A Hierarchical Method for Detecting CodeClone

D.GAYATHRI DEVI
Research Scholar, Computer Science Department
Karpagam University
Coimbatore, Tamilnadu, India.
e-mail: dgayadevi@gmail.com

Dr.M.PUNITHAVALLI
Director, Computer Science Department
SNS College of Arts & Science
Coimbatore, Tamilnadu, India.
e-mail: mpunitha_srcw@yahoo.co.in

Abstract - **To improve the efficiency of the software system, the classifiers goes through the process of fine tunning on code clone detection. As code clone affects the software as a major factor in maintenance. This can be accomplished by finding simple clones. Once these simple codes have been fine tuned it can be detected. Because when the size of the simple clones increases in software, the execution time naturally increases. Detecting such clones can help in understanding design of the system for better maintenance and in reengineering for reuse.**

Keywords – Code Duplication, similarity, Code Detection

I. INTRODUCTION

Code clones are a well-known and fine research kind of a bad smell. Code duplication is often cited as one of the major smells in software systems. Code duplication or copying a code fragment and then reuse by pasting with or without any modification is a well known code smell in software maintenance. Clones can be in the form of Clone Pairs or Clone classes or both. These specify the similarity relation between two or more cloned remains. As a result of this that the code that are very similar are called code clones.

A pair of code portion is called a clone pair, if there exists a clone similarity between them. A clone relative holds between two codes if and only if they are of same sequence. The detection of a particular type of clones should be independent of the detection approach. The user identifies the clones that are replicated in software.

Code clone of source codes in software system are one of the major factors in decreasing maintainability. Many duplication code detection methods have been proposed to find code duplication automatically from large scale software. However, it is still hard to find code duplication to improve maintainability because there are many code duplications that should remain.

This paper is structured as follows: Section 2, state the concepts of simple clones, and Section 3, specify the Prediction of code clones, and Section 4, provides the experimental Results and Section 5, concludes the remarks and its future work.

II. METHODOLOGY

A Group of similar code fragments form a simple clone.

A. Hierarchical Method

A classification of data uses metrics values to perform the clone set. The fundamental classification is subdivided into hierarchical by the type of structure obligatory on data.

A hierarchical method creates a hierarchical decomposition of the given set of data objects, where here the data objects refers to file, methods, attributes etc. This is a top down approach; it starts with each object, and finds the similar data objects, until a termination condition holds. The advantage of this hierarchical comes with the cost of effectiveness.

B. Software Measurement

A measurement is defined as "A mapping from empirical world to the formal, relational world, consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute".

C. Software Metric

Size (LOC) – Size of a method is used to evaluate the case of understandability of the code. It is measured by counting the line of code, number of statements and number of blank lines. This metric is used to evaluate the quality attribute like understandability, reusability and maintainability.

III. SIMILAR CODE DETECTION ALGORITHM

To detect exact and similar parts of source code. A pair of similar source code has the same mass within general distance respectively as long as they match together. When distance is one, a pair of code would be exactly matched. Our algorithm starts from making an upturned file. An upturned file includes chunks and corresponding positions. On the other hand, a file has information about position and corresponding chunk. Then we extract file as repeated for n times if we want to extract more than n times repeated similar code. By using a reversed file and a file, it is possible to extract similar parts of source code. With tracing adjoining files sequentially positions of adjacent similar code to adjoining files can be efficiently found from a reversed file.

Evaluated similarity is the ratio of total chunks in the duplicated series to matched chunks. When the foremost

978-1-4244-8679-3/11/$26.00 ©2011 IEEE

series fails to grow, all duplicated series to the leading series are evaluated and stored. The length of duplicated series is relatively small to the whole source code.

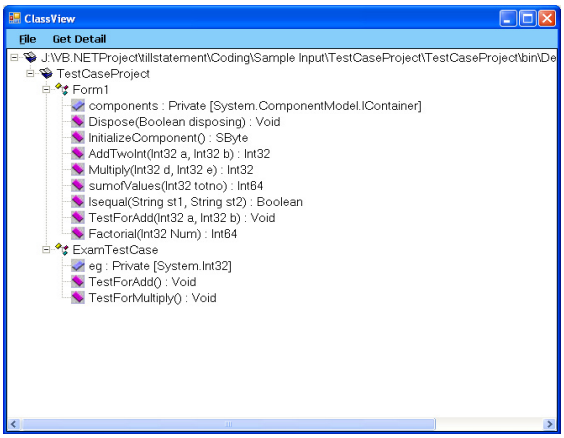
IV. PREDICTING CODE CLONE

The Assembly file is taken as an input, it is partitioned into functions, if the functions are predicted, it displays it the tree form. 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.

V. EXPERIMENTAL RESULTS

To evaluate the hierarchical methods it concentrates on the similarity of the code. This approach depends on the External and Internal criteria. The external criteria include an evaluation based on a predefined structure, which reflects the former information. Internal criteria are based on the evaluation based on measuring the data objects. This then compares their result with the other predicted data objects and predicts the simple clones in the file.

