COMPARISON AND EVALUATION ON METRICS BASED APPROACH FOR DETECTING CODE CLONE

D. Gayathri Devi[1]

[1]*Department of Computer Science, Karpagam University ,*
Coimbatore, Tamilnadu
dgayadevi@gmail.com

Dr.M.Punithavalli[2]

[2] *Director and Head, Sri Ramakrishna Engineering College,*
Coimbatore, Tamilnadu
mpunitha_srcw@yahoo.co.in

Abstract

Many techniques were proposed for detecting code clone. In this paper we have compared the metrics based on speed, quality and their cost as these are the most that affects due to code clone. The strength and weakness of some of the metrics based are compared and evaluated. This paper suggests an approach based on metric and the quality goals are accessed by the use of metrics.

***Keywords*:** Code Clone, Metric, Duplicate Code, Quality.

1. Introduction

Clone detection is concerned with finding similar pattern in source code, interpreting and using them in design, testing and other software engineering problems, presented in [3,4,5].They can be based on text, lexical or syntactic structure, or semantics. 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” conditions.
Various clone detection tools have been proposed and implemented [1] [2] [3] [8] [11] [12], and a number of algorithms for finding clones have been proposed, such as line-by-line matching for an abstracted source program [4], and similarity detection for metrics values of function bodies [12].
Duplicated fragments will be significantly increase the work to be done when enhancing or adapting code [13], and increases the maintenance cost. In this paper, we provide an overall comparison and evaluation of all of the currently available clone detection techniques based on metrics [4] [5].
The rest of the paper is organised as follows. Section 2 discusses the types of clone code using software perspective. Section 3 then defines the related works. Section 4 describes the approaches using metric based clone detection. Section 5 defines the clone detection process. Section 6 is about comparing results based on metric based code clone detection and Section 7 concludes the paper.

2. Types of Clones

Basically clones are code pairs or groups that have the similar functionality. Some codes are syntactically similar, but some are different. Based on syntactic similarity the codes are divided into four types:

2.1. *Extract Clones*

Identical code fragments except for variations in white space, layout, and comments.

2.2. *Renamed Clones*

Syntactically identical fragments except for variations in identifiers, literals, and variable types in addition to Type-1’s variations.

2.3. *Gapped Clones*

Copied fragments with further modifications such as changed, added, or deleted statements in addition to Type-2's variations.

2.4. *Semantic Clones*

Code fragments that perform similar functionality but are implemented by different syntactic variants.

3. RELATED WORKS

Many techniques have been proposed for performance code clone detection, ranging from lightweight line and token-based syntactic techniques to heavier-weight approaches that emphasize semantics, to metrics-based techniques that indirectly measure similarity. Due to limited space, we refer to existing summaries and focus on related work as it pertains explicitly to similarity.
Semantic-based techniques use representations such as abstract syntax trees and program dependence graphs to find semantically identical clones whose original source code may contain extraneous or reordered statement. These techniques cannot quantify similarity as we can.
Li et al. [14] have developed a tool called CP-Miner to detect copy-and-paste related bugs. CP-Miner uses data mining techniques to identify code clones ant he bug detection is implemented as part of the tool. Before passing to bug detection process, code clones detected gone through a series of pruning procedures. CP-Miner has the ability to handle clone that is not exactly the same.
Recently, Jiang et al. [9] developed a tool to detect bugs in code clones and their surrounding code, called context. The clone detection component of the tool is based on DECKARD, a clone detection tool developed by them. Bug caused by renaming inconsistency is called a typr-3 inconsistency in their work. The tool counts the number of unique identifiers in each code fragment within a pair of clones. Different number of unique identifiers is deemed as likely to bug.

4. APPROACH

Clone detection first parses the source code and then performs the program analysis on the parsed code. All similar code segments are identified and then inconsistency detection is performed. The entire procedure is executed and the results are stored in database.

4.1. Approach for Code Clone Detection

Several approaches are used to detect code clone. In textual analysis all types of codes fragments are detected. Each metrics values are stored in a particular database. The input source is identified using metrics and the similarities of code are detected. In scenario based algorithm the two methods are used token-based and line based.
A token-based technique will be more expensive in terms of time and space complexity than a line-based one because a source line contains several tokens.

4.2. Metric Based Code Clone Detection Process

4.2.1. *Metric Based Distance Algorithm*

This algorithm is mainly based on calculating the distance and similarity measures of the code clone. Levenshtein metric is easy to perform a range of query for fragment f and radius r to find all fragments f' to calculate the length len (f), the method range query takes a fragment f as first parameter and the radius as second parameter.
It returns the set of all clones to f. In this algorithm the radius was chosen to be a function of len (f) and a constant parameter distance. Current clone detection distance based methods can find some type 3 clones, but with great restrictions of thresholds and high computational cost.

