Vol.2 Issue.3, Pg.: 70-74
March 2014

INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

IDENTIFICATION OF CODE CLONES FOR APPLICATION OF EXTRACT CLASS REFACTORINGS IN OBJECT-ORIENTED SYSTEMS

Suchithra Chandran[1], Bright Gee Varghese. R[2]

[1] PG Student, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India, suchithracc@gmail.com
[2] Assistant Professor, Department of Computer Science and Engineering, Karunya University, Tamilnadu, India, brightfsona@yahoo.co.in

Abstract

Code clones or simply code duplication in a source code are fragments of code that are similar or identical. These code clones are one of the factors that degrade the design and quality of software. Major reason for the occurrence of code clone is code reuse by copying the existing program code. Code reuse takes place when one adds a new functionality to an existing system during maintenance. There are two types of clone pairs: in-module clone pair and inter-module clone pair. Inter-module clones increase the functional coupling between the modules, while in-module clones do not affect the strength of coupling between the modules. With code duplication the maintenance costs are much higher. In this paper a code clone detection method is proposed before performing the extract class refactoring. The code clone detection method includes three steps: Lexical analysis, Transformation, Match detection and formatting. The source code of the software system is performed with code clone detection before refactoring, which improves the system design. Refactorings are commonly used to improve software quality after a significant software development or evolution. By performing code clone detection before refactoring, it can identify far better refactoring opportunities.

Keywords*:* Code clone, Refactoring, Coupling, Lexical analysis, Transformation.

1. Introduction

Code cloning is the act of copying fragments of codes for making non-functional alterations to the software system. Source code cloning occurs when a developer reuses the existing code in a new context by making a copy that is altered to provide new functionality. This becomes a well-known problem to the evolving software system. Code clones also known as duplicate codes are malicious code smells that are to be removed for better maintainability of the software system. Code duplication is generally a poor programming style. In large systems, code cloning method becomes a standard way to produce variant modules.

Concerning with the detection of duplicated code, several techniques have been successfully applied on software systems. These techniques are classified into three categories: string-based [1], token-based, and parse-tree based [2]. A systematic technique for reducing code clones of object-oriented programs is also proposed [3]. Usually this type of cloning is found in the absence of good re-use development processes. A large number of software clones induces several side effects in the software system. The problem includes increase in the resources required by the software on the system. This also increases the cost of operation in the software system.

The main difficulty in detecting clones in a large scale system size. Due to the system's size, it is difficult to manually track down the clones. Usually no documentation is kept on cloning activities. Clone identification has great effort in the maintenance and re-engineering of legacy systems. Refactoring essentially means improving the design after it has been implemented. Refactoring [4] is the process of introducing behavior preserving restructurings to the code, in order to improve its design and enable it to support further development.

2. Proposed method

The proposed token-based method [6] of code clone detection involves three steps: (a) Lexical analysis, (b) Transformation, (c) Match detection and formatting. The token-based clone detection method [5][6] is applicable to a million-line code with affordable computation time and memory usage.

2.1. Lexical Analysis

Token based techniques use transformation algorithm by constructing a token stream from the source code, hence require a lexer [6]. Lexical analysis is the process of converting a sequence of characters into a sequence of tokens, i. e. meaningful character strings. A program that performs lexical analysis is called lexical analyzer or lexer. Here all the source files are divided into tokens based on lexical rules of the programming language (Figure 2). White spaces and comments are ignored in this analysis.

2.2. Transformation

The sequence of tokens is transformed, i.e., tokens are added, removed, or changed based on the transformation rules [6]. Then, each identifier related to types, variables, and constants are replaced with a special token (Figure 3). This replacement makes code fragments including different variable names as clone pairs. Transformation helps to identify the similar lines of codes in different modules.

2.3. Match detection and formatting

From all the sub-strings on the transformed token sequence, equivalent pairs are detected as clone pairs. Then, each location of clone pair is converted into line number on the original source files (Figure 4). This number is used as a reference in all other clone pairs.

