Indian Journal of Science and Technology, Vol 9(46), DOI: 10.17485/ijst/2016/v9i46/106918, December 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Review of UML Model Retrieval Approaches

Alhassan Adamu and Wan Mohd Nazmee Wan Zainon*

School of Computer Sciences, Universiti Sains Malaysia,
Penang - 11800, Malaysia; kofa062@gmail.com, nazmee@usm.my

Abstract

The objective of this review is to obtain an overview of the current state of the art of the existing approaches in matching and retrieval of UML diagrams. The paper presents a synthesis of key characteristics of the current available approaches of UML-based reuse, compare their matching and retrieval techniques identified their commonalties and differences. A number of related research papers where examined and categorized based on the type of approach they adopt. The review resulted in the identification of four main categories of UML models matching and retrieval: 1. Information retrieval, 2. Case-based reasoning, 3. Ontology-based, and 4. Structured based approach. A comprehensive overview of these approaches is presented. The findings of this review suggest the further research and practice in UML models reuse.

Keywords: Artefacts, Models Matchings, Model Retrieval, Software Reuse, UML Models

1. Introduction

There are three most important phases software development: analysis, design, and implementation. In the analysis phase, software developers analyze user's requirements and identified what the proposed system should do. In the design phase, detailed specification of how the software should perform its tasks is developed; at this stage the developer translates user requirements into a real world system, with a greater emphasis on any possible technical hurdles. Finally, throughout the implementation phase, programmers implement the new system based on the functionalities identified during the design phase. The choice of a programming language depends largely on the expertise of the programmers. Software engineers, without effective management, can find themselves re-inventing these software development processes. It is very possible to create new software from the beginning while similar software was developed before. In most engineering disciplines, systems are built by cloning existing components that is used in other systems rather than building them from scratch. But it is not the case in software engineering, which has often focused more on original development. This has resulted in duplicated software artifacts, increased maintenance costs and the ineffective use of specialists. Software reuse was introduced to counter this inefficiency with a paradigm of identifying existing software system to build new system rather than building it from scratch[1].

Software reuse promotes accelerated development, reduces process risk, utilizes specialists effectively, reduces development time, improves productivity and increases the overall quality of software products[2]. However, it does suffer from a dearth of supporting tools, increases maintenance costs, the not invented here syndrome, a unavailability of reusable component libraries, the additional cost of making components reusable, and the cost of reusing such components[2,3].

Software reuse can be categorized into deliberate reuse and accidental reuse. In deliberate reuse, components are purposely developed to be reused in the future; but many organizations are unwilling to outlay this initial cost since there is no guarantee that a developed component can be use in the future. In contrast, accidental reuse is easier because developers only come to the conclusion that a past component is reusable when it is found worthy of incorporating into a new system[3,4].

There are many areas of software reuse, it includes past successful software requirements, software designs, software code, test cases, and documentation. Source code

*Author for correspondence

reuse is the most common type of reuse. However, it is not the most efficient place for reuse to occur[5]. The reuse of design is a more complex and demanding task because the product of design is often intangible, and intimately linked to the experience and expertise of individual software engineers[6].

During system development, design plays a critical role in ensuring the quality of the software system. Designs are normally represented using models/diagrams and capture the development concerns of a particular problem domain. Successful designs can be reused to develop other systems. This paper reviews the existing work on the reuse of past designs represented in Unified Modeling Language (UML). Section 2 of this paper describes the background of software retrieval in the context of this work. Section 3 discusses the various techniques of UML diagrams matching and retrieval. Section 4 covers the query formulation and Section 5 presents the method of evaluating retrieval methods. Finally, the conclusion is presented in Section 6.

2. Software Retrieval

Matching and retrieval of software components are considered to be the main building blocks for software reuse. While the reuse of UML diagrams is becoming popular in the software engineering community, finding and retrieving appropriate UML diagrams remains a challenging task.

Previous software arte facts are kept in software library or repository for later use. Locating components within the repository is essentially a search problem because of the increase of repository size.

