Textual and Metric Analysis to Figure out all Types of Clones : In Detection Method

M. Dhivya, R.Bharathi
Prist university, Puducherry
dhivimadi@gmail.com,bharathi_r@gmail.com

Abstract: ***A Clone Detection approach is to find out the reused fragment of code in any application to maintain .Various types of clones are being identified by clone detection techniques. Since clone detection was evolved, it provides better results and reduces the complexity. A different clone detection tool makes the detection process easier and efficiently produces the results. In many existing system, it mainly focuses on line by line detection or token based detection to find out the clone in the system. So it makes the system to take long time to process the entire system. This paper proposes combination of textual and metric analysis of a source code for the detection of all types of clone in a given set of fragment of java source code. Various semantics had been formulated and their values were used during the detection process. This metrics with textual analysis provides less complexity in finding the clones and gives accurate results***

Keywords: *Clone Detection, Metrics, Complexity, Semantics.*

I. Introduction

In computer programs, we can also have different types of redundancy. We should note that not every type of redundancy is harmful. 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.

Clones are segments of code that are similar according to some definition of similarity. —Ira Baxter, 2002

According to this definition, there can be different notions of similarity. They can be based on text, lexical or syntactic structure, or semantics. They can even be similar if they follow the same pattern, that is, the same building plan. Instances of design patterns and idioms are similar in that they follow a similar structure to implement a solution to a similar problem. Semantic difference relates to the observable behavior. A piece of code, A, is semantically similar to another piece of code, B, if B subsumes the functionality of A, in other words, they have “similar” pre and post conditions. Unfortunately, detecting such semantic redundancy is undecidable in general although it would be worthwhile as you can often estimate the number of developers of a large software system by the number of hash table or list implementations you find. Another definition of redundancy considers the program text:

Two code fragments form a redundancy if their program text is similar. The two code fragments may or may not be equivalent semantically. These pieces are redundant because one fragment may need to be adjusted if the other one is changed. If the code fragments are executable code, their behavior is not necessarily equivalent or subsumed at the concrete level, but only at a more abstract level. For instance, two code pieces may be identical at the textual level including all variable names that occur within but the variable names are bound to different declarations in the different contexts. Then, the execution of the code changes different variables. The common abstract behavior of the two code segments is to iterate over a data structure and to increase a variable in each step.

Program-text redundancy is most often the result of copy&paste; that is, the programmer selects a code fragment and copies it to another location. Sometimes, these programmers are forced to copy because of limitations of the programming language. In other cases, they intend to reuse code. Sometimes these clones are modified slightly to adapt them to their new environment or purpose. Several authors report on 7-23% code duplication [4, 5, 6]; in one extreme case even 59%Clearly, the definition of redundancy, similarity, and cloning in software is still an open issue.

There are basically two kinds of similarities between two code fragments. Two code fragments can be similar based on the similarity of their program text or they can be similar in their functionalities without being textually similar. The first kind of clones is often the result of copying a code fragment and then pasting to another location. In this section, we consider clone types based on the kind of similarity two code fragments can have:

Textual Similarity: Based on the textual similarity we distinguish the following types of clones :

Type I: Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

Type II: Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.

Type III: Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments.

Functional Similarity: If the functionalities of the two code fragments are identical or similar and referred as Type IV clones.

Type IV: Two or more code fragments that perform the same computation but implemented through different syntactic variants.

The results of the code clone detection are usually given as clone pairs/clone clusters along with their location/occurrence.

Clone Pair (CP): pair of code portions / fragments which are identical or similar to each other.

Clone Cluster (CC): the union of all clone pairs which have code portions in common

Auspiciously, many techniques for the detection of code clones have been proposed. They show that lightweight text- based techniques can find clones with high accuracy and confidence, but detected clones often do not correspond to appropriate syntactic units [8]. An Incremental detection technique detects clones in less time in each revision separately [2]. Moreover, it only detects the similar clones of type 1. The complexity of all the methods is high and this can be reduced with the computed metrics values.

