ISSN No. 0976-5697

Volume 7, No. 6(Special Issue), November 2016

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Adaptive Segmented Block based Clone Detection and Refactoring using Divide and Conquer Method

Charnpreet Singh
Research Scholar, Department of Computer Engineering,
Punjabi University,
Patiala, India.
charnpreet.singh08@gmail.com

Dr. Dhavleesh Rattan
Assistant Professor, Department of Computer Engineering,
Punjabi University,
Patiala, India.
dhavleesh@gmail.com

***ABSTRACT**--The clone detection and refactoring methods are commonly utilized for the defragmentation of the source code of the software systems. The code cleaning is the process associated with writing the clean source code with minimum redundancy to achieve the higher performance. The code clone detection finds the regions with redundancy and the refactoring methods are utilized for the removal of the redundancy from the source code files. In this paper, the major focus has been kept at the clone detection with high accuracy by using the divide and conquer method, which is associated with the segmentation of the code blocks, which is followed by the template matching technique for the feature description from the given source code. The divide and conquer method will be capable to divide the code in the smaller blocks which would undergo the feature extraction for the dilution of the user declarations, which adds the uniqueness to the code segments. The user declarations are converted to the transparent entities by using the standard definition for the entities. The code cloning methods are usually categorized as the blind match detection algorithms, because they extract the training and testing data from the submitted source and do not hold any pre-determined source data. The proposed model will match each and every segment of code against every other segment. The exact matches would be extracted, notified and finally arranged in the cod clone database. The proposed model will track the code clones and perform the refactoring over the detected code clones. The refactoring method will utilize the segment level clone removal from the given source code. The segments extracted and marked under the code cloning methods would be evaluated, and the clones in the higher order classes will be preferred over the child classes. The proposed model is expected to solve the problems associated with the existing models in removal of the clones using the refactoring methods.*

***Keywords:** Adaptive clone detection, refactoring, blind clone detection, code cleaning.*

I. INTRODUCTION

Clone: A code fragment that has exact or similar code fragment(s) to within the source code, in general, terms as code clone. Copying code fragments and these are reuse by pasting with or without modifications are very common activities in software development. This type of re-use approach of existing code is named the **code cloning** and the pasted code fragment is named a clone of the original [9]. The process is called the **software cloning**. We also called the clones are those segments which are used according to some definition of similarity. However, in software engineering field, the term code clones is still checking out an appropriate definitions. Additionally, there are also several different software engineering tasks like understanding plagiarism, code quality software evolution analysis, aspect mining etcetera. Code clones are considered as bad smells of the software system [9]. Moreover, clones are additionally the results of copy paste activities. Clones are believed to possess a negative impact on evolution. More discussion on the reason for cloning can be found elsewhere [11].

Types of Clones:

There are 4 types of the clones. First three are called the "Textual Similarity" and therefore fourth one is understood as "Functional Similarity" [11].

Textual Similarity:

Type I: (Exact clone) here are the clones in that variations in white spaces and comments. **Type II:** (Renamed) program fragments which are structurally similar expects for changes in identifiers, literals, types and comments.

Type III: (Near miss) these are clones that have copied with further modifications like changes in variables , identifiers etcetera.

Functional Similarity:

Type IV: (Semantic clones) programs fragments are functionally similar while not being textually similar.

Clone Detection Techniques:

Various clone detection techniques are conferred within in the literature [9, 10].

Text Based Technique: There's many clone detection techniques that are based mostly on pure text-based strategies. In this, fragments are compared with one different to search out sequence of duplicate text. Text based technique match the text, line by line, with or without

CONFERENCE PAPER
International Conference on
Recent Trends in Computer Science & Information Technology (RTCSIT-2016)
21st August 2016
Guru Nanak College Budhlada, Punjab India

normalization the text by remaining the identifiers, clarify the comments and variations within the layout.

Token Based: In this approach, the all source system is transformed or parsed in to the sequence of tokens. After that, the tokens are scanned for finding the duplicate code and these duplicated sequence returned as clones.

Tree Based: During this approach, the program is split into parsed tree with a parser of the importance of the language. Further, the sub trees are explored with a few tree matching techniques and then the comparable code of the same tree are reported just as clone pairs.

PDG Based: Program Dependency Graph (PDG) is one step further in accruing a source code representation of high abstraction than alternative approaches by considering the semantic information of the source. PDG contains the control flow and data flow information of the program and therefore the code corresponding to the sub graph are identified and represent as derived (copied) code (clones).

Metrics Comparison Based Technique: To detect the function clone, Metrics based mostly technique is employed. There are numerous clone detection techniques that use totally different metrics for realize the similar code. A fingerprinting (a set of software's) functions are work out for syntactic units (a class, a function or a method) and then the value are compared to search out the copied code over original code.