There are four phases involve in reusing existing components: representation, retrieval, adaptation and integration[4,7]. In the representation stage, the user presents a new software component query for comparison. During the retrieval stage, software components that correspond to user query are shortlisted and ranked based on defined similarity metric. Similarity metric is a function that returns degree of similarity between two software artefacts[8]. During adaptation, components with minimal adaption cost are selected and modified to suit the new requirement. Finally, the new components are integrated into the repository for future reuse[9].

3. Retrieval Approaches

This section discusses the UML diagrams retrieval approaches based on the approaches. Each work is categorized based on the type of approach adopted as follows: Information Retrieval, Case-Based Reasoning Approach (CBR), Ontology Approach and Structure-based Approach.

3.1 Information Retrieval

Information Retrieval (IR) is one of the earliest approaches use for UML models retrieval. IR is a technique for comparing and finding documents that meet the information needs from a collection of documents[10]. The data stored and manipulated by IR system can be in any form such as textual, video and audio or multimedia documents. However, IR is also applicable to UML diagrams retrieval especially if the diagrams contained some considerable amount of text (e.g. use case description)[11].

The similarity between requirement speciation in query and repository is computed using IR[12]. The requirements specification were in the form of use case flow of events. The similarity is computed by the number of matching use case events flows in query and repository. Similar events flows from the same or similar domain are grouped together to form clusters based on their lexical meaning of words. The clustering process reduces the complexity of matching process considering a large number of different events flows in the domain model. Each of the use case's event flows is represented as a multi-dimensional vector space model, in which the dimension represent the number of events in a particular cluster. The similarity of two requirement specifications is computed using cosine distance measure.

Information retrieval technique is applied for scenario management reuse[8]. Each of the scenarios is represented as a set of attributes: goals, authors, events; actors, actions, and episodes. The similarity between two scenarios is computed as the sum common attributes values in the attribute list divided by the sum of the sizes of each attribute list.

An approach of retrieving software projects from repository based on faceted classification scheme using IR was proposed[13]. Models are classfied in to six perspectives: domain, abstraction, responsibilities, collaborations, design views, and asset types. Each of the facets describes

one aspect of the component and also capture the component functional requirements. During retrieval the similarity between query and repository models are computed using conceptual closeness and discrepancy reation. The conceptual cloness computes the similairty between models from different facets represented by taxonomy. The discrepancy ratio measures the degree of commonality of descriptor terms in query and repository models.

A framwework of retriving UML artifacts in two stages is describe[14]: 1. Indexing, in this stage UML diagrams elements (e.g. classes, attributes, and collaboration) in XMI format are extracted and stored in relational database, 2. In the retrieval stage, the similarity between UML diagrams is computed using query inclusion and query similarity. The query inclusion searches through the repository using a SELECT statement to retrieve only those diagrams that are defined in the query; this includes the sub-artefacts and relationships between them. Query similarity consists of topological and semantic similarity of the diagrams. The topological similarity is computed based on the type of relationships found in query and repository diagrams. The relationships are represented by vector space in which the dimension of the vector space represents the number of relationships in query and repository. The similarity between UML diagrams was calculated as the Euclidian distance of two vector space. Finally, the semantic similarity is computed based on the degree of overlap of terms occurring in query and repository models together with the distance between terms occurring in relationships within a thesaurus.

3.2 Case based Reasoning Approach

A decade ago, researchers have explored the use of case based reasoning in UML reuse. Case Based Reasoning (CBR) is an analogical paradigm whereby new problems are solved by adopting solutions of similar past problems[15]. CBR is an experienced driven paradigm, in which previous experiencesare stored as cases in a case library. Cases in CBR are composed of problem and their solutions. CBR is composed of five parts: situation and its goals; solution and its source; the result of carrying out the solution; clarification of the result and lesson learned from the solution[16].