In this paper, a novel code clone detection method using textual analysis and metrics-based approach has been proposed. It has also been implemented as a tool using Java. The tool efficiently and accurately detects type-1, type-2,type-3 and type-4 clones found in source codes at method level in JAVA open source code projects. This paper contains four major sections. Section II describes about the related research work are currently implemented in various domain. Section III describes the implementation of the proposed method. Finally, Section III concludes the paper.

2. Related Work

The foremost question to answer is "What is a clone?" Generally speaking, two code fragments form a clone pair if they are similar enough according to a given definition of similarity. Different definitions of similarity and associated levels of tolerance allow for different kinds and degrees of clones. Ideally, code is free of redundancy. A piece of code, A, is redundant if there is another piece of code, B, that subsumes the functionality of A, in other words, they have "similar" pre and post conditions. We call such a pair (A, B) a semantic clone. Unfortunately, detecting semantic clones is undecidable in general.

Another definition of similarity considers the program text: Two code fragments form a clone pair if their program text is similar. The two code fragments may or may not be equivalent semantically. These kinds of clones are most often the result of copy&paste; that is, the programmer selects a code fragment and copies it to another location. Clones of this nature may be compared on the basis of the program text that has been copied. We can distinguish the following types of clones:

•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 have been changed.

•Type 3 is a copy with further modifications; statements have been changed, added, or removed.

•Type 4 is a same functionality but different fragment of codes.

Several techniques have been proposed to find these types of clones.

Textual comparison: whole lines are compared to each other textually [15] using hashing for strings. The result may be visualized as a dot plot, where each dot indicates a pair of cloned lines. Consecutive duplicated lines can be spotted as uninterrupted diagonals or displaced diagonals in the dotplot [12].

Token comparison: Baker's technique is also a line based comparison where the token sequences of lines are compared efficiently through a suffix tree. First, each token sequence for whole lines is summarized by a so called functor that abstracts of concrete values of identifiers and literals. The functor characterizes this token sequence uniquely. Concrete values of identifiers and literals are captured as parameters to this functor. An encoding of these parameters abstracts from their concrete values but not from their order so that code fragments may be detected that differ only in systematic renaming of parameters. Two lines are clones if they match in their functors and parameter encoding. The functors and their parameters are summarized in a trie1 that represents all suffixes of the program in a compact fashion. Every branch in this trie represents program suffixes with common beginnings, hence, cloned sequences. Kamiya et al. increase recall for superfluous different, yet equivalent sequences by normalizing the token sequences [16] .Because syntax is not taken into account; the found clones may overlap different syntactic units, which cannot be replaced through functional abstraction. Either in a preprocessing [10, 13] orpost-processing [14, 16] step, clones that completely fall in syntactic blocks can be found if block delimiters are known.

Metric comparison: Merlo et al. gathers different metrics for code fragments and compares these metric vectors instead of comparing code directly [1, 11, 18, 20, 21]. An allowable distance (for instance, Euclidean distance) for these metric vectors can be used as a hint for similar code.

Comparison of abstract syntax trees (AST): Baxter et al. partition sub trees of the abstract syntax tree of a program based on a hash function and then compare sub trees in the same partition through tree matching (allowing for some divergences) [9]. A similar approach was proposed earlier by Yang [22] using dynamic programming to find differences between two versions of the same file.

Comparison of program dependency graphs: control and data flow dependencies of a function may be represented by a program dependency graph; clones may be identified as isomorphic sub graphs [17, 19].

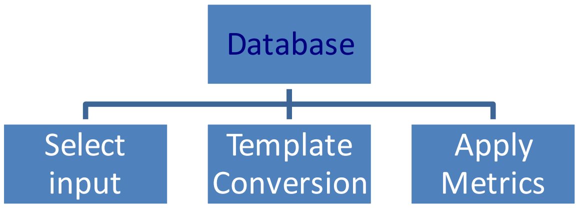

Figure 1. System Architecture of the developing tool.

