ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 2, Issue 12, June 2013

A Review on Code Clone Analysis and Code Clone Detection

Prajila Prem

Abstract*— Code duplication is a common problem, and a well known sign of bad design. But Code duplication is one of the most popular forms of software reuse among developers. Clone detection or code duplication detection is the technique concerned with the identification of code fragments that essentially compute the same results .The primary aim of clone detection is to identify clone code and replace them with a single function call where the function would mimic the behavior of a single instance from the set of clones. As a result of that, in the last decade, the issue of detecting code duplication led to various tools that can automatically find duplicated blocks of code. In this paper different methods for code clone detection, different tools and technique used for that and the code analysis will be discussed.

***Index Terms*—Code Analysis, Code Clones, Code Clone Detection.**

I. INTRODUCTION

In software development, it is common to reuse some code fragments by copying with or without small modifications. These kinds of code fragments are called code clones. Cloning or duplication of codes can be harmful or beneficial. Source code cloning occurs when a developer reuses existing code in a new context by making a copy that is altered to provide new functionality. There are other reasons like copy & paste; lack of technical knowledge in developers and sometimes accidentally clones are introduced. Clones are segments of code that are similar according to some definition of similarity --Ira Baxter, 2002 Cloned code can be problematic for the reasons: multiple (unnecessary) duplicates of code increase size of source code, maintenance costs and, inconsistent changes to cloned code can create faults and which lead to incorrect program behavior .So a code clone detection method is needed. Code clone detection is the process of locating segments of similar source code, according to the definition of similarity, within a software system. Code clone analysis uses those results to examine code cloning in a software system. The goal of clone analysis is to understand the use of code cloning and study the individual code clones. Code clone detection and analysis can be done as a three step process: generating a list of candidate clones, post-processing the results, and analyzing the clones.

II. RELATED WORKS

Related work begins with a basic introduction to clone detection terminology.

A. Code Fragment

A code fragment (CF) is any sequence of code lines (with or without comments). It can be of any granularity, e.g., function definition, begin-end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base and is denoted as a triple (CF.FileName, CF.BeginLine, CF.EndLine).

B. Code Clone

A code fragment CF2 is a clone of another code fragment CF1 if they are similar by some given definition of similarity, that is, f(CF1) = f(CF2) where f is the similarity function. Two fragments that are similar to each other form a clone pair (CF1; CF2), and when many fragments are similar, they form a clone class or clone group.

C. Clone Types

There are two main kinds of similarity between code fragments. Fragments can be similar based on the similarity of their program text, based on their functionality. The first kind of clone is often the result of copying a code fragment and pasting into another location. In the following the types of clones based on both the textual (Types 1 to 3) [1] and functional (Type 4) similarities are described:

Type-1: Identical code fragments except for variations in whitespace, layout and comments.

Type-2: Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

Type-3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type-4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

D. Precision and Recall

These are the two key terms used when discussing the characteristics of the candidate code clones returned by a clone detector. Precision refers to the quality of the candidates returned by the detection method: high precision indicates the candidate code clones are mostly correctly identified as code clones and low precision indicates the candidate code clones contain many candidates that are not actual code clones. Recall refers to the overall percentage of artifacts that exist in the source code that have been detected by the clone detector: high recall indicates most of the code clones in the source code have been found, low recall indicates most of the code clones in the source code have not been found.[2] When comparing code clone detection techniques, precision and

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 2, Issue 12, June 2013

recall are often referenced as measures of the accuracy and completeness of the candidate code clones.

Precision, p= number of correct found clone / Number of all found clone

Recall, r= number of correct found clone / Number of possible existing clone

The detection of code clones is a two phase process which consists of a transformation and a comparison phase. In the first phase, the source text is transformed into an internal format which allows the use of a more efficient comparison algorithm. During the succeeding comparison phase the actual matches are detected. Due to its central role, it is reasonable to classify detection techniques according to their internal format.

