JCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.-6, Issue-9, Sept. 2018 E-ISSN: 2347-2693

# Multilevel Code Cleaning using Root Extract Method for Java Programs

**Pooja Kapila[1*], A. Sharma[2], N. Kaur[3]**

[1]Dept. of Information Technology, Chandigarh Engineering College, Mohali, India
[2]Dept. of Information Technology, Chandigarh Engineering College, Mohali, India
[3]Dept. of Information Technology, Chandigarh Engineering College, Mohali, India

**Available online at: www.ijcseonline.org**

Accepted: 25/Sept/2018, Published: 30/Sept/2018

***Abstract***—The code cleaning requires the incorporation of the various processes to remove the clones from the source code as well as the programming irregularities, which improve the overall design of the code. In this paper, the proposed model has been designed for the purpose of code cleaning by using the multi-factor code cleaning algorithm. The proposed model is entirely based upon the elimination of the source code irregularities, which contains the bad smells, code clones and other such problems. The proposed model is designed to work in the three primary components, which includes the code clone and smell detection and marking algorithm, which is followed by the refactoring method estimation and then the application of the refactoring application in the final phase for the act of cleaning the source code. The proposed model utilizes the divide and conquer method, which is concerned with the extraction of the methods from the class files. Also the proposed model analyzed and extracts the independent statements from the extracted methods, which incorporates the common statement elimination, which removes the common statements from the duplication removal process. The proposed model has been designed to refactor the code on the basis of the bad smell detection and elimination with the appropriate method. The proposed model has been analyzed under the various kinds of the datasets for the experimental evaluation, where it has been found better. The proposed model has been recorded with the significant values of the parameters of the accuracy, precision and recall.

***Keywords***— Code clone detection, Code cleaning, Duplication detection, Overlapping shifting method.

## I. INTRODUCTION

The proposed model has been designed for the removal of the clones and other programming issues from the JAVA source code. The proposed model has been designed within the layered architecture to perform the various operations in the multiple stages. The various objectives of the proposed system are mentioned and described in the full detail in the experimental design section.
The proposed model has been undergone the development in the phases, which aims at performing the individual tasks over the input source code. The proposed model has been primarily divided in the four major parts, which can be defined with the following flowchart describing the layered architecture of the clone detection and refactoring model.

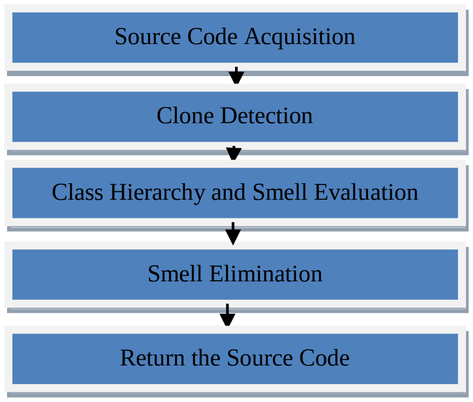


Figure 1: Workflow of the Proposed Model

The proposed model can easily described in the major sections as per shown in the Fig 1. The proposed model has been divided in the major modules as per the following listings along with their description.