3. Our Proposed Method

The proposed method is implemented as a tool in java. The system architecture of the tool is as shown in Fig. 1. The tool developed initially parses through the given input source code and identifies the various methods present. Then built-in hand-coded parser [6] parses the various methods using an islanddriven approach [6]. Having identified the methods, the various metrics formulated are computed for each method and the metrics values for each method are stored in the database. With the help of the metric values the possible potential clone pairs are extracted and are further put forth for the textual comparison.

The pairs that prove similar in the textual comparison are listed as the clones. The detection tool thus developed is lightweight i.e. it doesn't employ any external parsers and requires less overhead compared to the other methods. The process of clone detection has been divided into a number of phases. These phases include Input & Pre- Processing, Template Conversion, Metrics Computation, and finally the detection of the type-1, type-2, type-3 and type-4cloned methods.

3.1. Select Input Project

This phase includes the selecting input project, source code standardization and the normalization. Selecting the project involves the concatenation of all the files of the same project into a single large file for an effective parsing. In the next step the integrated file is parsed for the removal of comments, whitespaces and pre-processor statements. Source code is re-structured to a standard format which is important for establishing similarity of the cloned fragments. These steps are very similar to normalization procedures and yield a significant gain in the recall.

3.2. Template Conversion

Template conversion is nothing but the transformation of the inputted source code into a pre-defined set of statements or conversion into a standard intermediary form. For example, renaming of data types, variables, function names, etc., as in Fig. This type of format called the 'template' is used in the textual comparison of the selected candidates while detecting the type-2 cloned methods where as per the definition, function identifiers, variable names, types etc., are edited during the cloning process and mere textual comparison would not suffic once template conversion is getting over the source files and template file is stored in the database for applying metrics.

```
int sim_comp(avg,marks,tot,count)       DAT FUN_NAME(X,X,X,X,X)
{                                       DAT X;
float avg;                              DAT X;
int marks[];                            DAT X;
double tot[];                           DAT X;
int count;                              DAT X;
total=mark[i]+mark[i+1];                {
avg=total/count;                        LOOP
for(int i=0;i<count;i++)                {
{                                       IF
if(avg>=85)                             PRINT
System.out.println("Grade A");          }
}                                       PRINT
System.out.println("Thank you");        }

}
```

3.3. Apply Metrics

A set of 12 existing method level metrics are used for the detection of type-1, type-2, type-3 and type-4 clone methods. They are as follows:

1)No. of effective lines of code in each method.
2)No. of arguments passed to the method.
3)No. of function calls in each method.
4)No. of local variables declared in each method.
5)No. of conditional statements in each method.
6)No. of looping statements in each method.
7)No. of return statements in each method.
8)No. of function calling in each method
9)No. of inheritance in each method
10)No. of virtual functions in each method
11)No. of overloading constructor in each method
12)No. of overriding functions in each method

The metrics are computed for each of the methods identified and the values are stored in a database. The various metric values for the code fragment. The descriptive statistics of the metric values obtained for the various methods. Having computed the metric values, the method pairs with equal or similar set of values are identified by comparison of the records in the database. The short-listed set of candidates is then textually compared to be confirmed as clone pairs.

3.4. Finding Clone Types, Pairs and Clusters

The identification of the potential clone pairs is done by taking up a line by line comparison of the standardized and normalized source code for type-1 clone methods while comparison of the template methods for type-2 clone methods. There is some modifications in the fragments but there is some similarities means it should be declared astype-3 by matching template with the exact code.

Then fragments are completely different but produce similar output, then it is declared as type- 4 clone. The exact and corresponding matches in both cases are declared as cloned methods of the corresponding type[7,9]. The identified cloned methods are then clustered separately for each type and the clusters are uniquely numbered. Clustering gives a clear image of how the methods were cloned and helps to provide an easier review process.