III. AVAILABLE TECHNIQUES TO DETECT CLONES

There are mainly four types of code clone detection techniques. Table 1 gives the classification of code clone and its techniques.

1) Textual approach: Textual approaches (or text-based techniques) use little or no transformation on the source code before the actual comparison, and in most cases raw source code is used directly in the clone detection process. Examples: SDD, NICAD, Simian1, etc.

2) Lexical approach: Lexical approaches (or token-based techniques) begin by transforming the source code into a sequence of lexical "tokens" using compiler-style lexical analysis. The sequence is then scanned for duplicated subsequences of tokens and the corresponding original code is returned as clones. Lexical approaches are generally more robust over minor code changes such as formatting, spacing, and renaming than textual techniques. Examples: Dup, CCFinder [3], CPMiner etc.

3) Syntactic approaches: Syntactic approaches use a parser to convert source programs into parse trees or abstract syntax trees which can then be processed using either tree matching or structural metrics to find clones. Examples: CloneDr, Deckard, CloneDigger etc.

(4) Semantic approaches: Semantics-aware approaches have also been proposed, using static program analysis to provide more precise information than simply syntactic similarity. In some approaches, the program is represented as a program dependency graph (PDG). The nodes of this graph represent expressions and statements, while the edges represent control and data dependencies. Examples: Duplix, GPLAG etc.

Table 1: Classification of code clone & techniques

	Text based	Token based	AST based	PDG based
Category	textual	textual	semantic	semantic
Supported clone	Type 1	Type1,2	Type1,2,3	Type1,2,3
Complexity	O(n)	O(n)	O(n)	$O(n^3)$
Meaning of n	Lines of code	No. of token	Node of AST	Node of PDG

1. Textual Approaches

Textual approaches (text-based techniques) use little or no transformation on the source code before the actual comparison. One of the leading text-based clone detection approaches is that using string-based Dynamic Pattern Matching to textually compare whole lines that have been normalized to ignore whitespace and comments.

String based: String based techniques use basic string transformation and comparison algorithms which makes them independent of programming languages. Techniques in this category differ in the string comparison algorithm. Comparing calculated signatures per line is one possibility to identify for matching substrings. Line matching, which comes in two variants, is an alternative which is selected as representative for this category because it uses general string manipulations.

Simple line matching: is the first variant of line matching in which both detection phases are straightforward. Only minor transformations using string manipulation operations, which can operate using no or very limited knowledge about possible language constructs, are applied. Typical transformations are the removal of empty lines and white spaces. During comparison all lines are compared with each other using a string matching algorithm. This result in a large search space which is usually reduced using hashing buckets. Before comparing all the lines, they are hashed into one of n possible buckets. Afterwards all pairs in the same bucket are compared.

Parameterized line matching: is another variant of line matching which detects both identical as well as similar code fragments. The idea is that since identifier–names and literals are likely to change when cloning a code fragment, they can be considered as changeable parameters. Therefore, similar fragments which differ only in the naming of these parameters are allowed. To enable such parameterization, the set of transformations is extended with an additional transformation that replaces all identifiers and literals with one, common identifier symbol like"$P". Due to this additional substitution, the comparison becomes independent of the parameters. Therefore no additional changes are necessary to the comparison algorithm itself.

2. Token based approach

Token based techniques use a more sophisticated transformation algorithm by constructing a token stream from the source code, so it requires a lexer. Parameterized Matching with Suffix Trees consists of three steps. In the first step, a lexical analyzer passes over the source text transforming identifiers and literals in parameter symbols, while the typographical structure of each line is encoded in a non-parameter symbol. One symbol always refers to the same identifier, literal or structure. The result of this first step is a parameterized string.[4] Once the parameterized string is

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 2, Issue 12, June 2013

constructed, we have to decide whether two sequences in this parameterized string are a parameterized match or not. After the lexical analysis, a data structure called a parameterized suffix tree (p-suffix tree) is built for the parameterized string. The use of a suffix tree allows a more efficient detection of maximal, parameterized matches.