Tracking Clipboard Operations: This capability is relies on the copy-paste activity of programmers for the new creation of the clones. The fundamental result of this clipboard operation is the simply clipboard activities in the editor (internally in the IDE such as Eclipse). When the code segment pasted after copy this copied segments are recorded as clone pairs**.**

Clone Detection Tools:

- CloneDr [2]
- CloneTracker [1]
- Jcd [6]
- CCFinder [3]
- Iclone [5]

II. CLONE MANAGEMENT

Basically, the clone management is the set of activities. There are many activities in clone management like clone refactoring,, tracking, visualization, classification and evolution. Another, Clone management is the umbrella activeness that is sheltering all the aspects about the clone. In current status, some benefits of clone management [14]:

o It improves software quality and customer fulfillment.

o Clone awareness with visualization and during the debugging and modification. It helps to the developers.

o Modified clone regions are notified to the developers.

o Cloning patterns are identified, by this quality of software is improved.

To manage clone, first of all they have to be identified. The overview of the clone management system is summarized in the Fig.1. The end part of clone detection created the clone documentation that saves the location of code segment and their relationship.

If the copied code changes due to development, the changes and locations of clone to be tracked and the documentation need to be uploaded. Next visualization techniques can be 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 and may be initiate refactoring. Again documentation updated after the refactoring.

Ideal clone management activities:

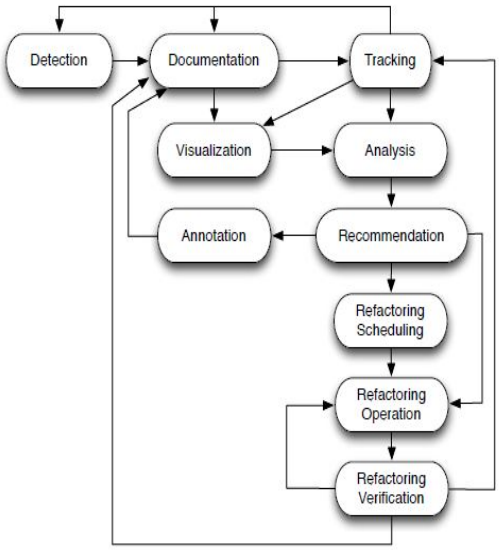

Fig.1: The overview of the clone removal methods [14]

III. REFACTORING

Refactoring is the confirmed technique for structuring on existing bodies of code renovating its internal structure while not ever-changing its external behaviour. It's the way to wash up the codes that minimize the probabilities of announcing bugs. In last line during you refactor, you are improving the look of the code just as it is been written [4].

With refactoring you can take a rough style, chaos even and reshape it into well designed code. Every step in refactoring is extraordinary simple; you progress a field from one class to different. Pull some code out of a methodology to compose into its own method and push some code up and down a hierarchy. With the assistance of refactoring you discover the balance of work changes. Additionally you find that design, instead of occurring all up front, occurs continuously during development. You learn from building the system a way to improve the design.

IV. LITERATURE SURVEY

N. Tsantalis et. al. [13] has worked on assessing the refactorability of software clones. The existence of code

clone in software structures is important and many other survey have displayed that clones will be probably risky with relevancy to the maintainability and evolution of the source code. Even with the impotence of the matter, there's stable finite support for removing software system clones through refactoring, as a result of the combination and merging of ditto code may be very challenging problem, particularly once software system clones responded to many modifications when their first opening. During this task, we tend to propose an approach for automatically assessing whether a pair of clones can be safely refactored without changing the behavior of the program. Specially, our approach examines if the variations present between the clones can be safely parameterized while not inflicting any side-effects.

C. K. Roy et. al. [10] has worked on studying the vision of software clone management: past, present, and future (Keynote Paper). This paper granted overall survey on the character 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. The authors have believed that we have got done a decent job in surveying the area of clone management and that this work may serve as a roadmap for future research in the area.

D. Rattan et al. [11] 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. [12] 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. To fill this gap, the authors have conducted an oversized empirical study over the amendment history of 200 open source projects from completely different software packages ecosystems and investigated when bad smells are introduced by developers, and therefore the circumstances and reasons behind their introduction. Their study needed the development of a strategy to identify smell introducing commits, the mining of over 0.5M commits, and therefore the manual analysis of 9,164 of them (i.e., those known as smell introducing).

M. Kim et. al. [8] 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. The authors have conjointly report on interviews with a chosen refactoring team that has led a multiyear, centralized effort on refactoring Windows. The quantitative analysis of Windows 7 version history finds the highest 5 percent of preferentially refactored modules expertise higher reduction within the variety of inter-module dependencies and several others complexity measures but increase size more than the bottom 95 percent.