4.2.2. *Textual Analysis Using Clone Detection*

The combination of textual and metric analysis to detecting all types of code, in a given set of fragment. It parses through the given input source code and identifies the various methods present. Then a built-in hand coded parses the various methods using an island-driven approach. The metrics values the possible potential clone pairs are extracted.
The metrics are computed for each of the methods identified and the values are stored in the database. The various metric values for the code fragment. The descriptive statistics of the metric values obtained for the

various methods. While computed metrics values, the method pairs with equal or similar set of values are identified by comparison of the records in the database.

4.2.3. *Detection Duplicated Code Using Mapping Algorithm*

Mapping is used to define the optimization criterion. The motivation for predicate better is to map reference to candidates that result in a good values as high as possible and in an ok value as high as possible without compromising the good values. Each constituent of better by means of the path the (good, ok) coordinate of the best matching reference takes during the algorithm.
If possible, it moves towards the upper right corner of the good/ok values space. A mapping from candidates to reference has to be established. Each candidate is mapped to the reference that is best matches.

4.2.4. *Metric Based Scenario Comparison Using Code Clone*

Metrics are calculated from names, layout, and expression and (simple) control flow of functions, and a clones is defined as a pair of whole function bodies with similar metrics values. A token-based technique will be more expensive in terms of time and space and complexity than a line based one because source line contains several tokens. Metric-based techniques gather a number of metrics for code fragments and then compare metrics-vectors rather than code or ASTs directly.
First approach uses direct comparison of metric values as similarity at the granularity of begin-end blocks. A modified version of five well known metrics that capture data and control flow properties is used. The second approach uses a dynamic programming (DP) technique to compare begin-end blocks on a statement –by-statement basis using minimum edit distance.

4.2.5. *Detecting structural Clones*

In this approach they consider the internal structure of the code files, in terms of simple clones shared by the files, to decide about their similarity. Structural analysis can establish clone relation among classes that differ in the order of methods in class definition. Structural clone concept may also improve the precision and recall of clone detection in general. Clone detectors usually detect clones larger than a certain thresholds (e.g., clones longer than 5 LOC).
The structure of the file is analyzed fully, it display the function names and counts the total number of functions that are predicted. The size metric is used to finds the number of statements in the file, total number of lines of source code, the size of each file, the largest and smallest file are calculated. The variables are stored, each variable is incremented if found, if there are repetition in the variables, the particular variable is detected.
If similar variables, methods etc are found it is detected and predicted as Simple Clones. The larger program structures, formed by configurations of simple clones, are predicted as Structural clones.

5. Clone detection process

Clone detection first parses the source code and then performs the program analysis on the parsed code. All similar code segments are identified and then inconsistency detection is performed. The entire procedure is executed and the results are stored in database. As first part of the analysis through parsing of source code, they find the similarity and in the second phase the metrics are calculated and finally they are detected.

6. Comparison

The clone deduction algorithm is used to calculating the distance. It is used for Type 3 clone deduction (i.e.) which are similar fragments or some similarity measures. The algorithm behind each task may differ between implementation we can characterize the functional structures of the code fragments with numbers.
The process of clone detection has been divided into a number of phases. These phases include Input & Pre-Processing, Template Conversion, and Metric Computation. The metrics are computed for each of the methods identified and the values are stored in the databases. The various metric values are calculated for the various methods. The method pairs with equal or similar set of values are identified by the comparison of the records in the database.
To detect clones may be affected by this normalization, but to make the comparison easier. Mapping algorithm is used to define optimization criteria. It provides the accurate results in the good or ok values. We can define several measures to determine various aspects of detection quality.
The complexity of the detector implementation, the bulk of which is the normalization, transformation and comparison, depends a great deal on the code representation. This information retrieval approach limits its

comparison to comments and identifiers, returning code fragments as clones when there is a high similarity between the identifiers and comments in them.
On detecting code duplication, the quality of code is improved. This tool detects a significant amount of code duplication. Identification and subsequent unification of simple clones is beneficial in software maintenance. As a structural clone comprises many simple clones, structural clones are bigger in size and smaller in number than simple clones.

6.1. Comparison based on algorithm

Table 1. Algorithm Comparison.