CBR was combined with a WordNet lexical ontology to retrieve previous software design cases in three stages: retrieval, verification, and retention stage. A case here refers to a previous design experience stored in a case-base library. Class diagrams are stored as cases in a case library. During retrieval three kinds of objects are considered: Packages, classes, and interfaces. The retrieval takes place in two stages: the first stage is computationally inexpensive otherwise known as pre-filtering based on WordNet relations to index the case library. The second stage is computationally expensive stage; the similarity between case in query and case-base is computed, similar cases are returned and ranked based on their similarity values. The latter is more accurate for object selection and ranking. The most similar cases is adapted for reuse in the new project. In verification stage, cases are checked for consistency and coherence to assess their performance and characteristics. Finally, at the retention phase, the system decides either the new case should be stored in the case library or not. Cases that are similar to existing cases are discarded as redundant[6,17-19].

Class diagrams were retrieved from case library based on the number of matching elements (class, relationship, direction, and multiplicity) found in the diagrams[20]. Class diagrams are retrieved using CBR and a graph matching algorithm in two stages. In the first stage, classes are assigned a weighted function reflecting their degree of influence in a class diagram. With the CBR the similarity between classes in the query and those in case library is computed. This stage is computationally inexpensive. In the second stage, the query and repository classes are converted to a weighted undirected graph with nodes denoting classes and edges represent the relationships between classes. With the aid shortest path algorithm the match between a pair of class diagrams was calculated as the distance of their shortest path length.

In ReDSeeDs projects previous software requirements and solutions are stored as cases in case library. The requirements are written using Requirement Specification Language[21] in three forms: scenario based, structure based and model based. During the retrieval, new requirement is compared with the previous requirements stored in the library. The requirements are used as the indexes that link to corresponding cases (designs, code) in the case library[22]. The most similar requirements are returned for adaptation. During adaptation, a transformation engine generates new interaction diagrams, business logic and application logic code from the new requirements[23].

An approach of retrieving previous use case diagrams from repository using CBR was presented[24]. The retrieval method is based on two dimensions: use case

and actor dimension, and relationships dimension. The use case and actor dimension consist of use case, actor, and system boundary components represented as text-based information. During retrieval, the words found in query and repository are extracted and formed a searched dictionary. The similarity is computed as the average of the number of matched words found between query and repository use case diagrams. In the relationships dimension, the similarity is calculated based on three subcomponents: the relationship type, navigator, and multiplicity relationship. Each of the relationships is assigned a weighted value indicating the influence of the relationship in the diagram. Finally, the actual degree of similarity is returned by the CBR engine, and the appropriate diagrams are selected for reuse.

3.3 Ontology based Approach

Ontology has a good way of specifying concepts and the relationships among those concepts, especially those concepts that are in the same or similar domain. Ontologism are ways in which information are organized to promote sharing and reuse of such information.

A WordNet specific ontology is used to classify use case event flows into lexical and semantic similarity[12]. Similar event flows in the domain are grouped into clusters by their names. The event flows in queries and the repository were represented in a multi-dimensional vector space, in which the dimension denotes the number of events in a cluster. Finally, the distances between the vectors are measured to determine the degree of similarity between the two requirement specifications.

Lexical Semantic ontology such as WordNet can be used to determine the meaning of words in a models diagrams. WordNet is being applied in combination of another approach such as CBR[6] and IR[11,12,25] to improve the retrieval results.

Similar UML diagrams were retrieved using two types of ontologies: Application ontology and domain ontology. The application ontologies measure the semantic similarity between UML class diagrams based on the relationships between their classifier and identifiers. The domain ontology measures the semantic relationship between classifier names in the class diagrams. During retrieval, each UML class diagrams in the repository are indexed and characterized according to application ontology. The overall similarity between query and repository diagrams is calculated as the weighted sum of both application and domain ontologies similarities[26].

Information about UML use case diagrams are stored in Ontology Web Language (OWL) database. During retrieval users query individual OWL entries in the database and retrieve the associated use case diagrams in XMI forms based on user defined parameters. The user defined parameters could restrict the search space to avoid the unwanted query expansion. The information in the parameters are interpreted as a query over OWL ontology and stored in an MYSQL database. The tool searches the individual use case diagrams that match the query and return the final result in XMI format for reuse[27].

