IRA-International Journal of Technology & Engineering
ISSN 2455-4480; Vol.06, Issue 03 (2017)
Pg. no. 38-43
Institute of Research Advances
https://research-advances.org/index.php/IRAJTE

Retrieval of Similarity Measures of Code Component

Prof. M. K. Patil
Assistant Professor, Department of Computer Applications
BVU AKIMSS, Solapur, India.

Prof. Dr. P. P. Jamsandekar
Professor, Department of Computer Applications
BVU IMRDA, Sangli, India.

Type of Reviewed: Peer Reviewed.
DOI: http://dx.doi.org/10.21013/jte.v6.n3.p1

How to cite this paper:
Patil, M., & Jamsandekar, P. (2017). Retrieval of Similarity Measures of Code Component. *IRA-International Journal of Technology & Engineering* (ISSN 2455-4480), 6(3), 38-43. doi:http://dx.doi.org/10.21013/jte.v6.n3.p1

ABSTRACT

Modern programming languages, especially object oriented languages facilitate to create libraries of reusable components (e.g. class definition). The majority of software companies are designing the components and reusing those wherever applicable. Maintaining such components (i.e. class library) and accessing those at right time in right form is challenging because large no. of components in library. Object Oriented Programming supports the reusability of the code. The major challenge in programming is to improve the learning quality and productivity of the software developer, subject teachers and students.

To support programming in Java, researcher implemented a design retrieval algorithm which will make it possible to search through potentially reusable Java classes. The proposed work, selects the appropriate descriptors of the inputted cases - .java files. It will separate the code components automatically and stores in the repository. The different levels of ambiguity in selection of cases are controlled through data preprocessing technique of data mining. The set of adjustments applied to get the similarity of the code components.

Keywords*:* *code reusability, retrieval, ambiguity, data mining*

Introduction

In software development, to increase higher productivity we need to reuse class libraries. The most of class libraries are effectively used in object oriented programming (OOP). To get access and retrieve the class libraries based on the problem situation is one of the difficult tasks. This mechanism is helpful for novice developer. Each engineer or programmer acquire specific knowledge on his own, which is then reused in other projects and tasks wherever required. OOP supports the reusability of the code. Most modern languages like Java help programmer to create good libraries. Creating case library and reusing it can help to think more clearly about what it supposed to do, thus help refining the design process. The researcher wants to make it easier for the programmers to make use of their sources contained in these libraries. The major challenge in programming is to improve the learning quality and productivity of the software developer, subject teachers and students. To support programming in Java, researcher implemented a design Retrieval Algorithm which will make it possible to search through potentially reusable Java classes. The algorithm is based on case-based methods to support retrieval and reuse.

Case Based Reasoning (CBR)

CBR can be described as the process of solving new problems based on the experience coming from similar past problems. In general, CBR cycle [1] can be described by the following:

Retrieve	When a new problem arrives the most similar cases are *retrieved.* *Retrieve* is the process of remembering a relevant experience or set of experiences
Reuse	Their solutions *reused* to provide a proposed solution
Revise	A proposed solution which may be *revised* after testing to create a final solution
Retain	As a final stage the new problem and solution can be *retained* as a new case in the case base, allowing the system to learn new knowledge
Case Base	Stores previously solved problems with their solutions
Case	Records several features and their specific values occurred in that situation

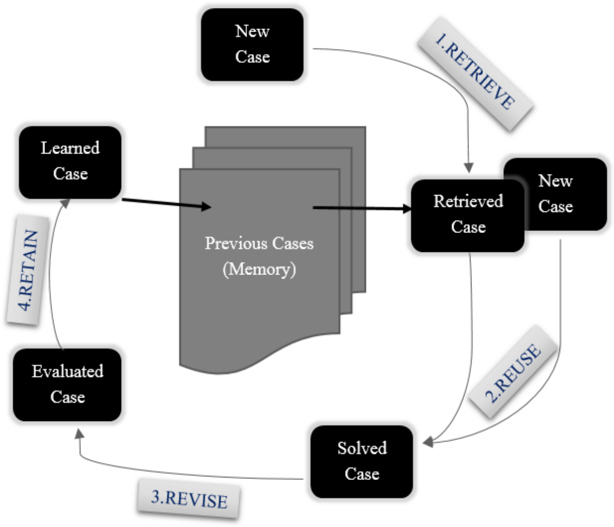

Figure (a): CBR Cycle [1]

