Barishal University Journal Part 1, 5(1&2): 179-193 (2018) ISSN 2411-247X

AN APPROACH FOR MEASURING SIMILARITY OF UML CLASS DIAGRAMS

Md. Samsuddoha* and Rahat Hossain Faisal

Department of Computer Science and Engineering, University of Barishal, Barishal-8200, Bangladesh.

Abstract

Assessment of software similarity is one of the ideal approaches to utilize experiences of effectively developed software for different purposes. Experiences obtained from previous projects can help software industries to deliver software project in a short period of time to minimize the cost and time. Similarity measurement can be one of the solutions to reuse the previous developed technique, code and different methods. Unified Modeling Language (UML) is used widely in software engineering where class diagrams are the top notation to present the core structure of any software system. In this paper, a UML class diagram measuring approach is proposed based on the systematic integration of structure based and component based matching that shows the similarities among the UML class diagrams. An experimental analysis has been conducted for evaluating the applicability of the proposed approach. The experiment justifies the approach that leads to the correct measurement. The experimental analysis was performed using seven different real life software projects by running a developed framework. The analysis shows that the precision, recall and F-measure of the tool are 0.83, 1 and 0.91 respectively which concludes that the proposed approach performs well.

Keywords: Similarity Metric, UML Class Diagram, XML, Similarity Measurement

Introduction

In today's fast changing business environment, software industries attempt to rapidly develop their products so that they can speed up the delivery of their latest innovations to customers. This makes software development more challenging as software developers need to design, implement, and test complex software systems as early as possible. As a

*Corresponding author's e-mail: msamsuddoha@bu.ac.bd

result, software companies are in search of some solutions that can help to deliver good quality and error free software in right time.

Reuse of previous developed model can help to achieve the goal. In this purpose, searching of similar projects those were developed earlier can help to find out proper model. Models in software development allow engineers to downscale the complexity of the software systems. In early stage of software development model provide great reuse potential (Ahmed et al., 2011). During the development of specification of any software project, there are a set of UML diagrams being developed those describe its structural, behavioral and functional perspectives. Class Diagrams represent the structural perspective of a project. In this paper, we focused on UML class diagram to measure the similarity among different software projects.

Overtime, industries are growing with large collections of model. These model represent different development concerns. Furthermore, these models are considered as a main source of knowledge which is retrieved from the minds of stakeholders. This knowledge is reused each and every time when a new software is created. Even, when comparing software systems, we usually find 60% to 70% of software products are functionally common (Tracz et al., 1998). From this fact, we can say that is very effective to use the previous knowledge during development of any new software product which will help to reduce the cost and time. Therefore, it is of paramount importance to have a systematic way to access and reuse existing software models in an effective way. One way of using these model is to measure the similarity with the existing. Another way is to have an efficient repository along with efficient retrieval mechanism. In all cases, similarity measurement is a fundamental operation.

Assessment of similarity is the task of identifying semantic correspondences between elements of two models (Chechik and Sabetzadeh, 2012).This task is error-prone due to the fact that these models represent similar functionalities as the model are developed independently by different developers and also time consuming. Therefore, their similarity and differences must be accurately quantified to find the appropriate match. Over the time, different metrics as well as different matching algorithms have been proposed in the literature to identify the similarity and the differences of the models to be matched, especially for UML diagrams (Alanen et al., 2003; Xing et. al, 2005; Walkinshaw et al., 2009 and Salami et al., 2012).

In this research, a method to measure similarity between Class Diagrams is proposed based on the structure and component. The proposed method computes similarity between intended class diagram with previous developed diagram to use previous

knowledge. To measure similarity score, a framework is developed. This framework measures similarity into two phases. In first phase, UML diagrams are parsed by an XML parser and converted to XML files format. In second phase, similarities between diagrams are measured by comparing the structure and different components of UML class diagram to obtain similarity. Each of the diagrams are compared separately and finally the similarity score is measured by integrating all comparison results.

Experimental analysis has been conducted for evaluating the applicability of the proposed approach. The experiment justifies the approach that leads to the correct measurement. The experimental analysis was performed using seven different real life software projects by running the developed framework. The analysis shows that the precision, recall and F-measure of the tool are 0.83, 1 and 0.91 respectively.

