WORKFLOW MAP BASED CLONE DETECTION IN THE OBJECTIVE ORIENTED PROGRAMMING

Pooja Kapila,
CGC Group of Colleges, Landran

ABSTRACT--The refactoring models contain the various kinds of the operations for detection of the smells within the source code. 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.

Keywords: *Clone management, Clone elimination, Efficient Programming, Refactoring.*

1. INTRODUCTION

A code fragment that has identical or similar code fragment(s) to within the source code, is termed as code clone. Copying code fragments and reusing that by pasting with or without modifications are very common activities in software development. However, in a post-development position, it is very difficult to prove that which code is original and which one is pasted from existing code. This type of re-use approach of existing code is called the **code cloning** and the pasted code fragment is called a clone of the original [17]. The process is called the **software cloning**. We also called the clones are those segments which are used according to some definition of similarity. All the same, in software engineering field, the term code clones is still searching for a suitable definitions. There are also many other software engineering tasks such as understanding code quality, software evolution analysis, aspect mining, plagiarism, etc. Code clones are considered as bad smells of the software system [17]. Moreover, clones are also the result of copy paste activities. Therefore, these activities are very simple and reduce the programming efforts and time as they use an existing code rather than rewriting similar code. Clones are believed to possess a negative impact on evolution. Code clones may affect the software system's quality, especially their maintainability and comprehensibility. More discussion on the reason for cloning can be found elsewhere [19].

Copy code is a computer programming term for an arrangement of source code that happens more than once, either inside a program or crosswise over various programs owned or kept by the same entity. Duplicate code is for the most part viewed as undesirable for various reasons. A minimum requirement is usually applied to the quantity of code that must show up in a sequence for it to be considered duplicate rather than coincidentally similar. Sequences of duplicate code are sometimes known as code clones or just clones; the automated procedure of discovering duplications in source code is called clone detection [17].

2. LITERATURE REVIEW

N. Tsantalis et. al. [21] has worked on assessing the refactorability of software clones. The presence of duplicated code in software systems is important and several other studies have shown that clones will be probably harmful with relevancy to the maintainability and evolution of the source code. Even with the significance of the matter, there's still finite support for eliminating software system clones through refactoring, as a result of the unification and merging of duplicated code may be very challenging problem, particularly once software system clones responded to many modifications when their initial introduction. C. K. Roy et. al. [18] has worked on studying the vision of software clone management: past, present, and future (Keynote Paper). This paper presents a comprehensive survey on the state of the art in clone management, with in-depth examination of clone management activities (e.g., tracing, refactoring, cost benefit analysis) beyond the detection and analysis. This can be the primary survey on clone management, wherever we tend to purpose of the achievements to this point, and reveal avenues for more research necessary towards an integrated clone management system. D. Rattan et al. [19] has surveyed on software clone detection: a systematic review of software clones and clone

detection and identify the need to develop model and semantic clone detection technique. Limitation of this study is multitude of meaning of the keyword 'clone'. Strings were manually searched to increase number of searches and manual searches may miss relevant articles. M. Tufano et. al. [20] has surveyed that when and why your code starts to smell bad. There are many factors that contribute to technical debt. One in all these is described by code bad smells, i.e., symptoms of poor design and implementation selections. Whereas the repercussions of smells on code quality are through empirically observation assessed, there is still solely anecdotal evidence on when and why bad smells are introduced. M. Kim et. al. [11] has worked on an empirical study of refactoring challenges and benefits at Microsoft. This paper presents a field study of refactoring advantages and challenges at Microsoft through three complementary study methods: a survey, semi-structured interviews with professional software engineers, and quantitative analysis of version history data. Our survey finds that the refactoring definition in practice is not confined to a rigorous definition of semantics-preserving code transformations which developers understand that refactoring involves substantial price and risks.