3. Syntactic Approaches

Syntactic approaches use a parser front-end like that of a compiler to convert source programs into parse trees or abstract syntax trees (ASTs)[5] which can then be processed using either tree-matching or metrics to find clones.

Tree-based Approaches: Tree-based method first convert the program to a parse tree or abstract syntax tree (AST) using a parser for the target language. Tree-matching techniques are then used to find similar sub trees, and the corresponding code segments are returned as clone pairs. Variable names, literal values and other tokens in the source may be abstracted in the tree representation, allowing for more sophisticated detection of clones. One of the AST-based clone detection techniques is *CloneDr*. A compiler generator is used to generate an annotated parse tree (AST). Subtrees are then compared based on a hash function and tree matching, and corresponding source code fragments are returned as clones. We can find exact and parameterized clones at a more abstract level by converting the AST to XML and using a data mining technique to find clones. To avoid the complexity of full subtree comparison AST subtrees are represented as serialized token sequences (suffix-trees), allowing syntactic clones to be detected more efficiently.

Metrics-based Approaches: Metrics-based techniques gather a number of metrics for code fragments and then compare metrics vectors rather than code or ASTs directly. One popular technique involves fingerprinting functions, metrics calculated for syntactic like a class, function, method or statement that provides values that can be compared to find clones of these syntactic units. In most cases, the source code is first parsed to an AST or CFG (control flow graph) representation to calculate the metrics. Metrics are calculated from names, layout, expressions and control flow of functions.[6] A clone is defined as a pair of whole function bodies with similar metrics values. Metrics-based approaches have also been applied to finding duplicate web pages and clones in web documents.

4. Semantic Approaches

Semantics-aware methods have also been proposed, using static program analysis to provide more precise information than simply syntactic similarity.

PDG-based Techniques: Program Dependency Graph (PDG)-based approaches go a step further in source code abstraction by considering semantic information encoded in a dependency graph that captures control and data flow information. Given the PDG of a subject program, a subgraph isomorphism algorithm is used to find similar subgraphs which are then returned as clones. One of the leading PDG-based clone detection methods finds isomorphic PDG subgraphs using (backward) program slicing. [7][8]There is also a recent PDG-based tool, GPLAG for plagiarism detection.

Hybrids: In addition to the above, there are some clone detection (and plagiarism) techniques for Lisp-like languages [9]. It provides a hybrid approach that combines syntactic techniques (using metrics) and semantic techniques (using call graphs) in combination with specialized comparison functions.

IV. CODE ANALYSIS

Program analysis is used to extract the embedded knowledge in the program. It is divided into 2 parts static code analysis and dynamic code analysis. Static analysis is the terms for simplified analysis wherein the effect of an immediate change to a system is calculated without respect to the longer term response of the system to that change. Such analysis typically produces poor correlation to empirical results. Dynamic analysis is an attempt to take into account how the system is likely to respond to the change. [10]These are used to improve software quality and productivity.

A. Static Code Analysis (SCA)

This code analysis is performed without execution of program; normally analysis is performed with a formal method with human analysis being called code review [11]. SCA is divided into different parts which are shown below in a table 2:

Table 2: Classification of SCA

T1	Identical code fragments
T2	Syntactically identical fragments
T3	Copied modules which is modified in identifier ,types
T4	Two or more modules that are performed same function but written in different places

The advantages are

- It can find weaknesses in the code at the exact location.
- It can be conducted by trained software assurance developers who fully understand the code.
- It allows a quicker turn around for fixes.
- It is relatively fast if automated tools are used.
- Automated tools can scan the entire code base.
- Automated tools can provide mitigation recommendations, reducing the research time.
- It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix.
- Static code analysis limitations:
- It is time consuming if conducted manually.
- Automated tools do not support all programming languages.
- Automated tools produce false positives and false negatives.
- There are not enough trained personnel to thoroughly conduct static code analysis.
- Automated tools can provide a false sense of security that everything is being addressed.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 2, Issue 12, June 2013

- Automated tools only as good as the rules they are using to scan with.
- It does not find vulnerabilities introduced in the runtime environment.