The rest of the paper is structured as follows: Section two describes the related work on existing similarity measurement approaches and techniques. Some problems also figured out from the existing literature and some critical judgments also described. In section three, the proposed approach has been described in different subsections. The proposed approach is demonstrated with some necessary diagrams and algorithms. Fourth section presents the result analysis and validation of the proposed approach. Finally the conclusion of this research is presented in the fifth section.

Related Work

Many methods have been proposed on measuring similarity and retrieving reusable assets. This section describes some existing work on similarity measurement of UML class diagrams.

Several numbers of methodologies for comparing UML specifications have been proposed. Tsantalis et al., has been proposed design pattern detection approach based on UML class diagramby measuring similarity **(**Tsantalis et al., 2006**)**. Their approach consisted of agraph-matching algorithm used to compare two UML diagrams. Another technique for detecting differences between UML class diagrams and to visualize those differences using color as part of the incremental development process has been presented in **(**Girschick at. al, 2006**)**. Robinson et al., presented an interesting approach for comparing UML sequence diagrams, based on transforming them to a SUBDUE graph, in order to perform general information retrieval **(**Robinson and Woo, 2004).

A fuzzy logic based approach was proposed for measuring similarity between two software projects in (Idri and Abran, 2001). The approach was used for categorical data and the categorical data was described by a fuzzy sets. Fuzzy reasoning was used to compute the different measures and those were validated byan axiomatic validation approach and similarity between two projects were measured for categorical data. Idri et

al., proposed an approach based on fuzzy logic using linguistic quantifiers (Idri and Abran, 2001) that improved the work in (Idri and Abran, 2001). It claimed that most of the software projects cannot be used for measuring similarity when the projects are described by the linguistic quantifiers and overcame the problem. Authors stated that similarity between two software projects are not null and built a rule based engine for each attribute to find distance. This work used only linguistic values to measure similarity between two software projects.

Method for documenting continuous integration of software covering various perspective have been highlighted in (Danieland Bosch, 2014). The authors identified differences in continuous integration procedures in different types of software and proposed a model to document those. However, assuming the need to preserve reliability during continuous integration through studying historical projects have been considered to a limited extent.

Two approaches for measuring similarity between software projects based on fuzzy C-means clustering and fuzzy logic were presented in (Azzeh et al., 2008). The proposed approaches overcame the problems of nearest neighborhood techniques. First approach was developed based on identification features of fuzzy sets and second approach was based on partition matrix that is obtained by fuzzy C-means. They stated that first approach outperforms second approach based on their experimental results. This approach is not applicable for linguistic values and only suitable for numerical and categorical data.

Some specific research has been done for computing difference between class diagrams. A generic difference algorithm is proposed for computing similarity of two UML models which were encoded in XML les from de-sign diagram (Kelter, 2005). The implemented algorithm performed well on runtime for small documents but not good for large documents. A comparative result were presented using basic graph by denoting node and edge. In this approach, at first the elements of each document were detected and then calculated similarity by a defined function that worked with some predefined criteria. Weight was defined for each criteria in a way that may mislead to a missed correspondences.

Some other works have been presented that uses design diagrams for various purposes (Nahar and Sakib, 2014) and (Nahar and Sakib, 2015). Nadia et al., proposed an automated test generation framework named as SSTF that used the design diagrams (class, state and sequence diagrams) and the software source code (Nahar and Sakib, 2014). This tool used an XML converter to convert the UML diagrams in XML format, and identified the required information of test case semantics from those XMLs. Another work from the author was to recommend software design patterns using the design diagrams (class and sequence diagrams) (Nahar and Sakib, 2015). A tool named ADPR

was proposed in this paper, which detected anti-patterns in soft-ware design and recommended the corresponding design patterns. Here the XMLs of the class diagrams were stored in the tool as 2-dimensional matrices of prime numbers for maintaining the cardinality of the class relationships. Both these papers are the examples of usefulness of design diagrams for various purposes.