Many other studies have adopted WordNet ontologies to compute the similarity between UML diagram[28-30] to determine the semantic similarity between class names, attributes, names, operations in class diagrams respectively.

Semi-automatic approach to adapt UML activity diagrams to create new use case diagrams is proposed[31]. The information regarding use case diagrams, activity diagrams and a class diagrams is stored in a model repository. Consequently, the similarity of two use cases is computed based on their semantic similarity. The semantic similarity is computed in two aspects: the similarity of the sole use cases and the similarity of the context in which the use case exist. The measure of the semantic similarity is based on WordNet. Finally, the semantic similarity of two use cases is computed as the weighted sum of their similarity values.

3.4 Structural-based Approach

Structural-based similarity considers the structural representation of models to in query and repository. There are two approaches for computing the similarity of structured models: Graph-based and Description Logic (DL)[25,32]. The similarity of two models is computed by comparing the vertices and arcs in the equivalent graphical representation of the models. Both of these approaches focus on the models structural representation and compare subgraph using taxonomic comparisons of model elements and their relationships to other elements. Normally vertices represent the models name (e.g. class names), and the arc denotes the relationships between UML models elements.

The similarity computation is based on the estimation of the conceptual distance between terms in the query and the terms in repository models[13].

Similarity between sequence diagrams was computed using SUBDUE[34] graph matching algorithm. Sequence diagrams are represented as conceptual graphs in which the object names in the sequence diagrams represents vertices, and the relationships between the diagrams (messages) represent the edges of the graph[33]. The SUBDUE algorithm find the similarity between the graph by comparing the substructures of sequence diagrams in query and repository.

A two-stage framework to retrieve UML artifacts from repository is proposed[7]. In the first the similarity between class diagrams is computed using Structured Mapping Engine (SME). SME is an analogical reasoning mapping techniques which allows mapping of knowledge from one domain to another by considering the communalities between objects in the domain regardless of the objects involved in the relationships. The subset of the repository UML projects are selected for subsequent comparison, the first stage is considered as the pre-filtering stage. In the second stage sequence diagrams in the shortlisted models are converted to Message-Order-Graph (MOOGs), where nodes denote the location where events occur (message send or received) in sequence diagrams and the edges denote the flow of events between objects and the flow of time inside each object. The similarity between two MOOGs is computed based on the number of nodes and edges in each of the graph using graph matching algorithm.

A framework for retrieving class diagrams using genetic algorithm[35]. UML class diagrams are converted to graphs, in which classes denote the nodes of the graph and the relationships between the classes denotes the edges of the graph. The similarity of class diagrams is computed using genetic algorithm using fitness function that relies on similarity measures based on the concept names (class names) on the graph topology.

Similarly, particle swamp optimization algorithm was used to retrieve similar class diagrams[36]; the similarity between two class diagrams is computed as an aggregate of classifier similarity and relationship similarity.

In another work multi-objectives algorithms is used to retrieve similar class diagrams from repository[37]. Class diagrams are converted to directed graph in which the class represents the nodes of the graph and the relationship between the classes represents the edges of the graph. The similarity between class diagrams is computed using the relationship and name similarity. The relationship similarity measure the topological similarity of the diagrams while the name similarity measure the conceptual closeness between concepts in the class diagrams using Levenshtein Distance[38]. The fitness function is the aggregation of the two similarity measure[39].

Sequence diagrams are converted to a directed graph, the similarity between the graphs was determined with the aid of Genetic Algorithm (GA)[40]. The GA helps to terminate the searching process in order avoid exhaustive comparison. The termination criteria is based on three conditions: first if the fitness value reaches 0 indicating the maximum similarity between class diagrams, second if the maximum number of iteration reached, or if the fitness function does not improve within a given number of iterations.