2.4. Extract Class Refactoring

Refactoring [3] is the process of introducing behaviour preserving restructurings to the existing code to improve the design quality of the system. Here extract class refactoring [4] is performed which simplifies the large, complex classes and less cohesive classes. Extract class refactoring is performed by three steps: identification of extract class refactoring opportunities, ranking of the identified refactoring opportunities based on the impact they bring to the system, application of extract class refactoring.

```
…..
a=0;
b=a-c;
If(b>0) n=1;
c=0;
…..
```

Module 1

```
…..
}
b=x-y;
if(b>0) n=0;
While(x>0) {
…..
```

Module 2

```
…..
a = 0 ;
b = a – c ;
if ( b > 0 ) n = 1 ;
c = 0 ;
…..
```

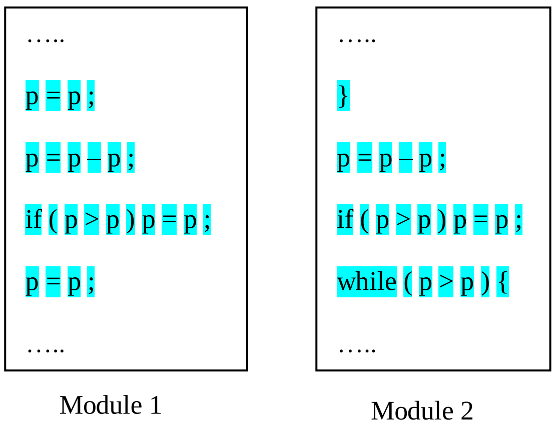
Module 1

```
…..
}
b = x – y ;
if ( b > 0 ) n = 0 ;
while ( x > 0 ) {
…..
```

Module 2

Figure 1: Original Source Code

Figure 2: Lexical Analysis



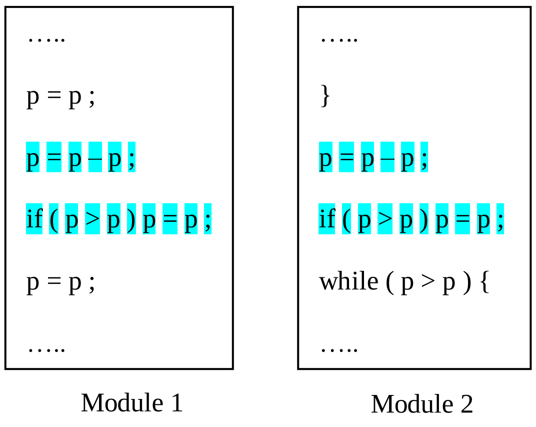
```
…..
p = p ;
p = p – p ;
if ( p > p ) p = p ;
p = p ;
…..
```

Module 1

```
…..
}
p = p – p ;
if ( p > p ) p = p ;
while ( p > p ) {
…..
```

Module 2

Figure 3: Transformation



```
…..
p = p ;
p = p – p ;
if ( p > p ) p = p ;
p = p ;
…..
```

Module 1

```
…..
}
p = p – p ;
if ( p > p ) p = p ;
while ( p > p ) {
…..
```

Module 2

Figure 4: Match detection and formatting

The identification of refactoring opportunities consists of two steps: First each class is analyzed to extract dependency information among the class members, then clustering algorithm uses this information to identify the cohesive classes that can be extracted as separate classes. Secondly the classes identified are filtered by applying a set of rules to evaluate whether they have sufficient functionality and whether the suggested refactoring preserves the behaviour of original system. The clustering algorithm used here is agglomerative clustering algorithm [4] and the distance metric used for clustering is Jaccard distance.

