Code Clone Detection with Refactoring support Through Textual analysis

G. Anil kumar[1] **Dr. C.R.K.Reddy[2]** **Dr. A. Govardhan[3]** **Gousiya Begum[4]**

[1,4]MGIT, Dept. of Computer science Hyderabad, India
[2]CBIT, Dept. of Computer science,Hyderabad, India
[3]SIT, JNTUH Dept. of Computer science, Hyderabad, India

Abstract

Copying code fragments and then reuse by pasting with or without minor modifications or adaptations are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment without is called a clone of the original. One of the major shortcomings of such duplicated fragments is that if a bug is detected in a code fragment; all the other fragments similar to it should be investigated to check the possible existence of the same bug in the similar fragments. In this paper, we compare different clone detection techniques and tools. First part of this paper explains the classification of clone detection techniques and the later work done in this area and proposed method.

Key words: Software clone, Clone Detection, clone cluster, clone pair

1. INTRODUCTION

Copying code fragments and then reuse by pasting with or without minor modifications or adaptations are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment (with or without modifications) is called a clone of the original [1]. The area of clone detection (i.e., searching for duplicate fragments of source code) has received wide interest recently as indicated by numerous efforts in clone detection tool development [2]. A clone detector must try to find pieces of code of high similarity in a system's source text. The main problem is that it is not known beforehand which code fragments may be repeated. Thus the detector really should compare every possible fragment with every other possible fragment. Such a comparison is prohibitively expensive from a computational point of view and thus, several measures are used to reduce the domain of comparison before performing the actual comparisons. Even after identifying potentially cloned fragments, further analysis and tool support may be required to identify the actual clones [3].

The act of copying indicates the programmer's intent to reuse the implementation of some abstraction. The act of pasting is breaking the software engineering principle of encapsulation. While cloning may be unstructured, it is commonplace and unlikely to disappear via fiat. A clone is a program fragment that identical to another fragment. A near miss clone is a fragment, which is nearly identical to another [4]. There are different forms of redundancy in software. Software comprises both programs and data. Sometimes redundant is used also in the sense of superfluous in the software engineering literature. Redundant code is also often misleadingly called cloned code although that implies that one piece of code is derived from the other one in the original sense of this word. Although cloning leads to redundant code, not every redundant code is a clone.
There may be cases in which two code segments that are no copy of each other just happen to be similar or even identical by accident. Also, there may be redundant code that is semantically equivalent but has a completely different implementation [5].

Clones in general are classified under 4 categories. The Clones in general are classified under 4 categories. The first two may be detected through the similarities found in the program text that has been copied. They may be defined as:
• Type 1 is an exact copy without modifications (except for whitespace and comments).
• Type 2 is a syntactically identical copy; only variable, type, or function identifiers vary.
The results of the code clone detection are usually given as clone pairs/clone clusters along with their location/occurrence.
• Type 3 is copy with further modifications, (a new statement can be added, or some statements can be removed)
• Type 4 clones are the results of semantic similarity between two or more code fragments.

Clone detection techniques attempt at finding duplicated code, which may have undergone minor changes afterward. The typical motivation for clone detection is to factor out copy-paste-adapt code, and replace it by a single procedure [6]. Clone detection finds code in large software systems that has been replicated and modified by hand. Remarkably, clone detection works because people copy conceptually identifiable blocks of code, and make only a few changes,

which means the same syntax is detectably repeated. Each identified clone thus indicates the presence of a useful problem domain concept, and simultaneously provides an example implementation. Differences between the copies identify parameters or points of variation. Clones can thus enhance a product line development in a number of ways: removal of redundant code, lowering maintenance costs, identification of domain concepts for use in the present system or the next, and identification of parameterized reusable implementations [7]. Detecting code clones is useful for software development and maintenance tasks including identifying refactoring candidates, finding potential bugs, and understanding software evolution. Most clone detectors are based on textual similarity [8].