The uses of UML diagram is increasing day by day in the architecture and design phases of software engineering since its inception **(**Idri and Abran, 2001**)**. When using UML, software systems are described by constructing a set of diagrams. Usually, these diagrams are created independently and typically contained overlapping information. As a result, it is a challenging task to measure the appropriate similarity between UML diagrams. Without appropriate means for assessing the similarity between diagrams, inconsistencies are likely to arise or even worst remain undetected when they arise. The inconsistency between UML diagrams increases the chance of errors and potentially wrong similarity values between software designs projects. To solve those problems several researches have been done but they have some limitations. In this research, we proposed a different approach that can overcome those problems.

Similarity Measurement Approach

Class Diagram shows the static structure of a software system. It describes the system classes and interrelationships i.e. association, aggregation and generalization among objects, attributes and operations of the classes (Szlenk et al., 2006). In this section we presented an approach for computing similarity between two class diagrams encoded as XML files. Similarity of class diagram is measured based on the similar features between query and repository diagrams of different projects. The similarity between the query and the existing class diagrams of software projects in the repository are computed through a numeric computation and the computed value lies between 0 and 1. The proposed approach describes two similarity measures for computing the similarity during the multi-view similarity assessment. In this section we have discussed those measures into two phases. In the first phase, the similarity is calculated based on the structural view matching of class diagrams where the relationships were considered. In the second phase, the similarity is calculated based on the different components of class diagram where number of attributes, number of methods, number of relationships and number of classes were considered. Each of the phases is discussed in the subsequent subsections.

Structure based Matching

For structural matching of two classes, the class diagrams are presented as graph. In the view of a class diagram, the whole system can be compared by relationships of classes.

At first, class diagrams of a system are converted to XMLs and inputted to the tool. Then elements are parsed by an XML parser to proceed the next step. Hence, *classes* are considered as *nodes* and *relationships* are as *edges*. As there are different relationships exist among classes, so the edges should be weighted. For structural matching in the first step, matrix for a class diagram is retrieved from the main diagram. For keeping these relationship information a two dimensional matrix is used. The matrix is *n x n* prime numbered matrix as noted by Dong et al. (Dong, Jing and Yajing Zhao, 2007). Here, the usage of prime number is for following cardinality of the relationships when multiple relationships exist between two classes.

Table 1. Assigning Prime Number for Class Relationship.

Class Relationship	Prime Number
Association (*As*)	2
Generalization (*G*)	3
Aggregation (*Ag*)	5

As product of prime numbers is unique, it is possible to identify the types of the multiple relationships between classes. For example, Figure 1 shows two class diagrams and relations among each class of each diagram. Class diagram (a) has three associations, two aggregations and one generalization relationships among classes. Class diagram (b) has two associations, two aggregations and two generalizations relationships among classes. Multiple relationship between two classes are stored by their product value as products of prime numbers are unique, it can express the relationship types in existence of multiple relationships as well. Table 1 represents the defined prime value for each relationship. These prime value are used to represent the relationship between classes in matrix.

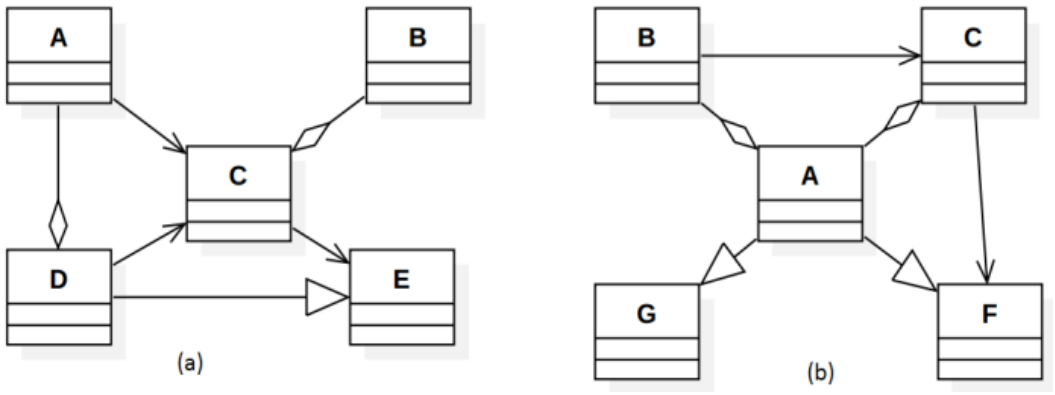

Fig. 1. Example of Class Diagram- (a) Diagram one & (b) Diagram two.