The Jaccard distance between two entities α and β with entity set A and B in a class is calculated as follows:

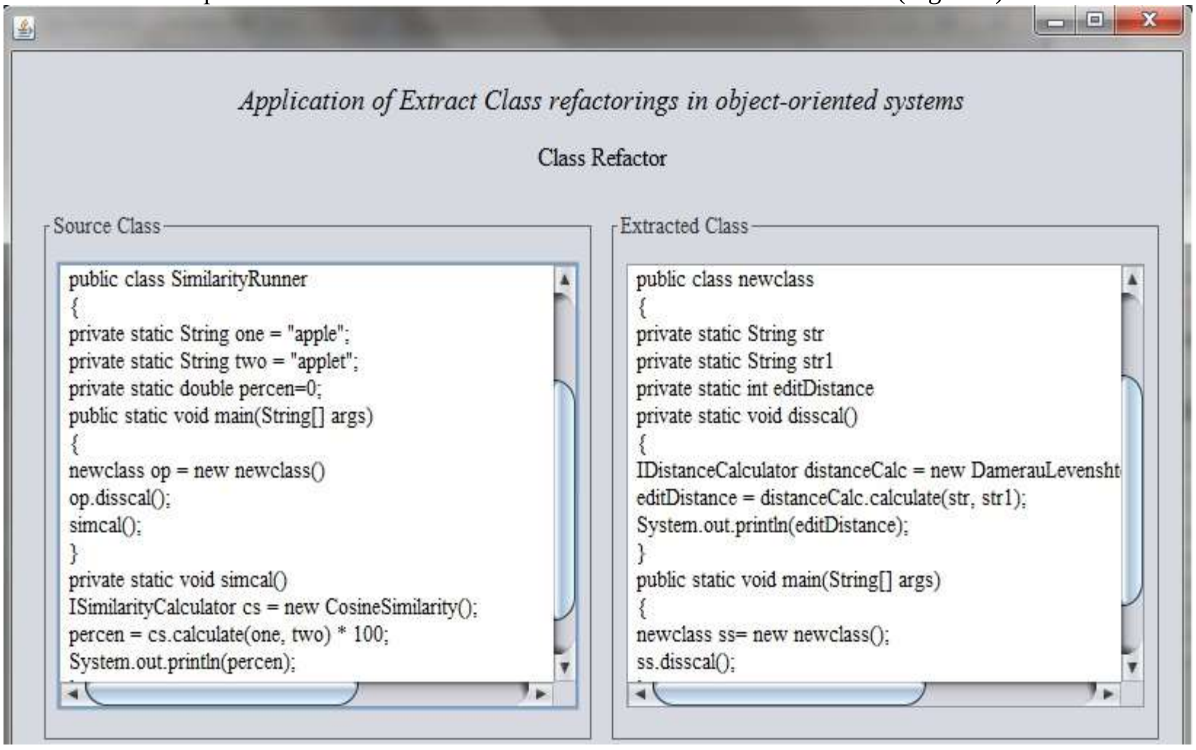
$$d_{\alpha,\beta} = 1 - \frac{|A \cap B|}{|A \cup B|} \qquad (1)$$

Ranking of the identified refactoring opportunities is done based on the impact they bring to the system. To estimate the expected design improvement for the suggested refactoring, we calculate the entity placement value for each class. The lower the entity placement value, the better the resulting design. It calculates the distances of the entities belonging to a class from the class itself(cohesion) divided by the distances of the entities not belonging to the class from the class itself(coupling)[4].

The Entity Placement value for a class C (EP_C) is the ratio of its average distance from the entities that belong to class C to its average distance from the entities that do not belong to the class.

$$\mathrm{EP_C} = \frac{\sum_{e_i \in C} \mathrm{distance}(e_i, C) \big/ |\mathrm{entities} \in C|}{\sum_{e_i \notin C} \mathrm{distance}(e_i, C) \big/ |\mathrm{entities} \notin C|} \qquad (2)$$

During the application of extract class refactoring, first an empty class entity set is created. For each extracted entity, its origin class is changed from the source class to the new class. The entity sets of all the entities that access or are accessed by the extracted entities are updated. The extracted entities are inserted in the entity set of the new class. The extracted entities are removed from the entity set of the source class. Import the necessary header files and class declarations for the new class. Then the variables and methods that are to be extracted are imported to the new class. Then it's removed from the source class (Figure 5).