Cloning is known to hamper productivity of software maintenance in classical code-based development environments. This is due to the fact that changes to cloned code are error-prone as they need to be carried out multiple times for all (potentially unknown) instances of a clone. Hence, the software engineering community developed a multitude of approaches and powerful tools for the detection of code clones [9]. Code clones are sections of code that occur in multiple locations in a program. Clone detection tools aim to automatically search for clones and report any detected clones back to the user. A textual representation of the result consists of clones being listed together, along with the source file names and line locations (i.e., starting and ending line location) of each clone instance. A scatter plot is a popular graphical representation of clone detection results where duplicate sections of code are identified as a sequence of connected dots in a graph [10]. Fast algorithms typically fail to identify some Type-2 and most Type-3 clones, but scale to large systems, while those that target Type-3 clones using dependence-based algorithms may find Type-3 clones, but at a high computational cost. Thus, the current state of the art presents the software engineer with a classic 'speed-quality' trade off [11].

Code clones are required to be tracked, managed, and possibly should be removed through refactoring wherever feasible. And support for such activities should be integrated with the IDEs for blending clone management with actual development effort. However, most clone detectors are developed as separate tools. Those few tools that are integrated with IDEs are mostly focussed in detecting Type-1 and Type-2 clones, and are yet to offer sufficient support for flexible clone management and refactoring [12]. Clone detection techniques are promising in this respect, due to two likely causes of code cloning occurring within scattered crosscutting concern implementations. First, by definition, scattered code is not well modularized. Several reasons can be identified for this lack of modularity, including missing features of the implementation language (exception handling or aspects, for instance), or the way the system was designed. In both cases, developers are unable to reuse concern implementations through the language module mechanism. Therefore, they are forced to write the same code over and over again, typically resulting in a practice of copying existing code and adapting it slightly to their needs [13]. An important application of clone detection is the improvement of source code quality by refactoring duplicated code fragments [14].

2. MOTIVATIONS OF THE RESEARCH

Clone detection techniques aim at finding duplicated code, which may have been adapted slightly from the original [15]. Token based approach is applied for the detection of simple clones. It provides a suitable level of flexibility for the task by limiting the language dependence, being resilient to the differences in code layout, while providing a good mechanism for detecting parameterized simple clones. Having transformed a source program into a string of tokens, we compute the maximal repeats in the string with a suffix array based algorithm [16]. These maximal repeats, with some heuristic based pruning, form clone classes. Although our detection of simple clones is much similar to the previously published approach, the novel contribution is in the introduction of a simple and flexible tokenization technique, and the selection of efficient data structures and algorithms for token string manipulation [17].

Software projects contain much similar code (i.e., code clones), which may be introduced by many commonly adopted software development practices, such as reusing a generic framework, following a specific programming pattern, and directly copying and pasting code. These practices can improve the productivity of software development by quickly replicating similar functionalities. However, such practices, especially copying and pasting, can also reduce program maintainability and introduce subtle programming errors. For example, when enhancements or bug fixes are done on a piece of duplicated code, it is often necessary to make similar modifications to the other instances of the code [18].

Copying code fragments and then re-use by pasting with or without minor modifications or adaptations are common activities in software development. This type of re-use approach of existing code is called *code cloning* and the pasted code fragment is called a '*clone'* of the original. The cloned fragments have also been classified under four categories based on the extent of their similarity. Code cloning is not only assumed to inflate maintenance costs but also considered defect-prone as inconsistent changes to code duplicates can lead to unexpected behavior. Such cloned code is considered harmful for two reasons: (1) multiple, possibly unnecessary, duplicates of code increase maintenance costs and, (2) inconsistent changes to cloned code can create faults and, hence, lead to incorrect program behavior. It is important to understand, that clones do not

directly cause faults but inconsistent changes to clones can lead to unexpected program behavior. Clone detectors have been applied to a large variety of tasks in both research and practice, including quality assessment, software maintenance and reengineering, identification of crosscutting concerns, plagiarism detection and investigation of copyright infringement.

2.1 RELATED WORKS

A handful of clone detection schemes, which employ textual and metric techniques for improved performance, have been presented in the literature. In addition to the above, some researchers have made use of code detection techniques for detecting 2 or more clones in a software code. Recently, incorporating textural and metric schemes for detecting the clones to improve its performance and effectiveness has received a great deal of attention among researchers in software engineering community. A brief review of some recent researches is presented here.

Fabio Calefato *et al* [19] described how a semi automated approach could be used to identify cloned functions within scripting code of web applications. The approach was based on the automatic selection of potential function clones and the visual inspection of selected script functions. The results obtained from the clone analysis of four web applications showed that the semi automated approach was both effective and efficient at identifying function clones in web applications, and could be applied to prevent clone from spreading or to remove redundant scripting code.