To have an efficient retrieval, CBR plays vital role by reusing the similar past experiences of problem solving. OOP paradigm has some style of problem solving which is generalized to store as case and reuse wherever applicable. To retrieve the most appropriate experience (stored case) one need to have efficient retrieval method. A purely case based reasoning approach is adopted for OOP class library reuse.

Working Model

The researcher carried out the following module to retrieve effectively the code components. The module structure is depicted in following figure (b).

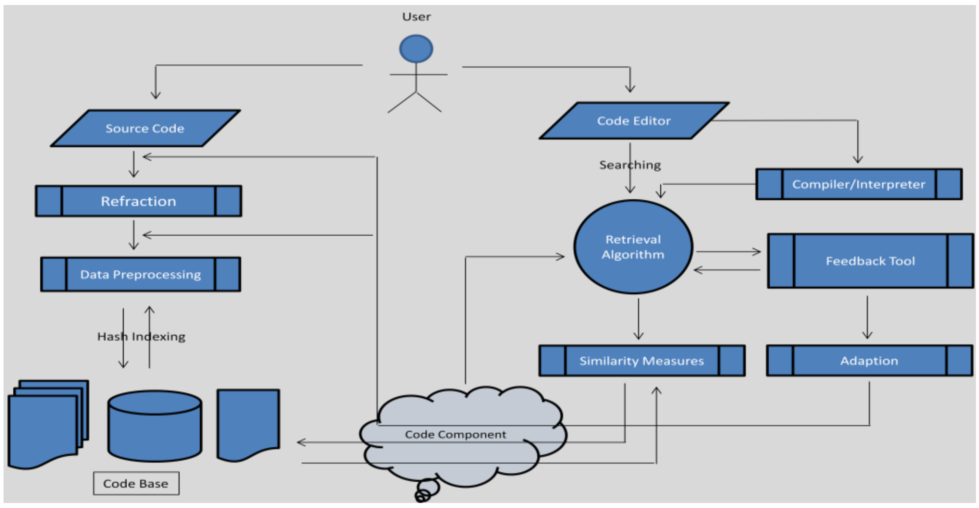

Figure (b): Working Module

Module Description

1. Input – Output Process
   This code block requests user inputs or in other words requests for new case inputs. These new case descriptors stored in appropriate variables and further processed using the Similarity Measure code block to find suitable matches for the test case in the library.

```
User Input Code Snap:

class MyString {
char[] data;
int length;
MyString(){}
MyString(char c){}
MyString(int i){}
void Compares(String s);
}
```

2. Case Base, Data Sources and Connectors
This code block initializes the case library to be used by the algorithm. The case library is 'hard coded' into the using databases or text files. The code block simply initializes all the cases with their predefined values. It uses java reflection method.

3. Case Retrieval Algorithm
The purpose of algorithm is to help the developer to locate reusable code and to aid in program understanding and adaptation. The tool matches Java classes from the class repository (base cases) to the target case (the class under construction) and then suggests similarities between them. It ensures that the automated retrieval and adaption strategies will be immediately useful and work with existing software repositories.

The case-based reuse tool supports retrieval and reuse of classes based on their signatures (methods return types and arguments etc.), which in this case is viewed upon as cases. The reuse component may suggest mappings between signatures of a retrieved case and the target, and the user may accept or discard the suggestions. Java's reflective capabilities are used to extract case descriptions from compiled Java classes and case-based reasoning is applied to support retrieval and adaptation of reusable components.

The retrieval algorithm worked for the code (1) as:
The retrieval fetched the data from the repository uploaded around 1200 cases. It took about 87 seconds to retrieve the selection based retrieval. That means,

Code Component	Code	Cases Selected	Selection %	Similarity %
Class	MyString	89	7.41	53 – 76
Method Name Only	Compares	19	1.58	9 – 11
Method Name with Parameter	Compares (String s)	14	1.16	7

Table 1.0: Case Selection

The retrieval and case selection is totally based on the repository data.