Fig. 2 represents the generated matrix of the Figure 1 where Figure 2 (a) presents the matrix of class diagram 1 (a). The relationship among classes of figure 1(a) is as following:

$$A \overset{Ag}{\rightarrow} D, A \overset{As}{\rightarrow} C, B \overset{Ag}{\rightarrow} C, D \overset{As}{\rightarrow} C, D \overset{G}{\rightarrow} E, and\ C \overset{As}{\rightarrow} E$$

Fig. 2 (b) presents the matrix which is generated from figure 1(b). The relationship among classes of figure 1(b) is as following:

$$B \overset{Ag}{\rightarrow} A, B \overset{As}{\rightarrow} C, A \overset{Ag}{\rightarrow} C, A \overset{G}{\rightarrow} G, A \overset{G}{\rightarrow} F, and\ C \overset{As}{\rightarrow} F$$

	A	B	C	D	E
A	0	0	2	5	0
B	0	0	5	0	0
C	0	0	0	0	2
D	0	0	2	0	3
E	0	0	0	0	0

(a)

	B	C	A	G	F
B	0	2	5	0	0
C	0	0	0	0	2
A	0	5	0	3	3
G	0	0	0	0	0
F	0	0	0	0	0

(b)

Fig. 2 : Generated Matrices of Class Diagram from Fig. 1.

After generating the matrix, the similarity is calculated based on the maximum matching of two class diagrams. For finding out the matching, we have implemented a customize Breadth First Search (BFS) algorithm as noted by Smith et al. (Smith and Plante, 2012). Then, the similarity score from the highest matching is calculated using a simple Jaccard Similarity equation (Jaccard, 1912). The highest number of classes found from the matching algorithm is divided by total number of classes for calculating similarity by structural matching. For example, Figure 1 represents two class diagrams and the similarity value obtains using this algorithm from the figure is 4/5. Because, from this figure, we get maximum 4 matching and highest number of classes in each diagrams are 5.

The algorithm is presented at Algorithm 1 which shows the whole matching process of structure. For structural matching, the matrix is generated in (Algorithm 1 Line 6). The value of edges are assigned based on relationship between classes (Algorithm 1 Line 8-17). Similarity assessment approach is presented at (Algorithm 1 Line 19).

Algorithm 1. Structure Based Matching

Input: XMLs (CD)
Result: Sim_{str}
Initialize$mat[u][v] \leftarrow 0$
Initialize $Setof Relations \leftarrow R$
for *edges(u,v) in diagrams* **do**
| *mat[u][v]* ← *getValues(2)*
end for
procedure *GETVALE*
if $relations \in Association$ **then**
setValue=2
else if $relations \in Generalization$ **then**
setValue=3
else if $relations \in Generalization$ **then**
setValue=5
end if
end procedure
$Sim_{str} \leftarrow matchBFS()$

Component based Matching

In Component Matching of class diagrams, at first we have selected some common criteria those are comparable between two classes. Based on these criteria, a similarity function has been defined to measured similarity. Some criteria of measuring similarity between two classes are noted by Ketle et al., (Kelter et al., 2005) and author also defined function for these corresponding criteria.By following this approach, we have defined criteria for our approach. The selected criteria for our approach are: Number of Attributes, Number of Operations, Number of Classes and Relationships.

However, we only considered *Number_of_Attribute* and *Number_of_Operation* for criteria matching for class diagram as we have considered relationships in the structural matching and measured similarity using number of classes. We also defined a function for measuring similarity that is demonstrated in the Algorithm 2.

Moreover, after structural matching we also used component matching because in every cases, structural matching cannot provide accurate similarity score. Thus, quantitative

value needed to be considered as an important fact of measuring similarity of class diagram.

A criteria matching algorithm is developed that is shown in Algorithm 2. For measuring criteria similarity, a class of first diagram is compared with all of the classes of the second diagram and stored in a matrix. With the similarity score a*nxn* matrix is formed.