3. EXPERIMENTAL DESIGN

Clone management-is a cross cutting and an umbrella activity. Basically, the clone management is the set of activities. There are many activities in clone management like clone refactoring, classification, tracking, visualization and evolution. In current status, some benefits of clone management [24]:

- It improves customer satisfaction and software quality.
- Clone awareness with visualization and during the debugging and modification. It helps to the developers.
- Modified clone regions are notified to the developers.
- Cloning patterns are identified, by this quality of software is improved.

To manage clone, first of all they have to be identified. The workflow for clone management system is summarized in the Figure 1. The end part of clone detection created the clone documentation that saves the location of code segment and their relationship [19]. If the copied code changes due to ongoing development, the changes and locations of clones need to be tracked and the documentation need to be uploaded accordingly. The clone documentation may be analyzed to determine justification of clone to find potential clones for removal. Visualization techniques can aid to find the potential clones for erase. Further, clones are documented or annotated. Upon the application of refactoring operations, a follow up verification may examine if the refactoring caused any change in program. Again documentation is updated after the refactoring [24].

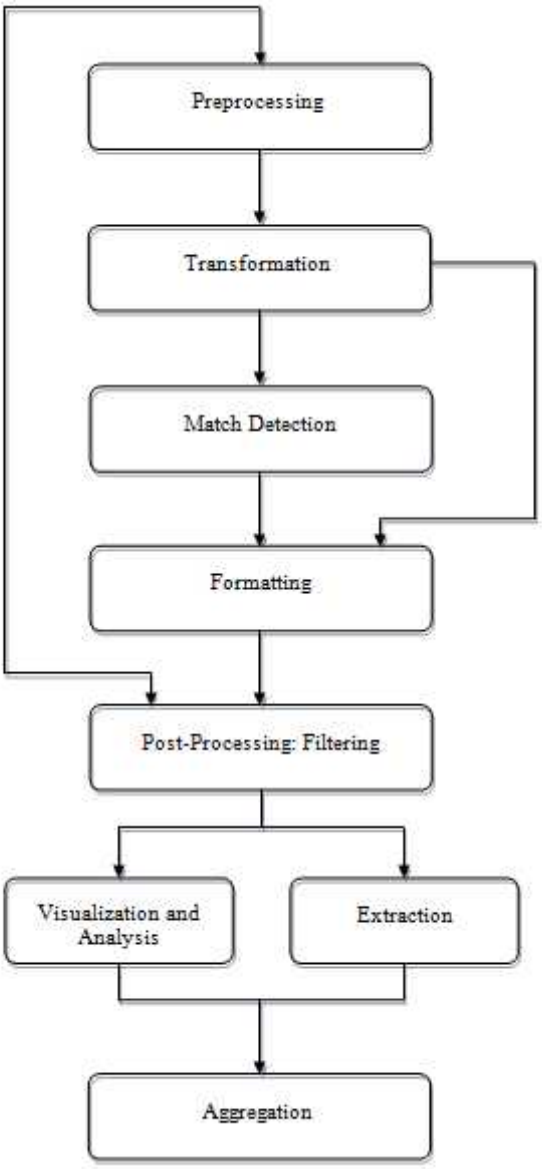

Figure 0: Clone Management Workflow [24]

The clone detection method works on the basis of the divide and conquer method, where the method level extraction plays the vital role in the detection of the clones. Each class is subdivided into the declared methods, which are further evaluated under the scanner by comparing with the methods extracted from the similar class or the other classes. The clone detection model works on the basis of the method level extraction and line level extraction for the detection of the clones within the source code.

Class Hierarchy and Smell Evaluation: The class hierarchy and smell evaluation carries the three major parts, which can be easily defined as the following:

i. Clone Position Marking
ii. Locate the source code irregularities
iii. Locate the improperly written statements

Each of the above mentioned method is responsible for the realization of the complete system. The proposed model has been designed to eliminate the cloned methods, statements and the bad smells in the source code. This section under the proposed model has been mainly designed for marking of the clone positions, which are detected under the latter section. Additionally, this part of the model is responsible for findings the irregularities and improperly written statements in the source code.