B. Dynamic Code Analysis

This analysis of code is performed by executing program which is build for particular software. The target program is executed with sufficient test. The main objective of dynamic code cloning is to reduce debugging time by automatically pinpointing the cloning. The used tool depends on the type of cloning is wanted. Different types of tools are used for different cloning procedure. They are tokenization tool, parsing tool, data flow analysis and control, match detection tool, automated heuristic. The advantages are

- It identifies vulnerabilities in a runtime environment.
- Automated tools provide flexibility on what to scan for.
- It allows for analysis of applications in which you do not have access to the actual code.
- It identifies vulnerabilities that might have been false negatives in the static code analysis.
- It permits you to validate static code analysis findings.
- It can be conducted against any application.
- Dynamic code analysis limitations are
- Automated tools provide a false sense of security that everything is being addressed.
- Automated tools produce false positives and false negatives.
- Automated tools are only as good as the rules they are using to scan with.
- There are not enough trained personnel to thoroughly conduct dynamic code analysis [as with static analysis].
- It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

V. CONCLUSION

Code clone is an important problem. As a form of reuse, it is usually caused by programmers' copy and paste activities. Although it seems to be a simple and effective method, these duplication activities are usually not documented, which causes a lots of negative effects on the quality of the software, increasing the amount of the code which needs to be maintained, and duplication also increases the defect probability and resource requirements. In this paper I mainly focused on detection techniques and clone analysis methods. This will help for understanding code clones and the different techniques used. The textual approach gives a crude overview of the duplicated code that is quite easy to obtain, so it is most appropriate during problem detection and problem assessment. The token-based approach provides a precise picture of a given piece of duplicated code and is robust against rename operations. Therefore it works best in combination with fine-grained refactoring tools that work on the level of statements. Syntactic techniques are very good at revealing duplicated subroutines, irrespective of small differences, so it works best in combination with refactoring tools that work on the method level.

REFERENCES

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, Comparison and Evaluation of Clone Detection Tools, Transactions on Software Engineering, 33(9):577-591 (2007).

[2] Cory J. Kapser, Waterloo, Ontario, Canada, 2009 toward an Understanding of Software Code Cloning as a Development Practice

[3] Toshihiro Kamiya, Member, IEEE, Shinji Kusumoto, Member, IEEE, and Katsuro Inoue, Member, IEEE CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code IEEE transactions on software engineering ,vol.28, No.7,july 2002

[4] T. Kamiya, S. Kusumoto and K. Inoue, CCFinder: A Multilinguistic, "Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Transactions on Software Engineering, 2008.

[5] I. Baxter, A. Yahin, L. Moura, M. S. Anna, and L. Bier. Clone Detection Using Abstract Syntax Trees. In Proceedings of ICSM. IEEE, 1998.

[6] D. Gayathri Devi, Dr.M.Punithavalli, Comparison and evaluation on matrices based approach for detecting code clone. Indian Journal of Computer Science and Engineering (IJCSE)

[7] J. Krinke, "Identifying Similar Code with Program Dependence Graphs", in Proceedings of the 8th Working Conference on Reverse Engineering, (2001).

[8] R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code", in: Proceedings of the 8th Int. Symposium on Static Analysis, (2001).

[9] Ms.KavithaEstherRajakumari1,Dr.T.Jebarajan Research Scholar, athyabama University, Importance Of String-Based Techniques In Clone Detection Int. J. on Recent Trends in Engineering & Technology, Vol. 05, No. 01, Mar 2011

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

[11] Wikipedia, "Static Code Analysis ", 2012.

AUTHOR'S PROFILE

Mrs.Prajila Prem P.G Student Department of Computer Science at MSRIT Bangalore, INDIA .Presently doing IV th Sem M.Tech in Computer Science and Engineering at MSRIT,Bangalore,INDIA