## II. LITERATURE REVIEW

M. F. Zibran et. al. [24] has outlined the road to software clone management: a survey. With regards to the scheduling techniques, the evolutionary algorithms like GA similarly because the artificial intelligence (AI) techniques like heuristic based mostly approaches could suffer from local optima, and do not guarantee optimality. M, O'Keeleet. et. al. conducted associate degree empirical comparison of simulated annealing (SA), GA and multiple ascent hill-climbing techniques in scheduling refactoring activities in five software systems written in Java. They minimize that among those AI techniques, the hill-climbing approach performed the most effective. M. F. Zibran et. al. [25] has worked an on conflict-aware optimal scheduling of inimized code clone refactoring. Among few potential refactoring open doors, the determination and request of gathering of refactoring activities could have distinguishable effect on the design/code quality measured regarding software system metrics. Additionally, there are also dependencies and conflicts among those refactorings of different needs. Tending to all of the conflicts, priorities and dependencies, a manual plan of an ideal refactoring timetable is extremely high priced, if not outlandish. In this way an automated refactoring scheduler is vital to maximize benefit and inimize refactoring efforts. Nonetheless, the estimation of the efforts expected to perform code clone refactoring could be a troublesome task. T. Mens et. al. [15] has developed the A Survey of software refactoring. This paper gives a broad outline existing examination in the field of software refactoring. This exploration is compared and discussed in light of various criteria: the refactoring activities that are supported, the particular methods and formalisms that are utilized for supporting these activities, the types of software artifacts that are being refactored, the essential issues that need to be taken into account when building refactoring tool support, and the impact of refactoring on the software process. E. Murphy-Hill et. al. [14] has outlined the why don't people use refactoring tools? Tools that perform refactoring are at present under-used by software engineers. As more progressed refactoring tools are outlined, an incredible great argument between how the tools must be utilized and how programmers need to utilize them. In this position paper, we portray the predominant procedure of refactoring; shows the numerous research tools don't support this procedure, and start a suggestion to take action for originators of future refactoring tools. E. Kodhai et. al. [12] has surveyed that Method-Level Code Clone Modification using Refactoring Techniques for Clone Maintenance. Researchers concentrated on exercises for example, clone support to help the programmers. Refactoring is a surely understood procedure to enhance the maintainability of the software. Program refactoring is a method to improve readability, structure, execution, abstraction, maintainability, or different characteristics by changing a program. This paper adds to more brought together approach for the phases of clone maintenance with an emphasis on clone modification.

## III. EXPERIMENTAL DESIGN

Extract Method: Extract method is a forming strategy inside the refactoring techniques. During this extract technique, these take a group of codes and switch it into its own method. Additionally, flip the fragment into a method whose name clarifies the point of the method [4].

e.g.:

```
void printOwing (double amount) {
printBanner ( );
//print details
System.out.println (“name:” + _name);
System.out.println (“amount” + amount);
}
```

Becomes:

```
void printOwing (double amount) {
printBanner( );
printDetails(amount);
}
void printDetails (double amount) {
System.out.println (“name:” + _name);
System.out.println (“amount” + amount);
}
```

Inline Method: A method's body is pretty much as clear as its name. Put the method's body into the body of its callers and expel the method [4]. For example, check the following code segment:

```
int getRating ( ) {
        return (moreThanFiveLateDeliveries ( ) ) ? 2 : 1;
}
458inimiz moreThanFiveLateDeliveries ( ) {
return _numberOfLateDeliveries > 5;
}
```

Become:

```
int getRating ( ) {
                return (_numberOfLateDeliveries > 5) ?
2 : 1;
}
```

Move Method: A method is, or will utilize, or utilized by extra features of another class than the class on which it is characterized [4]. Make a replacement with a same body within the class it utilizes most. Either transform the old method into a simple delegation, or take away it inside and out.

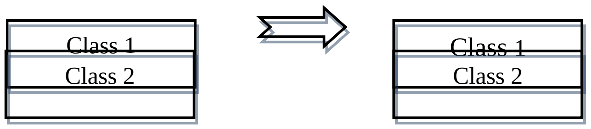


Figure 2: Move Method

e.g.:

```
class Project {
        Person[ ] participants;
}
class Person {
        int id;
        458inimiz participate (Project p) {
                for(int i=0; i<p.participants.length; i++)
{
                 if (p.participants[i].id == id)
return(true);
                        }
        return (false);
                }
        }
.. if (x.participate(p)) ...
```

After applying the Move Method

```
class Project {
        Person[ ] participants;
        459inimiz participate(Person x) {
                for(int i=0; i<participants.length; i++) {
                if      (participants[i].id     ==      x.id)
return(true);
                }
                return (false);
        }
}
class Person {
int id;
}
... if (p.participate (x)) ...
```

Pull Up Method: At the point when copied code crosswise over two separate classes then the best refactoring technique to implement is to pull (Drag) that duplicate code up into a super class thus we tend to DRY (Don't Repeat Yourself) out the code and allow it to be used in different places while not duplication (which means changes in future just need to happen in one place) [4]. Fundamentally, Pull Up Method is managing the Generalization. Generalization creates its own batch of refactoring, basically managing moving methods around a hierarchy of inheritance. Pull Up Field and Pull Up Method both promote function up hierarchy. There are lots of procedures that are utilized with Generalization [4].