Stephane Ducasse *et al* [20] investigated a number of simple variants of string-based clone detection that normalize differences due to common editing operations, and assessed the quality of clone detection for very different case studies. Their results confirmed that the inexpensive clone detection technique generally achieved high recall and acceptable precision. Overzealous normalization of the code before comparison, however, could result in an unacceptable numbers of false positives.

C. Kapser *et al* [21] presented an in-depth case study of cloning in a large software system that is in wide use, the Apache web server; they provided insights into cloning as it exists in this system, and they demonstrated techniques to manage and make effective use of the large result sets of clone detection tools. In their case study, they found several interesting types of cloning occurrences, such as “cloning hotspots”, where a single subsystem comprising only 17% of the system code contained 38.8% of the clones. They also founded several examples of cloning behavior that were beneficial to the development of the system, in particular cloning as a way to add experimental functionality.

Chanchal K. Roy *et al* [22] provided a qualitative comparison and evaluation of the current state-of-the-art in clone detection techniques and tools, and organized the large amount of information into a coherent conceptual framework. They began with background concepts, a generic clone detection process and an overall taxonomy of current techniques and tools. They then classified, compared and evaluated the techniques and tools in two different dimensions. Finally, they provided examples of how one might use the results of that study to choose the most appropriate clone detection tool or technique in the context of a particular set of goals and constraints.

Robert Tibshirani *et al* [23] applied the fused lasso method to the “hot-spot” detection problem in comparative genomic hybridization (CGH) data. The CGH signal was approximated by a piecewise function that has relatively sparse areas with nonzero values. Hence, the method was useful for determining which areas of the signal were likely to be nonzero.

Mohammed Abdul Bari *et al* [24] discussed the concept of code cloning, presented overcall taxonomy of current techniques and tools, and classified evolution tools in two different format as static code clone and dynamic code cloning, that together presented with program analysis, secondly as a solution the static code was divided into four parts as T1, T2, T3, T4, to finally develop a process to detect and remove code cloning.

3. PROPOSED METHODOLOGY

The area of Clone Detection has considerably evolved over the last decade, leading to approaches with better results, but at the same time with increasing complexity using elaborate algorithms and tool chains. Some of the existing techniques for clone detection are textual comparison, token comparison, comparison of Abstract Syntax trees, Suffix trees, Program Dependency Graphs, etc. The existing scalable and semantics-based approaches are limited to finding program fragments which are similar only in their syntax or semantically equivalent control structures. The other techniques listed above require more complex parsing techniques while the Precision and recall of the techniques on the average remain more or less equal. Also most of the Clone Detection techniques are confined only to a certain type of clone. No clone detection tool has been proposed for the detection of all four types of clones.

This is a proposal for a new technique for code clone detection, which helps us to detect all the four types of clones as given in literature. It is a lightweight method for the detection of clones. It also provides refactoring support for further solutions with the detected clones. Our proposal is the hybrid combination of metric-based approach combined with the textual comparison of the source code for the detection of functional Clones in source code. Various metrics had been formulated and their values were utilized during the detection process. Compared to the other

approaches, this method is considered to be the least complex and is to provide a more accurate and efficient way of Clone Detection. It has to be implemented as a tool using Java.

REFERENCES

[1] Chanchal Kumar Roy and James R Cordy, "A Survey on Software Clone Detection Research", *Computer and Information Science*, Vol. 115, No. 541, pp. 115, 2007

[2] Robert Tairas, "Clone detection and refactoring", Proceeding of OOPSLA '06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 780-781, New York, USA, 2006

[3] Chanchal K. Roy, James R. Cordya and Rainer Koschkeb, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal Science of Computer Programming, Vol. 74, No.7, pp. 470-495, May 2009

[4] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna and Lorraine Bier, "Clone Detection Using Abstract Syntax Trees", Proceedings of the International Conference on Software Maintenance, pp. 368, Washington DC, USA 1998

[5] Kodhai.E, Perumal.A, and Kanmani.S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones", Proceedings of the International Joint Journal Conference on Engineering and Technology, pp. 99-103, 2010