Algorithm 2: Component based Matching

Input: XMLs (CD)
Result: Sim_{com}
Initialize $max \leftarrow -1$
Initialize $Set \leftarrow R$
for $i \leftarrow 1$ to n**do**
  $max \leftarrow -1$
  $matchedItem \leftarrow NULL$
  for $j \leftarrow 1$ to m**do**
    if $e_j^{'} \notin R$ **then**
      $score = compare(e_i, e^{'}_j)$
      If $max < score$**then**
        $max \leftarrow score$
        $matchedItem \leftarrow e_j$
      End if
    End if
  End for
  If $matchedItem\ != NULL$**then**
  $R = R \cup \{e^{'}_j \longmapsto matchedITem\}$
  Sim_{temp} + = *max*
  Sim_{com}=Sim_{temp}/*numberofItem*
End if
Endfor

All the time number of classes of a diagram will not same as comparing with class. As a result a*nxn* matrix need to generate. Then the similarity score is measured by parsing the matrix. The highest value between two classes are selected and the same class cannot be matched with any other class of the second diagram. The criteria similarity score is calculated (Algorithm 2 Line 20).

Finally the total similarity is calculated by integrating the value of Sim_{str} (Structure based matching) and Sim_{com} (Component based matching).We have used only arithmetic mean for measuring similarity score because there is no impact using arithmetic mean that is proved experimentally in the result section. A function is defined for calculating the score which is presented in the equation 1.

$$Sim(C1, C2) = W * Sim_{str}(C1, C2) + (1 - W) * Sim_{com}(C1, C2) \text{ ---------- (1)}$$

Here, C1 and C2 defined the respected class diagrams those will be compared. W is the weight of each stage. As we used arithmetic mean for measuring similarity, so there is no effect of W in this equation. Sim_{str} is the similarity score retrieved from structure based matching and Sim_{com} is the score obtained from component based matching. In the next section an experiment have been conducted for validate the proposed approach.

Experiment

This section presents the experiment with requirements to evaluate the similarity approach and compare the result with some existing methods. The experiment has been conducted on some UML class diagrams of some software project collected from different sources. Initially the data have been encoded into XML files and similarity has been computed. Towards this aim a framework has been developed in java language.

A. Experimental Data

The analysis was performed on 7 different software project requiring different diagrams those are used in this research. These projects have been collected from the student of Institute of Information Technology, University of Dhaka. Table II presents the project name along with class diagrams information. The collected datasets have been encoded into xml files as the developed framework took xml file as input. For converting UML diagrams into XML files we have used an open source converter Star UML. The dataset is uploaded into github which is available here (Dataset, 2018).

Table 2. Experimental Dataset Description.

Serial	Project Name	#Class diagrams
1	Inventory Management System (IMS)	5
2	Student Information System (SIS)	6
3	AmaderChakri.com (AC)	9
4	Program Office Management System (POMS)	5
5	Library Circulation System (LCS)	7
6	Cricket Circle (CC)	4
7	Cloud Portal (CP)	6

Result and Discussions

For experimental result, dataset projects were run using the developed framework. The similarity score was measured between two class diagrams. In the dataset, among 7 project IMS is the selected for query and other projects as repository where class diagram of IMS will be compared with other projects. Table 3 presents the similarity value of class diagrams were calculated using the developed framework. In this table, first column presents the query project (QP) and second column presents the repository project (RP).

Table 3. Similarity score of UML Class Diagram.

QP	RP	Sim_{str}	Sim_{com}	Similarity Score
IMS	SIS	0.79	0.84	**0.82**
	AC	0.49	0.63	**0.56**
	POMS	0.71	0.81	**0.76**
	LCS	0.77	0.81	**0.79**
	CC	0.41	0.51	**0.46**
	CP	0.42	0.53	**0.48**

Structure based similarity score is presented in the third column (Sim_{str}) and component based score is presented in fourth column (Sim_{com}). Last column presents the desired output obtained from the developed framework. Table 3 showed that the highest similarity is found between class diagram of IMS (Query project) and SIS (Comparable project). Now, we can sort the project based on their similarity score which is obtained from their class diagram comparison. From the table it is very clear that IMS is mostly similar to SIS and less similar to CC. Main purposes of this research is to find the best similarity among projects based on the class diagram. This result will help the software industry as well as researchers to reuse previous developed tool, code, system and different tools. For example, when the project SIS was developed, the quality assurance engineer used a testing tool T. From the table 3 it is very clear that IMS is mostly similar to SIS, so during the testing of IMS the testing tool T can be used (based on the recommendation of this research).For experimental analysis, the developed framework and dataset is available on (Implementation, 2018 and Dataset, 2018).