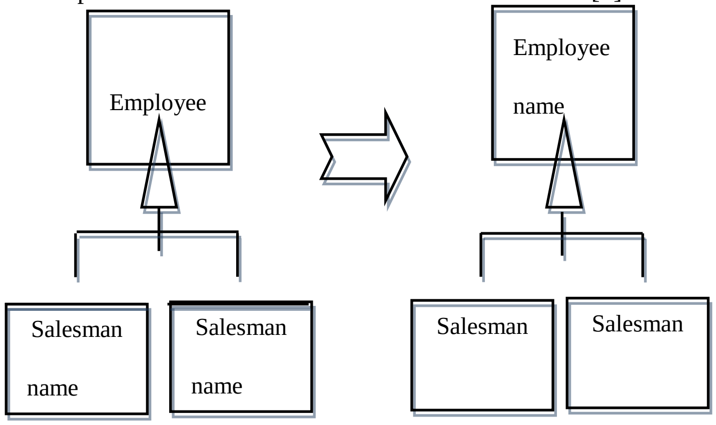


Figure 3: Pull Up Method

Push Down Method: Push down method is defines as behavior on a superclass is relevant only for some of its subclasses. Push Down Method and Push Down Field push function downward.

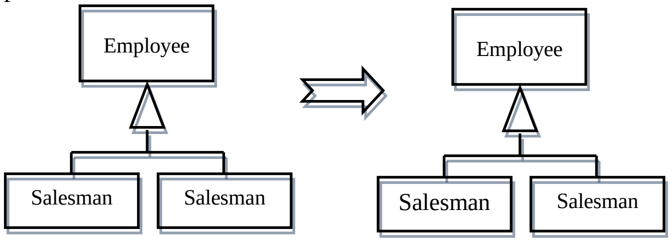
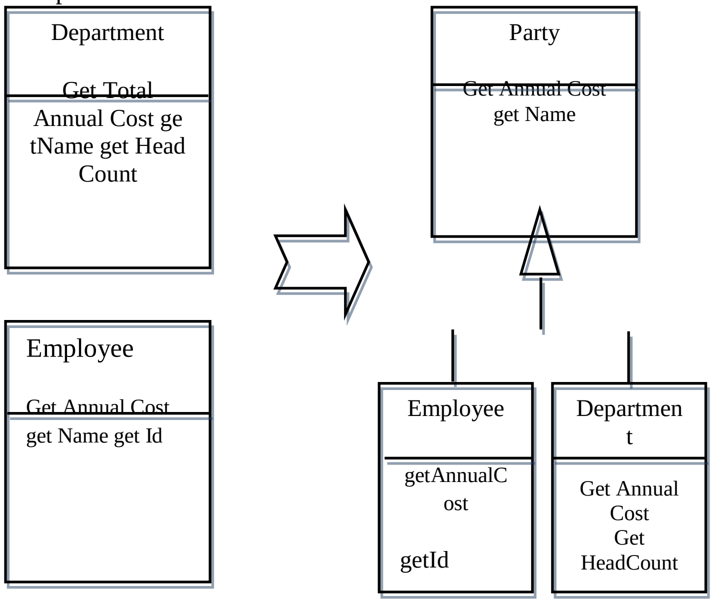


Figure 4: Push Down Method

Extract Superclass Method: In Extract Superclass Method, Create a superclass and move the common features to the superclass.


Department
Get Total Annual Cost ge tName get Head Count
Employee
Get Annual Cost get Name get Id
Party
Get Annual Cost get Name
Employee
getAnnualCost
getId
Department
Get Annual Cost Get HeadCount
Figure 5: Extract Superclass Method

Introduce Explaining Variable and Rename Methods: In these two methods, place the results of the style, or portions of the expression, in a very temporary variable with a name that clarifies the point [4].
e.g.:

```
if ( (platform.toUpperCase ( ) .indexOf ("MAC") > - 1)
&&
(browser.toUpperCase ( ) .indexOf ("IE") > -1) &&
wasInitialized ( ) && resize > 0 )
{
// do something
}
```