4. Conclusion

The proposed paper uses a light weight technique to detect functional clones with the computation of metrics based technique with the textual analysis technique. By implementing various metrics in this paper gives various benefits to improve the precision and recall and also reducing the total comparison overhead. Various metrics and textual comparison is performed over different fragments. And also plan obtain higher recall and precision value. Currently working on the implementation part of this paper in finding and classifying the functional clones in JAVA.

5. References

i. *[1]Kodhai.E, Kanmani.S, Kamatchi.A, Radhika.R, Detection of Type-1 and Type-2 Clone Using Textual Analysis and Metrics in ITC,2010.*

ii. *[2]Chanchal K. Roy, James R. Cordy, NICAD: Accurate Detection of Near-Miss Intentional Clones using FlexiblePretty-Printing and Code Normalization in ICPC, pp 172-18,2008*

iii. *[3]B. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In WCRE, pp. 86-95, 1995*

iv. *[4]C. Kapser and M. Godfrey."Cloning Considered Harmful" Considered Harmful. In WCRE, pp. 19 -28, 2006.*

v. *[5]Z. Li, S. Lu, S. Myagmar and Y. Zh hou. CP-Miner:Finding Copy- Paste and Related Bugs in Large--ScaleSoftware Code. IEEE TSE, 32(3):176-192, 2006.*

vi. *[6]F.V. Rysselberghe and S. Demeyer. . Evaluating Clone Detection Techniques. In ELISA, 12 pp., 2003.*

vii. *[7]Merlo, Detection of Plagiarism in University Projects using Metrics based Spectral Similarity. In the Dagstuhl Seminar: Duplication, Redundancy, and Similarity in Software- 2007*

viii. *[8]Rainer Koschke, Raimar Falke, PierreFrenzel, Clone Detection using Abstract Syntax Suffix Trees– Working Conference on Reverse Engineering – 2006.*

ix. *[9]I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier.Clone Detection Using Abstract Syntax Trees. In ICSM, 1998.*

x. *[10]J. R. Cordy, T. R. Dean, and N. Synytskyy. Practicallanguage-independent detection of near-miss clones. In CASCON. IBM Press, 2004.*

xi. *[11]G. Di Lucca, M. Di Penta, and A. Fasolino. An approach to identify duplicated web pages. In COMPSAC, 2002.*

xii. *[12]S. Ducasse, M. Rieger, and S. Demeyer. A Language Independent Approach for Detecting Duplicated Code. In ICSM,1999.*

xiii. *[13]D. Gitchell and N. Tran. Sim: a utility for detecting similarity in computer programs. In SIGCSE. ACM Press, 1999.*

xiv. *[14]Y. Higo, Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue. On software maintenance process improvement based on code clone analysis. In International Conference on Product Focused Software Process Improvement, volume 2559 of Lecture Notes In Computer Science. Springer, 2002.*

xv. *[15]J. H. Johnson. Identifying redundancy in source code using fingerprints. In CASCON. IBM Press, 1993.*

xvi. *[16]T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: AMulti-Linguistic Token-based Code Clone Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, 28(7):654–670, 2002.*

xvii. *[17]R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In Proc. Int. Symposium on StaticAnalysis, 2001.*

xviii. *[18]K. Kontogiannis, R. D. Mori, E. Merlo, M. Galler, and M. Bernstein. Pattern matching for clone and concept detection. Automated Software Engineering, 3(1/2):79–108, 1996.*

xix. *[19]J. Krinke. Identifying Similar Code with Program Dependence Graphs. In WCRE, 2001.*

xx. *[20]B. Lagu¨e, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In ICSM, 1997.*

xxi. *[21]F. Lanubile and T. Mallardo. Finding function clones in web applications. In CSMR, 2003.*

xxii. *[22] W. Yang. Identifying syntactic differences between twoprograms.Software–Practice and Experience, 21(7):739–755,1991.*