Evaluation of the Proposed Approach

For the justification of this approach, an empirical analysis was performed. We took helped from software experts (two software analysts from software industry) to conduct the evaluation process. We measured precision, recall and f-measure for justified this

developed approach. For these purposes, the result obtained from the developed approach is considered as actual result and the similarity result performed by design experts is expected result. Table 4 presents the expected result and actual result that was performed by the developed approach and the expert. After a deep analysis and discussion with experts, we concluded that the system provide similarity more than 60 % is similar to projects. However, similar projects were chosen based on a threshold value that is greater than or equal 0.6 (threshold ≥ 0.6) suggested by expert. Now, from the actual and expected result that is shown in the Table 4, the precision and recall of proposed method can be measured.

Table 4. Result analysis.

QP	RP	Actual Result (threshold≥0.6)	Expected Result (Expert)
IMS	SIS	Yes	Yes
	AC	Yes	No
	POMS	Yes	Yes
	LCS	Yes	Yes
	CC	No	No
	CP	No	No

Let, tp =true positive, fp =false positive, fn=false negative. From Table 4 we got tp = 5, fp = 1, fn = 0.Thus,

$$Precision = \frac{tp}{tp + fp} = \frac{5}{5 + 1} = 0.833$$

$$Recall = \frac{tp}{tp + fn} = \frac{5}{5 + 0} = 1$$

As, proposed method provides 1 false positive result, it possesses the precision 0.833 and1 false negative result which possesses the maximum recall. Using the precision and recall, the F-measure or the balanced F-score (F1score) can be calculated.

$$F_1 = 2.\frac{Precision * Recall}{Precision + Recall} = 2.\frac{0.833 * 1}{0.833 + 1} = 0.91$$

The analysis shows that it can effectively determine similarity since it has an excellent indicator with a precision of 0.83, recall of 1 and F-measure of 0.91 respectively which concludes that proposed approach for measuring similarity of UML class diagram performs well.

Threat to Validity

Some validity threats have been discussed for our proposed approach using the classification suggested by (Wohlin et al., 2012). First threat comes from the issues of data sets used in this research. As, we have worked on some UML diagrams from different SRS document of project those were collected from others. So, we have no control on the quality of the issues collected. Overall, we have tried to reduce this threat by modifying some diagrams to achieve the standard.

The second threat comes from the unstructured data sets as our collected datasets were not prepared by same person. Using defect data sets predicting result may deviate from actual result. However, we considered the selected projects stable in production. This problem is considered as an internal threat for this thesis because our proposed method need proper design diagrams for better performance during the evaluation of result. Besides this internal threat, we have identified an external threat for this research that is using the in-house class room projects. Due to the unavailability we cannot use the industrial project for the assessment of our proposed system.

Finally, we have identified another threat in the similarity approach. As we considered structure based and component based matching during the similarity calculation, it may provide contradict similarity. In that case, a good similarity score may be found in spite of being different projects.

Conclusion

Reuse of software can minimize the cost and time during development of any software. This benefit can be maximized if it is carried out at the early phase of development. Similarity is one way to provide the opportunities to reuse previous developed resources and this task can be done in early phase of development using UML diagrams. Therefore, the most important task of the comparison of two models is exactly the UML class diagram comparison and evaluation. There are a very few methods how to compare the UML class diagrams and they don't provide a valuable result.

This paper focused on an approach for measuring similarity between UML class diagrams with an aim to reuse resources during the development of any project. The approach is based on the structure and component metrics of UML class diagrams that is described in details in the methodology section. An empirical analysis has been conducted to evaluate the process. Moreover, for the justification of the proposed approach the precision, recall and F-measure were calculated that possesses a precision of 0.83, recall of 1 and F-measure of 0.91. The result shows that the proposed approach performs well. Currently, our approach is based on structure and component based

matching, in future different matching metric can be considered to improve the matching accuracy. Additionally, the results of this research need to be investigated based on some real life industry dataset. Moreover, the similarity of other UML diagrams such as sequence diagram and state transition diagram need to be calculated to measure the similarity among software projects.