Become:

```
final booleanisMacOs = platform.toUpperCase (
).indexOf ("MAC") >-1;
final booleanisIEBrowser = browser.toUpperCase (
).indexOf ("IE") >-1;
final booleanwasResized = resize > 0;
if (isMacOs && isIEBrowser && wasInitialized() &&
wasResized) {
// do something
}
```

## IV. RESULT ANALYSIS

The proposed source code clone analyzer has been deeply analyzed for its performance over the various source code segments. The proposed model utilizes the pattern matching method for the detection of the code clone in the given source code. The rules are predefined in the training data, which are further analyzed for the purpose of the code clone detection in the given source code. The clones have been detected by using the iterative approach over all of the code segments which successfully processed under the feature description algorithm.

Table 1: The Results Obtained From the Source Code Analyzer over the Given Source Code Files

| Program | Correctly Detected Clones | False Positives | False Negatives |
|---|---|---|---|
| Code1.JAVA | 2 | 0 | 1 |
| Code2.JAVA | 1 | 0 | 0 |
| Code3.JAVA | 1 | 1 | 0 |
| Code4.JAVA | 0 | 0 | 1 |
| Code5.JAVA | 3 | 1 | 1 |
| Total | 7 | 2 | 3 |

Total of five files of the JAVA source code has been used for the testing the code clone code analyzer. The proposed model has been found better with the few testing, where it has also failed to detect and analyze some of the source code. The proposed model has been evaluated for its performance over the given JAVA dataset. The proposed model has been evaluated for the multi-disciplinary clones by using the pattern matching and classification. The proposed model has been recorded with the moderately higher accuracy because of the higher level of false positive and false negative cases.

Table 2: The Accuracy Based Evaluation of the Proposed Model

| | |
|---|---|
| Accuracy | 58.33 % |
| Precision | 77.78 % |
| Recall | 70.00 % |

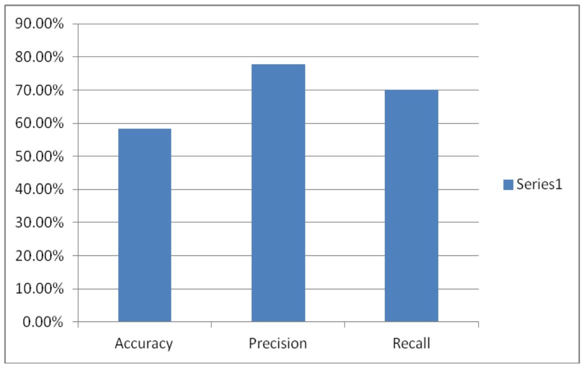


Figure 6: Graph Showing Accuracy of the Proposed Model

The overall accuracy of the proposed model is 58.33% recorded under this performance evaluation study. The proposed model recall rate has been recorded at 70% and the precision at 77.78%. The proposed model can be improved by using the more robust pattern recognition with the highly dense pattern training data for the code clone detection. Also the classification method can be further improved from the non-probabilistic to the probabilistic classifier for the code clone detection.

## V. CONCLUSION

The proposed model has been designed for the code clone detection in the method level and statement level evaluation. The proposed model has been designed by using the divide and conquer method, which is responsible for the method level extraction by estimating the delimiting characters for the function definitions. The code clone estimation is performed in the dual behavior, where the dual level detection method includes the detection of the code clones in the inter-class method and intra-class fashion. The proposed model also evaluates the other forms of the source code smells by analyzing the source code. The various kinds of the experiments over the source code evaluation and then apply the bad smell elimination, which has been tested over the variety of the testing sets. The proposed model has been found efficient on the basis of all of the evaluations over the acquired datasets. In the future, the code cloning can be detected by utilizing the code crawlers for the in-depth analytical application. The proposed model can be also extended by using the swarm intelligent algorithm based solution for the optimization and assessment of the code clones and the bad smells.