Algorithm	Metric	Speed	Cost	Quality
Metric Based Distance Algorithm	Yes	Very High	High	High
Textual Analysis Using Clone Detection	Yes	Low	Low	Low
Detection Duplicated Code Using Mapping Algorithm	Yes	High	Low	High
Metric Based Scenario Comparison Using Code Clone	Yes	Very High	High	High Uniqueness and low similarity
Detecting Structural Clones	Yes	Low	Low	High

7. Conclusions

This paper suggests an approach to quality based on metric. The use of metric to assess quality is key to our approach. The metric provides to assess quality. We used clone detection and metrics to evaluate quality. We consider this as an indication that metric based approach can be used to assess and assure quality.

Acknowledgments

We would like to thank fellow colleagues for valuable insights and comments that helped to improve this paper. We want to acknowledge my parents and Mr.P.RamKumar, Mr.D.Siddharthan for their grand support and completing the work successfully.

References

[1] Baker.B.S, (1995)" On finding Duplication and Near- Duplication in Large Software Syatem", Proc. Second IEEE Working Conf. On Reverse Eng., pp. 86-95.
[2] Baxter.I.D, a. Yahin, L.Moure, M.Sant' Anna, and L. Bier, "Clone Detection Using Abstract syntax Trees", (1998) Proc. Of IEEE Int'l Conf. on Software Maintenance (ICSM) '98, pp. 368-377, Bethesda, Maryland.
[3] Ducasse.S, M.Rieger, and S.Demeyer. (1999) "A Language Independent Approach for Detecting Duplicated Code", Proc. Of IEEE Int'l Conf. on Software Maintenance (ICSM) '99, pp.109-118. Oxford, England.
[4] Gayathri Devi.D, .Punthavalli.M, (2011) A Detecting structural Clones , In IEEE International Conference on Network Communication and Computer.
[5] Gayathri Devi.D ,.Punithavalli.M (2011) A Method for Detecting Code Clones, In IEEE International Conference on Electronics Computer Technology..
[6] Girard Juergens, J. F,. .Deissenboeck,B.Hummel, ,B.sehaetz,S.Wagner,S.Teuchert ,. (2008) Clone detection in automotive model based development. In proceeding of the international conference on software engineering, IEEE computer society press.
[7] Guo. J and Y.zou, detecting clones in business applications. (2008) In proceeding of the working conference on reverse engineering.
[8] Johnson.J.H, (1993) "Identifying Redundancy in Source Code using Fingerprints", Proc. Of IBM Centre for Advanced Studies Conference (CAS CON) '93, pp. 171-183, Toronto, Ontario.
[9] Jiang.L , G.Misherghi , (2007) Z. Su , DECKARD : Scalable and accurate Tree-based Detection of Code Clones , In ICSE.
[10] Lagu.B, K.Kontogiannis ,M.Balazinska,E.Merlo,M.Dagenais, (2000) Advnced clone analysis as a basic for object oriented syatem refactoring. In proc. Working conference on reverse engineering [WCRE] pages, 98-107, IEEE computer society press.
[11] Lauge.B, E.M.Merlo, J.Mayarand, and J.Hudepohl, (1997) "Assessing the Benefits of Incorporating Function Clone detection in a Development Process", Proc. Of IEEE Int'l Conf. on Software Maintenance (ICSM) ' 97, pp. 314-321, Bari, Italy.
[12] Lablanc.C, J.Mayland, and E.M.Merlo, (1996) " Experiment on the Automatic Detection of Function Clones in a Software Maintenance (ICSM) '96, pp. 244-253, Moterey, California.
[13] Leblanc .C,J.Mayrand, and E.Merlo, (1996) Experiment on the Automatic Detection of Function Clones in a Software Syatem Using Metrics, In ICSM, pp. 244-253.
[14] Z.Li, S.Lu , (2006) CP –Miner. Finding Copy-Paste and Related bugs in Large-Scale Software Code, IEEE TSE.

Authors

1. D.Gayathri Devi M.C.A., M.Phil., She is currently a Assistant Professor Department of computer Science at Sri Ramakrishna College of Arts and Science for Women, Coimbatore, Tamilnadu and pursuing her Ph.D Karpagam University, Coimbatore. She has 7 years of teaching experience and presented 3 papers in National and International Conferences and produced 2 M.Phils so far

2. Dr.M.Punithavalli Ph.D, She is currently a Director and Head, Department of Computer Science at SNS College of arts and Science, Coimbatore ,Tamilnadu She has 21 years of teaching experience and presented National and International Conferences and Produced more than 15 M.Phil and more than 5 Ph.D so far