Application of Extract Class refactorings in object-oriented systems
Class Refactor
Source Class
Extracted Class

```
public class SimilarityRunner
{
private static String one = "apple";
private static String two = "applet";
private static double percen=0;
public static void main(String[] args)
{
newclass op = new newclass()
op.disscal();
simcal();
}
private static void simcal()
ISimilarityCalculator cs = new CosineSimilarity();
percen = cs.calculate(one, two) * 100;
System.out.println(percen);
```

```
public class newclass
{
private static String str
private static String str1
private static int editDistance
private static void disscal()
{
IDistanceCalculator distanceCalc = new DamerauLevensht
editDistance = distanceCalc.calculate(str, str1);
System.out.println(editDistance);
}
public static void main(String[] args)
{
newclass ss= new newclass();
ss.disscal();
```

Figure 5: Source class and Extracted class

3. Result

The result of the application of extract class refactoring along with code clone detection will develop a well modularized system with high cohesion and low coupling. The proposed method improves the design quality of the system. The identification of code clones also reduces the code complexity and the number of code lines. The clustering method used for the identification of extract class opportunities identifies better refactoring opportunities. It also identifies far better refactoring opportunities by performing code clone detection. The impact of the performed refactoring is measured in terms of coupling and cohesion using the message-passing coupling (MPC) metric. The MPC for a class C is defined as the number of invocations of methods not implemented in class C by the methods of class C. The value of this metric is compared to Entity placement, which is based on Jaccard distance that calculates coupling and cohesion.

4. Conclusion

In this work, code clone detection identifies the similar code in the software system which degrades the quality of the software. The identified code clones are transformed into special token and all the clone pairs are replaced with this special character. By performing this transformation the code complexity of the source code is reduced. After identifying this clone pairs extract class refactoring is performed. Extract class refactoring is performed by restructuring the code by making the low cohesive classes to high cohesive ones by decomposition of the original class, thereby improving the quality of software design.

REFERENCES

[1] Van Rysselberghe. F, Demeyer. S, "Evaluating clone detection techniques from a refactoring perspective", In Proceedings of the 19th International Conference on Automated Software Engineering, pp 336-339, 2004.

[2] Baxter. I. D, Yahin. A, Moura. L, "Clone detection using abstract syntax trees", In Proceedings of International Conference on Software Maintenance, pp 368-377, 1998.

[3] Fowler. M, Beck. K, Brant. J, Opdyke. W, and Roberts. D, "Refactoring: Improving the design of existing code," Addison-Wesley, 1999.

[4] Fokaefs,M., Tsantalis,N., Stroulia,E., Chatzigeorgiou,A., "Identification and Application of Extract Class refactorings in object-oriented systems," Journal of Systems and Software, vol.85, issue 10, pp. 2241-2260, 2012.

[5] Kamiya. T, Kusumoto. S, and Inoue. K, "A token-based code clone detection technique and its evaluation," Technical Report of IEICE (The Institute of Electronics, Information and Communication Engineers), Vol. 100, No. 570, pp. 41-48, Jan. 2001.

[6] Monden. A, Nakae. D, Sato. S, Matsumoto. K., "Software Quality Analysis by Code Clones in Industrial Legacy Software." In Proceedings of the 8th Symposium on Software Metrics, pp. 87 – 94, 2002.

Biographies

Suchithra Chandran is pursuing MTech (Computer Science and Engineering) degree from Karunya University, Tamil Nadu, India. She received her B.E (Computer Science and Engineering) degree from AMS Engineering College, Tamil Nadu, India.

Bright Gee Varghese. R completed his B.E (Computer Science and Engineering) from Sun College of Engineering and Technology, Nagercoil, Tamil Nadu, India in 2003. He started his career as Lecturer in Sun College of Engineering and Technology, Nagercoil, from June 2003. He completed his M.E from Vinayaka Mission university, Salem, Tamil Nadu in 2011.Currently he is working as an Assistant Professor in Computer Science Department in Karunya University and pursuing part time Ph.D in Karunya University, Tamil Nadu, India. His main research area is Software Engineering.