## REFERENCES

[1] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
[2] W. G. Griswold, W. F. Opdyke, The Birth of Refactoring: A retrospective on the Nature of High-Impact Software Engineering Research, IEEE Software, 32 (6) (2015), 30-38.
[3] R. Geiger, Evolution Impact of Code Clones, Diploma Thesis, University of Zurich, October, 2005.
[4] M. Hafiz, J. Overbey, Refactoring Myths, IEEE Software 32 (6) (2015), 39-43.
[5] ToolIclonehttp://www.softwareclones.org/iclones.php>(accessed December 2015).
[6] ToolJcdhttp://www.swag.uwaterloo.ca/jcd/> (accessed December 2015).
[7] R. Koschke, I. D. Baxter, M. Conradt, J. R. Cordy, Software Clone Management Towards Industrial Application, Dagstuhl Seminar 12071 on 2 (2) (2012) 21-57.
[8] M. Kim, Z. Thomas, N. Nachiappan, An Empirical Study of Refactoring Challenges and Benefits at Microsoft, Software Engineering, IEEE Transactions on 40, (7 ) (2014) 633-649.
[9] E. Kodhai, S. Kanmani, Method-Level Code Clone Modification using Refactoring Techniques for Clone Maintenance, Advanced Computing: An International Journal (ACIJ), 4 (2), March 2013, 7-26.
[10] E. Murphy-Hill, C. Parnin, A.P. Black, Refactoring Tools: Fitness for Purpose, IEEE Transactions on Software Engineering, 25, (5), (2008) 38-44.
[11] E. Murphy-Hill, A. P. Black, Why Don't People Use Refactoring Tools?, in proceedings of the Computer Science

Faculty Publications and Presentations, Portland State University, Portland, (2007), Paper 115.
[12] T. Mens, T. Tourwé, A Survey of Software Refactoring, IEEE Transactions on Software Engineering, 30, (2), (2004) 126-139.
[13] M. O'Keeffe, M. Ó, Cinnéide, Search-based refactoring: an empirical study, J. Softw.Maint.Evol, Res. Pract., 20, (2008), 345–364.
[14] C.K. Roy, J.R. Cordy, A Survey on Software Clone Detection Research, Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, 2007, p.115.
[15] C.K Roy, M.F Zibran, R. Koschke, The Vision of Software Clone Management: Past, Present, and Future (keynote paper), in: proceeding of the Software Maintenance and Reengineering and Reverse Engineering, Antwerp, Belgium, (CSMR-WCRE), 2014, pp. 18-33.
[16] D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, Information and Software Technology 55 (7) (2013) 1165-1199.
[17] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. D. Penta, A. D. Lucia, D. Poshyvanyk, When and Why Your Code Starts to Smell Bad, in: Proceedings of the 37th International Conference on Software Engineering, Florence, Italy, 2015, pp. 403-414.
[18] N. Tsantalis, D. Mazinanian, G. P. Krishnan, Assessing the Refactorability of Software Clones, Software Engineering, IEEE Transactions on 41 (11) (2015) 1055-1090.
[19] M. D Wit, A. Zaidman, A. V. Deursen, Managing Code Clones Using Dynamic Change Tracking and Resolution, In: proceeding of the ICSM, 2009 pp. 169–178.
[20] M. D Wit, Managing Clones Using Dynamic Change Tracking and Resolution, Master's thesis, Software Engineering Research Group, Delft University of Technology, 2009.
[21] M. F. Zibran, C. K. Roy, The Road to Software Clone Management: A survey, Technical Report 2012–03, The University of Saskatchewan, Canada, Feb., 2012 p. 54.
[22] M. F. Zibran, Chanchal Kumar Roy, Conflict-aware optimal scheduling of prioritised code clone refactoring, IET Software, on 7 (3) (June, 2013) 167-186.

**Authors Profile**

Pooja Kapila received the B.TECH degree in information technology from Chandigarh engineering college in mohali, Punjab, in 2015.She is currently pursuing the MTECH in information technology with the Chandigarh engineering college mohali, Punjab. Her research interests include software engineering.

Mr. Amitabh Sharma has a total experience of 20 years in the field of teaching. He has been working as Assistant Professor at Chandigarh Engineering College since 2006. He has so far published around 50 papers in various International and National Scopus journals

Assistant Prof. Navleen Kaur is working in the field of education for last 5 years and has been offering her services in Chandigarh Group of Colleges, Landran since June 2013 in the Department of Information Technology (IT) at Chandigarh Engineering College. She has so far published around 15 papers in various International and National Scopus journals. Her research work is in the field of software engineering in Test Case Prioritization