4. RESULT ANALYSIS

The statistical parameters to measure the statistical errors (Type 1 and Type 2) are measured in order to evaluate the overall performance of the proposed model by evaluating the samples by the means of the programming or the manual binary classification. The proposed model evaluation is entirely based upon this statistical 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

rogram	Correctly Detected Clones	False Positives	False Negatives
Code1.JAVA	2	0	1
ode2.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 %

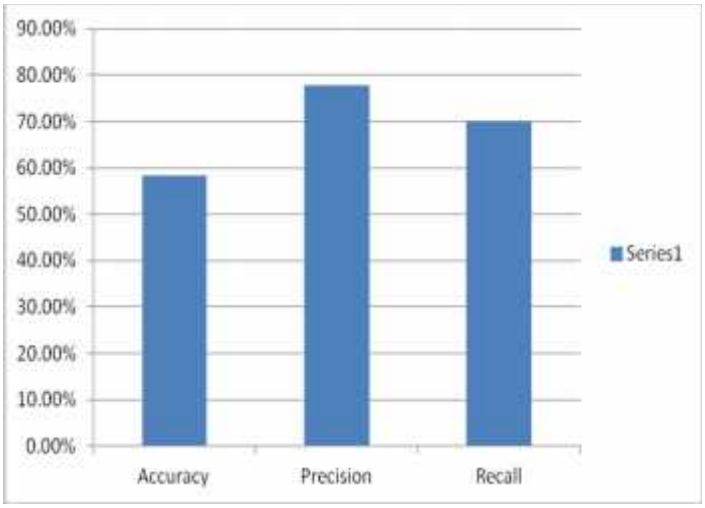

Figure 2: Graph Showing Accuracy of the Proposed Model

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.

5. CONCLUSION

The proposed model has been developed for the code cleaning and refactoring of the source code. 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.

REFERENCES

[1] ToolCloneTracker<http://cs.mcgill.ca/~swevo/clonetracker/> (accessed January 2016).
[2] ToolCloneDr<http://www.semdesigns.com/products/clone/> (accessed December 2015).
[3] ToolCCFinder<http://www.ccfinder.net/ccfinderxos.html> (accessed May 2015).
[4] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
[5] 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.
[6] R. Geiger, Evolution Impact of Code Clones, Diploma Thesis, University of Zurich, October, 2005.
[7] M. Hafiz, J. Overbey, Refactoring Myths, IEEE Software 32 (6) (2015), 39-43.
[8] ToolIclone<http://www.softwareclones.org/iclones.php>
[9] Tool Jcd <http://www.swag.uwaterloo.ca/jcd/> (accessed December 2015).
[10] 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.
[11] 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.
[12] 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.
[13] E. Murphy-Hill, C. Parnin, A.P. Black, Refactoring Tools: Fitness for Purpose, IEEE Transactions on Software Engineering, 25, (5), (2008) 38-44.
[14] 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.
[15] T. Mens, T. Tourwé, A Survey of Software Refactoring, IEEE Transactions on Software Engineering, 30, (2), (2004) 126-139.
[16] M. O'Keeffe, M. Ó, Cinnéide, Search-based refactoring: an empirical study, J. Softw.Maint.Evol, Res. Pract., 20, (2008), 345–364.
[17] 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.
[18] 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.
[19] D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, Information and Software Technology 55 (7) (2013) 1165-1199.
[20] 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.
[21] N. Tsantalis, D. Mazinanian, G. P. Krishnan, Assessing the Refactorability of Software Clones, Software Engineering, IEEE Transactions on 41 (11) (2015) 1055-1090.
[22] 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.
[23] M. D Wit, Managing Clones Using Dynamic Change Tracking and Resolution, Master's thesis, Software Engineering Research Group, Delft University of Technology, 2009.
[24] 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.
[25] M. F. Zibran, Chanchal Kumar Roy, Conflict-aware optimal scheduling of prioritised code clone refactoring, IET Software, on 7 (3) (June, 2013) 167-186.