References

M. Ahmed. 2011. Towards the development of integrated reuse environments for UML artifacts. Sixth International Conference on Software Engineering Advances, pp. 426-431.

W. Tracz. 1998. Software Reuse: Emerging Technology. New York: IEEE Press.

M. Chechik, S. Nejati, and M. Sabetzadeh. 2012. A relationship-based approach to model integration. Innovations in Systems and Software Engineering. 8: 3-18.

M. Alanen and I. Porres. 2003. Difference and union of models. UMLP. Stevens, J. Whittle, G. Booch, Ed. Springer. 2863: 2-17.

Z. Xing and E. Stroulia. 2005. UMLDiff: An algorithm for object-oriented design differencing. 20th IEEE/ACM International Conf. on Automated Software Engineering (ASE ’05). pp. 54-65.

K. Bogdanov and N. Walkinshaw. 2009. Computing the structural difference between state-based models. WCRE ’09 Conf., IEEE. pp. 177-186.

H. O. Salami and M. A. Ahmed. 2012. A framework for class diagram retrieval using genetic algorithm. 24th International Proceedings of Software Engineering and Knowledge Engineering (SEKE'12). pp. 737-740.

Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., & Halkidis, S. T. 2006. Design pattern detection using similarity scoring. IEEE transactions on software engineering. 32(11): 896–909.

Girschick, Martin, and Tu Darmstadt. 2006. Difference detection and visualization in UML class diagrams. Technical university of darmstadt technical report TUD-CS-2006-5. (2006): 1-15.

Robinson, W.N. and Woo, H.G., 2004. Finding reusable UML sequence diagrams automatically. IEEE software, 21(5): 60-67.

Idri A, Abran A. 2001. A fuzzy logic based set of measures for software project similarity: validation and possible improvements. Seventh International Software Metrics Symposium, METRICS 2001. IEEE. pp. 85-96.

Idri, Ali, and Alain Abran. 2001. Evaluating software project similarity by using linguistic quantifier guided aggregations. IFSA World Congress, 20th NAFIPS Int. Conference, IEEE. 1: 470-475.

Sthl, Daniel, and Jan Bosch. 2014. Modeling continuous integration practice differences in industry software development. Journal of systems and software (JSS), Elsevier. 87: 48-59.

Azzeh, Mohammad, Daniel Neagu, and Peter Cowling. 2008. Software project similarity measurement based on fuzzy C-means. International Conference on software process, Springer. pp. 123-134.

Kelter, Udo, JrgenWehren, and JrgNiere. 2005. A Generic Difference Algorithm for UML Models. Software Engineering. 64(105-116): 4-9.

Dataset, 2018.https://github.com/samsuddoha/ThesisDataset.

Nahar, Nadia, Kazi Sakib. 2014. SSTF: A novel automated test generation framework using software semantics and syntax. In Computer and Information Technology (ICCIT), IEEE. pp. 69-74.

Nadia Nahar and Kazi Sakib, 2015. Automatic Recommendation of Software Design Patterns Using Anti-patterns in the Design Phase: A Case Study on Abstract Factory. QuASoQ/WAWSE/CMCE@ APSEC. pp. 9-16.

M. Szlenk, 2006. Formal semantics and reasoning about uml class diagram. IEEE. pp. 51-59.

Dong, Jing, Dushyant S. Lad, and Yajing Zhao, 2007. DP-Miner: Design pattern discovery using matrix. In Engineering of Computer-Based Systems, ECBS'07. 14th Annual IEEE International Conference and Workshops, IEEE. pp. 371-380.

Implementation, 2018. https://github.com/samsuddoha/ThesisImplementation.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A., 2012. Experimentation in software engineering. Springer Science & Business Media.

P. Jaccard, 1912. The distribution of the flora in the alpine zone. New philologist. 11(2): 37-50.

Smith, S., D. R. Plante. 2012. Dynamically recommending design patterns. SEKE. pp. 499-504.