State machine diagrams are converted to labeled directed graphs in which the states in the state machine diagram denotes the node of the graph and the transition in the state machines denotes the edges of the graph[41]. Information regarding state machine diagrams is stored in an adjacency matrix; the similarity between two state machines is computed based on their graph representation using Different matrix (DiffE). The DiffE act as a lookup table which indicates the degree of similarity between the different types of edges in the state machine diagrams.

3.5 Discussion

The existing work on UML models retrieval can be classified into two: based on the textual description of the models (i.e. matching of the diagram concept) or relationship between the models (i.e. the structural representation of the models).

Concept matching is the most common way used in finding the match between elements in the diagrams. IR approach is being applied by many researchers to retrieved similar diagrams from the repository by matching the concepts contained in the models, especially those models that contained considerable amount of text such as use case diagrams. However the drawback of IR approach is that two models are considered equals if they contain similar words in the same frequency. Ambiguity here arises when the two models representation/structure is the same but the actual meaning is different. The IR approach does not solve the ambiguity problems. Moreover, the structural representation of the models is not capture by the IR approach. Therefore two models

with the same words meaning but with opposite procedures are considered to be equal in this approach.

Basic CBR approach uses the traditional IR measure for string comparison. In CBR, two cases are considered equal if they share the same case representation[25]. CBR application provides a good way to retrieve models that belong to the same domains. However, with reuse spans to a variety of applications domains the CBR approach do not address the ambiguity problems.

Lexical Ontologies are used to determine the meaning of words in the models and can be used to overcome the ambiguity problem in IR and CBR. Several similarity measure have used WordNet lexicon to measure similarity between synset pair, noun pairs, and verb pairs. WordNet measures the distance between two synsets based on their path length defined by their semantic relations. Two different synsets are considered similar when there is a short distance in their path length. However, WordNet needs to be applied in combination of other approaches such IR or CBR to solve the ambiguity problem.

In contrast to concept matching, relationship matching extends the comparisons among the attributed concepts to include concept relationships. The approach relies on computing the structure of the diagrams by converting the corresponding diagrams to a graph's structure in which the concepts denotes the vertices and relationships denotes the edges of the graph. The graph-based approach of diagrams retrieval computes the similarity for all potentials matches between the elements within the diagrams. It is well suited for comparing the diagrams with a structured format like sequence diagrams. However, this approach relies only on structural representation of the UML diagrams neglecting the conceptual information inside the diagram.

4. Query Formulation

A query is a pre-requisite for component retrieval. It is a way of formulating a request that can select some components as a result of satisfying some similarity criteria. To find the most similar diagram from a set of previously designed UML diagrams, the user should formulate a query and send that query to the components repository for possible matching and retrieval. While most of the existing retrieval engine is based on the text box to search for retrieval terms, a retrieval search engine for UML information is thought to be graphical. For example, suppose a new banking information system is to be design and the designer has the system analysis and design in the form of use case diagram. The designer extracts some of the initial entities identified during system analysis and drawn in a class diagram. The initial diagram represents part of the classes identified in the system requirements. To achieve this process some authors uses existing case tool such Object Aid[39], Altova[28,35] to obtained the equivalent XMI documents of the UML diagrams. The XMI documents serve as the input to the UML retrieval engine, in which the retrieval engine provides a concrete solution to the UML elements mapping problem.

Query in XMI format is transformed into an information representation model based on relationships between the artifacts and the sub-artifacts using RSPH[42]. An XMI parser detects the structure of the document to ascertain if the document conforms to the XMI standard. The Parser identifies the XMI elements (attributes, classes, relations) in a model and identifies the relationships among those modeled elements. The similarity of two documents is calculated based on their semantic distance of the common concepts in both documents and the common RSHPs of both documents[14].

UML documents in XMI form was indexed using .NET application to categorize the class diagram elements according to their application ontology and stored in a knowledge base[26]. The class diagram in the query was also indexed the same way as the knowledge base. The similarity between the two XMI documents is calculated as the weighted sum of different class diagram categories. The authors used Poseidon[4] CASE Tool to obtain the XMI equivalent for the UML class diagrams.