R. Koschke et. al. [7] has developed the software clone management towards industrial application. This report documents the program and the ending of Dagstuhl Seminar 12071 "Software Clone Management towards Industrial Application". code clones are identical or similar items of code or design. Lots of analysis has been dedicated to software clones. In contrast to previous research, this seminar place a specific stress on industrial application of software clone management methods and tools and aimed toward gathering concrete usage situations of clone management in industry, which is able to facilitate to new industrially relevant aspects so as to form the longer term research.

M. F. Zibran et. al. [14] 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'Keele 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 rumored that among those AI techniques, the hill-climbing approach performed the most effective. CP is a comparatively recent technique that mixes the strengths of both AI and OR techniques and so are often expected to perform better. Still, an empirical comparison of CP with AI and evolutionary algorithms in optimizing the scheduling of code clone refactoring can be an interesting study.

V. FINDINGS OF THE LITERATURE REVIEW

The existing model is created to detect the code clone fragments with body of the same method or other methods and duplicate method declarations somewhere in the code files. The existing method recognizes the variability in member variables but not in the procedure body variables. In usual terms the clone detection can be classified in two major types:

i. Exact Classification

ii. Probabilistic Classification

The exact classification techniques are always bounded to find the exactly similar clones in the given source code, whereas the probabilistic classification techniques are intended to extract the exact as well as probable neighbours with matching data more than the given threshold. The exact clone detection finds the exact copy of the code in the two different places. The probabilistic approach can be classified as more stronger and flexible approach. In this study have evaluated the following:

a. The existing model relies upon the exact clone detection when it comes to the procedure clone detection and does not evaluate the changed variable or other entity declarations.

b. Two methods, which are having the exactly same flow but declared with different variable entities, are also considered clones.

c. The certain improvement must be made in order to evaluate such clones also.

The clone detection is made between the various clones of same number of lines of code. The small or sub-clones in the longer programs or functions than the evaluated code chunk are usually neglected and returned with no matching results. If some programmer adds a new lines of code with optional parameters in the end of the function which matches with a shorter function can with stand in the code by using the existing scheme. Hence it can be called that the existing method is enough capable of detecting the clones in non-optimal and optimal paradigms, but it does not evaluate the third case where the extra-optimal case has been employed. In case a third condition is called in the code clone which is making its structured unique than the existing method declaration, can be also combined by adding the third condition in the bottom to sum up the code in single segment.

The code mapping is one the most difficult task for the clone detection. A clone should not be found within the same file on the same position; otherwise it can interrupt and scramble the whole code. The accurate code mapping is very important and significant for the code clone detection. The mapping of the source code is performed by using the flow graphs or directed graphs which connect the code segments in the tree structure format. The existing method has been found inefficient in the following paragraph along with the proposed solutions:

a) The existing refactoring method relies upon the control flow graphs of individual code fragments.

b) A full mapping tree can be employed over the whole code to refactor the code with more intelligence.

VI. CONCLUSION

The proposed clone detection and refactoring methods are based upon the divide and conquer methods for the efficient removal of the code clones. In the proposed model, for the code clone detection, the code fragmentation method using the divide and conquer method for all types of clones will be utilized to detect the code clones within the code file or group of code files. The refactoring methodology will utilize the hybridized method using the combination of extract method amalgamated with inline and move method techniques for the procedure definitions. The inline temp, split temp variable along with replace loop and replace method based techniques would be utilized to realize the refactoring methods for replacing and marking the code fragments. The proposed model is expected to solve the issues related with the inability for code clone detection by using the new combination of clone detection and refactoring. The performance of the proposed model will be measured by using the accuracy measures such as recall, precision, f1-measure, elapsed time, etc.

REFERENCES

[1] Tool CloneTracker<http://cs.mcgill.ca/~swevo/clonetracker/> (accessed January 2016).

[2] Tool CloneDr<http://www.semdesigns.com/products/clone/> (accessed December 2015).

[3] Tool CCFinder<http://www.ccfinder.net/ccfinderxos.html> (accessed May 2015).

[4] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.

[5] Tool Iclone<http://www.softwareclones.org/iclones.php>

[6] Tool Jcd<http://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] 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.

[10] 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.

[11] D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, Information and Software Technology 55 (7) (2013) 1165-1199.

[12] 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.

[13] N. Tsantalis, D. Mazinanian, G. P. Krishnan, Assessing the Refactorability of Software Clones, Software Engineering, IEEE Transactions on 41 (11) (2015) 1055-1090.

[14] 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.