4. Similarity Measure Module
This is the centerpiece of the algorithm providing the match information. In this part of the program every attribute of the test case is compared with its counterpart in a library case and their match is normalized [2]. This process is done for every library case and the library case with the least error

is chosen as the solution for the test case. The algorithm utilizes the cases provided in the case library and uses them as a reference to compare with when it is tested with a test case. A Similarity Measure code block is developed to serve this purpose. The results of the pattern matching code block provide the degree of match, which provides the user intervention in the results.

The similarity measure, worked on the basis of selection of code components. Here, based on class name and method name MyString and Compares(String s) being scanned from the repository. The best cases were selected with the similarity percentage as shown above table. While finding the similar case, user has provided a threshold selection for retrieval. In this case, its default setting scanned 1200 cases.

5. Feedback Module & Evaluation
   After retrieving a candidate case the algorithm provides the user with feedback about what it has carried out. The feedback section is an evaluation tool for checking the performance of the algorithm. The evaluation module is responsible for the adoption of the solution if it is new one. It compares the case parameters; accordingly it will provide the proper justification for the result. It retrieves the no. of attempts exempted for the perfect match as well failure rate also.

Pseudo Code:

```
CaseDescriptor (String File)
- Input <*.java> file located in any drive
- Identify the possible modifier: get_modifier()
- Extract the file content into case descriptor: retrieval_Code()
- Apply hash index on code components: GetHashData()
- Store in database
similarityMeasures(String argument list)
- Input parameter
- Apply threshold on selection
- Set k value
- Apply k – nearest neighbor for searching
- Import java classifiers: KNN()
- Read file using BufferedReader technique
- Select the appropriate case or use default selection
feedbackEvaluation(case)
- Check the number of similarity measures exists from the repository
- Justification of retrieval case in the terms of values
- Check the performance measure in batch
```

Table Structure

Files (fileID, filename)
Package (packageID, packageName, importedPackage, codeContent, fileID)
Classes (classID, className, inheritedClass, inheritedFrom, implementedFrom, implementedTo, codeContent, fileID)
Interface (interfaceID, interfaceName, inheritedClass, codeContent, fileID)
Methods (methodID, methodName, argumentType, argumentName, returnValue, codeContent, fileID)

Applicability of the Proposed Work

The applicability of this algorithm is to supports java reflection. Most of the programmer are used the classes of code component and the entire file or code the program. None of the programmer used the packages and variables so far and even parameters too. The programmers interested in adopting the interfaces and some kind of methods. The performance of the system depended entirely upon the nature of the case library. The work covered the following aspect that governs:

- It helps the developer to locate the code for the potential reuse.
- It gives a solution strategy based on retrieval of cases and its usage.

Conclusion

The researcher tests the algorithm for a set training data. Those will be selected from several different sources. The .java files downloaded from the web. After training the case-base with each of these training sets in turn we tested it with a test set also taken from real class libraries samples of object oriented programming. It selects the appropriate descriptors of the inputted cases - .java files. It will separate the code components automatically and stores into the repository. The different levels of ambiguity in selection of cases are controlled through data preprocessing technique of data mining. The set of adjustments applied to get the similarity of the code components. In case of packages, the conversion mechanism adopted to get the built in classes files and retrieve the relevant dependencies of the cases. The effectiveness is evaluated on the basis of success factors of the possible retrievals. It includes the successful cases being used for adaption for further use. The successful no. of terms retrieved based on selected code component. The researcher has put the further work to be revised on packages of java files.

References

1. J. L. Kolodner, "An Introduction to Case-Based Reasoning," in *Artificial Intelligence Review*, Atlanta, GA, College of Computing, Georgia Institute of Technology, 1992, pp. 3--34.

2. S. I. Morisbak, "The Road to ASCRARAD: The Development of Agent Support for a Case-based Reuse Application for Rapid Application Development," June 22, 2000.