5. Evaluation Procedure

Various studies evaluated their work using standard information retrievals such as Precision and Recall; the idea is to test whether the proposed retrieval approaches produce a reasonable degree of matching between query and repository diagrams. Precision is the proportion of retrieved documents/diagrams that are similar to the user query while recall is the proportion of matched repository diagrams that are retrieved. There is trade-off between these two measures; for instance high recall with low precision can be achieved by retrieving all repository diagrams. On the other hand high precision but low recall can be achieved by not retrieving any of the repository diagrams[10]. F-measure combines these two measures in one single value; it is the weighted harmonic mean of pre-

cision and recall depending on the relative importance of a measure. It is worth noting that the result of various studies will be difficult to compare due to unavailability of open datasets since most of the study rely on textbook examples while other studies rely on reverse engineering from source code. Moreover, the varying size of query and repository diagrams used in various studies makes it hard to compare the result of two different studies.

6. Conclusion

In this review, we discussed the current state of the art of UML diagrams reuse. Information retrieval is one of the earliest approaches to retrieved similar UML diagrams from repository. Information retrieval techniques are mostly in the UML diagrams that contained a considerable amount of text, for example use case diagrams. Other retrieval techniques combine IR with other techniques such as WordNet lexicon ontology. Many studies applied WordNet lexicon ontology because it is freely available. However, the limitation of using WordNet is a lack of its incorporations of technical information regarding the domains of specific knowledge because it is just a lexical database of English. Therefore, it is not surprising that WordNet is applied in conjunction with other retrieval techniques such IR and CBR. The use of Case-based reasoning holds a promising future in software reuse because of its capability in allowing the re-user to adapt the retrieved diagrams into new software design automatically, thereby reducing the effort required during reuse. It is worth to note that most existing approaches use more than one retrieval techniques.

UML consist of fourteen diagrams; class, use case, and sequence diagrams are the most type of diagrams considered by the majority of the authors. There is only little work on reusing the other types of UML diagrams.

7. Acknowledgements

This work was supported by the Ministry of Higher Education of Malaysia, under the Fundamental Research Grant Scheme (FRGS: 203/PKOMP/6711533).

8. References

1. Krueger CW. Software reuse. ACM Computing Surveys (CSUR). 1992; 24(2):131-83.
2. Sommerville I. Software Engineering. 9 ed. Vol. 9. Addison-Wesley United State: Pearson Education, Inc Publishing; 2011. 790.
3. Keswani R, Joshi S, Jatain A. Software reuse in practice. IEEE 4th International Conference on Advanced Computing and Communication Technologies (ACCT); 2014.
4. Salami HO, Ahmed MA. UML Artifacts Reuse: State of the Art. 2014.
5. Prieto-Diaz R. Status report: Software reusability. IEEE Software. 1993; **10**(3):61-6.
6. Gomes P, et al. Case retrieval of software designs using wordnet. ECAI; 2002.
7. Park W-J, Bae D-H. A two-stage framework for UML specification matching. Information and Software Technology. 2011; **53**(3):230-44.
8. Alspaugh TA, et al. An integrated scenario management strategy. IEEE Proceedings of International Symposium on Requirements Engineering; 1999.
9. Salami HO, Ahmed M. A framework for reuse of multi-view UML artifacts. 2014.
10. Manning CD, Raghavan P, Schutze H. Introduction to information retrieval. Vol. 1. Cambridge: Cambridge University Press; 2008.
11. Wolter K, Krebs T, Hotz L. Determining similarity of model-based and descriptive requirements by combining different similarity measures. 2nd International Workshop on Model Reuse Strategies (MoRSe); Beijing, China. 2008.
12. Blok MC, Cybulski JL. Reusing UML specifications in a constrained application domain. Proceedings of IEEE Software Engineering Conference; Asia Pacific. 1998.
13. Ali FM, Du W. Toward reuse of object-oriented software design models. Information and Software Technology. 2004; 46(8):499-517.
14. Llorens J, Fuentes JM, Morato J. Uml retrieval and reuse using xmi. Proceedings of the IASTED International Conference on Software Engineering; 2004.
15. Aamodt A, Plaza E. Case-based reasoning: Foundational issues, methodological variations and system approaches. AI Communications. 1994; **7**(1):39-59.
16. Bergmann R, Kolodner J, Plaza E. Representation in case-based reasoning. The Knowledge Engineering Review. 2005; 20(03):209-13.
17. Channarukul S, Charoenvikrom S, Daengdej J. Case-based reasoning for software design reuse. IEEE Aerospace Conference; 2005.
18. Smialek M, et al. Complementary use case scenario representations based on domain vocabularies. Model Driven Engineering Languages and Systems. 2007; 544-58.
19. Bildhauer D, Horn T, Ebert J. Similarity-driven software reuse. ICSE Workshop on Comparison and Versioning of Software Models (CVSM '09); 2009.