[6] Magiel Bruntink, Arie van Deursen,Tom Tourwe and Remco van Engele, "An Evaluation of Clone Detection Techniques for Identifying Crosscutting Concerns", Proceedings of the 20th IEEE International Conference on Software Maintenance, pp. 200- 209,Washington DC, USA 2004

[7] Ira D. Baxter and Dale Churchett, "Using Clone Detection to Manage a Product Line", Clone detection using abstract syntax trees, pp. 1-3,1998

[8] Heejung Kimy, Yungbum Jungy, Sunghun Kimx and Kwangkeun Yi, "MeCC: Memory Comparison-based Clone Detector", 33rd international conference on software engineering, Waikiki,Honolulu, Hawaii, May 21-28,2011

[9] Florian Deissenboeck, Benjamin Hummel, Elmar Jurgens, Bernhard Schatz, Stefan Wagner, Jean-François Girard and Stefan Teucher, "Clone detection in automotive model-based development", Proceedings of the 30th international conference on Software engineering, pp. 613-622,New York, NY, USA,2008

[10] Robert Tairas, Jeff Gray and Ira Baxter, "Visualization of clone detection results",Proceedings of the 2006 OOPSLA workshop on eclipse technology exchange ACM, pp 50-54, New York,USA,2006

[11] Yue Jia, David Binkley, Mark Harman, Jens Krinke and Makoto Matsushita, "KClone: A Proposed Approach to Fast Precise Code Clone Detection", Computer and Information Science, pp. 12-16, 2009

[12] Minhaz F. Zibran and Chanchal K. Roy, "Towards Flexible Code Clone Detection, Management, and Refactoring in IDE", Fifth International Workshop on Software Clones,Waikiki, Hawaii, USA,May 23,2011

[13] M. Kim, L. Bergman, T.A. Lau, and D. Notkin, "An Ethnographic Study of Copy and Paste Programming Practices in OOPL," Proc. Int'l Symp. Empirical Software Eng. (ISESE '04), pp. 83-92, Aug. 2004

[14] M. Rieger, S. Ducasse, and G. Golomingi, "Tool Support for Refactoring Duplicated OO Code," Proc. European Conf. Object- Oriented Programming (ECOOP '99), pp. 177-178, June 1999.

[15] Magiel Bruntink, Arie van Deursen,Remco van Engelen, and Tom Tourwe, "On the Use of Clone Detection for Identifying Crosscutting Concern Code", Ieee Transactions On Software Engineering, Vol. 31, No. 10,pp. 804-818, October 2005

[16] Abouelhoda M.I., Kurtz S.and Ohlebusch E, "The enhanced suffix array and its applications to genome analysis", In Proc. Workshop on Algorithms in Bioinformatics, vol. 2452,pp. 449–463, Berlin, 2002

[17] Hamid Abdul Basit and Stan Jarzabek, "Detecting Higher-level Similarity Patterns in Programs", European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 1-10 Lisbon, Sept. 2005 [18] Lingxiao Jiang, Zhendong Su and Edwin Chiu, "Context-based detection of clone-related bugs", Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 55 – 64, New York, USA, 2007.

[19] Fabio Calefato, Filippo Lanubile, Teresa Mallardo, "Function Clone Detection in Web Applications: A Semiautomated Approach", Journal of Web Engineering, Vol. 3, No.1, pp.003-021, 2004.

[20] Stephane Ducasse, Oscar Nierstrasz and Matthias Rieger, "On the effectiveness of clone detection by string matching", Journal of Software Maintenance and Evolution, Vol.18, pp.37–58, 2006.

[21] C. Kapser and M. W. Godfrey. Supporting the Analysis of Clones in Software Systems: A Case Study. J. Softw. Maint. Evol., Vol.18, No.2, pp.61–82, 2006.

[22] Chanchal K. Roy, James R. Cordya and Rainer Koschkeb, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal Science of Computer Programming, Vol. 74, No.7, May 2009.

[23] Robert Tibshirani, Pei Wang, "Spatial smoothing and hot spot detection for CGH data using the fused lasso", Biostatistics, pp.1-7, 2007.

[24] Mohammed Abdul Bari, Dr. Shahanawaj Ahamad, "Code Cloning: The Analysis, Detection and Removal", International Journal of Computer Applications (0975 – 8887) Vol. 20, No.7, April 2011.