requirements would be determined to ensure that the correct and proper requirements are retrieved.

So, in this research, a database is built to store use cases which belong to different types of domain which had been collected. Each use case is indicated a keyword so as to ease the search of the relevant use cases as required by users. Levenshtein Distance algorithm has been applied in computing whenever different keyword having the same meaning is obtained. Number of types of domain could also be increased as well so that user requirements would be able to be fulfilled.

In short, use case acts as description of the system where it provides direction for developers to understand users and satisfy them. So, through the reuse of use cases, the failure of the software would be less likely to occur. Instead, the quality of the software system might be improved. The duration of developing a system could also be shortened.

ACKNOWLEDGMENT

This work was supported in part by Universiti Sains Malaysia and Ministry of Higher Education, Malaysia. FRGS Grant (FRGS: 203/PKOMP/6711533).

REFERENCES

[1] Prieto-Diaz, R.: Status Report: Software Reusability. IEEE. 10: 61-66 (1993).

[2] Smolarova, M., Navrat, P.: Software Reuse: Principles, Patterns, and Prospects. Journal of Computing and Information Technology, 5, 1, pp. 33-48 (1997).

[3] Hislop, G.W.: Evaluating a Software Reuse Tool. Assessment of Quality Software Development Tools, Proceedings Third Symposium on. IEEE. pp. 184-190 (1994).

[4] Akadej Udomchaiporn, Nakornthip Prompoon, Pizzanu Kanongchaiyos. (2006). Software Requirements Retrieval using Use Case Terms and Structure Similarity Computation. *Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific*. IEEE. 113-120

[5] Udomchaiporn, A., Prompoon, N., Kanongchaiyos, P.,: Software Requirements Retrieval using Use Case Terms and Structure Similarity Computation., APSEC 2006. 13th Asia Pacific Software Engineering Conference. IEEE. pp. 113-120 (2006).

[6] Barros, F.: Increasing Software Quality through Design Reuse. 2010 Seventh International Conference on the Quality of Information and Communications Technology. IEEE Computer Society Press. 236-241 (2010).

[7] Keswani, R., Joshi, S., Jatain, A.: Software Reuse in Practice. 2014 Fourth International Conference on Advanced Computing & Communication Technologies. pp. 159-162 (1994).

[8] Poulin, J.S.: Measuring Software Reusability. Software Reuse: Advances in Software Reusability, Proceedings Third International Conference on. IEEE. pp. 126-138 (2014).

[9] Leach, R.J.: Software Reuse: Methods, Models and Costs. Ronald J Leach. McGraw-Hill New York (1997).

[10] Salami, H.O., Ahmed, M.: A Framework for Reuse of Multi-view UML Artifacts. The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3. 156-162 (2013).

[11] Krueger, C.W.: Software Reuse. *ACM Computing Surveys (CSUR)*, 24: pp 131-183 (1992).

[12] Salami, H.O., Ahmed, M.: UML Artifacts Reuse: State of the Art. International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3. 115-122 (2014).

[13] Adamu, A., Zainon, W.M.N.W., Salami, H.O.: Pre-filtering Repository Models, The 9th Malaysian Software Engineering Conference (MySec2015)., pp. 200-205 (2015)

[14] Robinson, W.N., Woo, H.G.: Finding Reusable UML Sequence Diagrams Automatically. Software, IEEE. IEEE. 21(5): 60-67 (2004).

[15] Somerville, I.: (2006). Software Engineering 8th edition. China Machine Press.

[16] Srisura, B., Daengdeg, J.: Retrieving use case diagram with case-based reasoning approach. Journal of Theoretical and Applied Information Technology. 19(2): p. 68-78 (2010).

[17] Haldar, R., Mukhopadhyay, D.: Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. Computer Research Repository abs:1106-4098, 2011.