20. Straszak T, Wolter K. Comprehensive system for systematic case-driven software reuse. Theory and Practice of Computer Science (SOFSEM); 2010. p. 697.
21. Srisura B, et al. Retrieving use case diagram with case-based reasoning approach. J Theor Appl Inf Technol. 2010; 19(2):68-78.
22. Wolter K, Krebs T, Hotz L. A combined similarity measure for determining similarity of model-based and descriptive requirements. Proceeding of the Artificial Intelligence Techniques in Software Engineering Workshop (AISEW) at the ECAI; 2008.
23. Robles K, et al. Towards an ontology-based retrieval of UML Class Diagrams. Information and Software Technology. 2012; 54(1):72-86.
24. Bonilla-Morales B, Crespo S, Clunie C. Reuse of use cases diagrams: An approach based on ontologies and semantic web technologies. Int J Comput Sci. 2012; 9(1):24-9.
25. Paydar S, Kahani M. A semi-automated approach to adapt activity diagrams for new use cases. Information and Software Technology. 2015; 57:543-70.
26. Gonzalez-Calero PA, Diaz-Agudo B, Gomez-Albarran M. Applying DLs for retrieval in case-based reasoning. Proceedings of Description Logics Workshop (Dl'99); Linkopings Universitet. 1999.
27. Robinson WN, Woo HG. Finding reusable UML sequence diagrams automatically. Software. 2004; 21(5):60-7.
28. Jonyer I, Cook DJ, Holder LB. Graph-based hierarchical conceptual clustering. The Journal of Machine Learning Research. 2002; 2:19-43.
29. Salami HO, Ahmed M. Class diagram retrieval using genetic algorithm. IEEE 12th International Conference on Machine Learning and Applications (ICMLA); 2013.
30. Wesley Klewerton Guez Assunc SRV. Class diagram retrieval with particle swarm optimization. 25th International Conference on Software Engineering and knowledge Engineering (SEKE); 2013. p. 632-7.
31. Assuncao G, Klewerton W, Vergilio SR. A multi-objective solution for retrieving class diagrams. IEEE Brazilian Conference on Intelligent Systems (BRACIS); 2013.
32. Levenshtein VI. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady. 1966.
33. Assuncao WKG, Vergilio SR. Class Diagram retrieval with particle swarm optimization. The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE); 2013.
34. Salami HO, Ahmed M. Retrieving sequence diagrams using genetic algorithm. IEEE 11th International Joint Conference on Computer Science and Software Engineering (JCSSE); 2014.
35. Ahmed M, Salami HO. Behavior-based Retrieval of Software. 2015.
36. Salami HO, Ahmed MA. A framework for class diagram retrieval using genetic algorithm. SEKE. 2012.
37. Morillo JL, Fuentes JM, Diaz I. RSHP: A scheme to classify information in a domain analysis environment. ICEIS (2); 2001.