The Assembly file (project) is imported into the test suit environment. On the basis of external criteria, a predefined structure it finds the functions in the assembly file by hierarchical method. By hierarchical method it decomposes file, classes, methods etc.



Secondly, based on internal criteria, it measures the internal objects line by line by lexical analysis (parsing) in the assembly file. It finds all the variables and functions in the project and displays it.

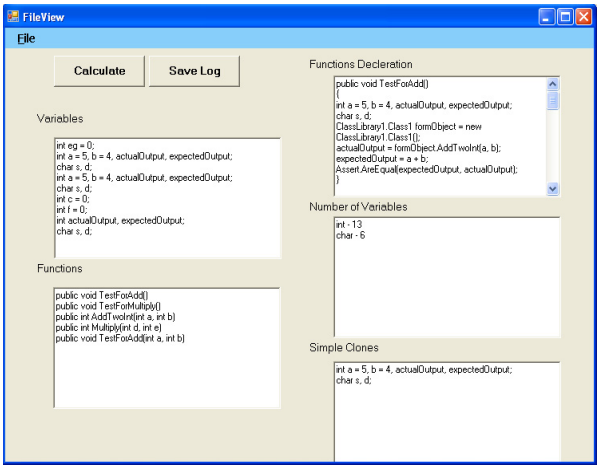


Fig. 2 Function and Variable Display

As a human intuition we use software metric. Metrics is a measure of some property of a piece of software. With the Size metrics, i.e line of code the total number of variables, size of each file in project, total number of lines in each file and from that the largest and smallest file is e predicted.

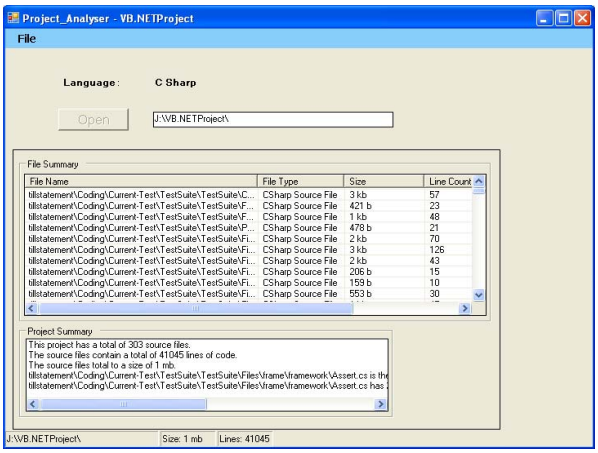


Fig. 3 Finding the Line Count for Each File

The Fig 3 specifies that the TestCaseProject assembly file contains 7 source files; it gives the size of each file in Kb and the total number of lines in each file. It specifies the largest and smallest file among the TestCaseProject.

Finally, each variable is compared with the each predicted variables. It variable predicted is similar to another predicted variable, it is defined to be the simple clones and it is predicted in bold.

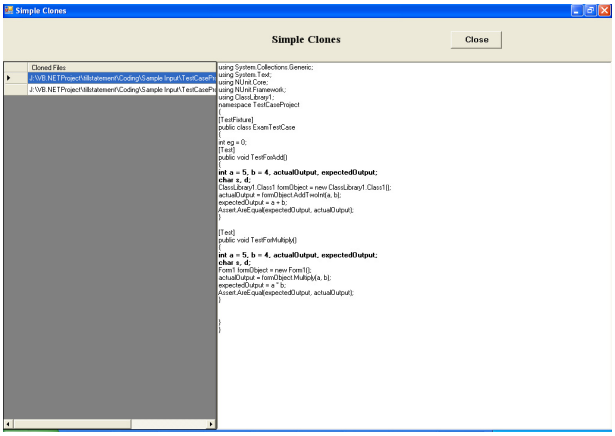


Fig. 4 Simple clone set formed by similar code

In the above diagram int a=5, b=4, actualOutput, expectedOutput; char s,d; forms a simple clone set by code fragments. Here the simple clones are highlighted in bold.

The count of simple clone is found. The Simple clones are obtained in large numbers in each and every file. On detecting simple clones the code becomes more efficient one.

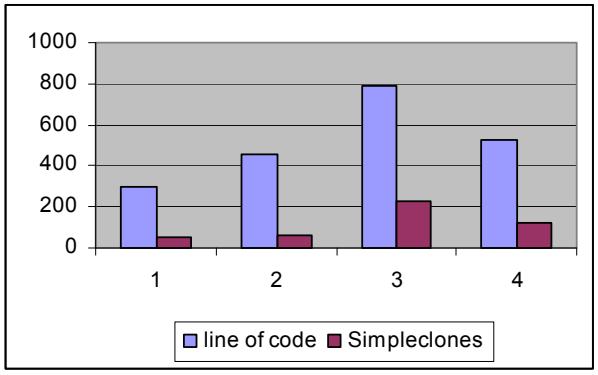


Fig. 5 Line of code and the Simple clones occurred

VI. CONCLUSION

On eliminating duplicate code of source code, it saves comprehension time and space. We believe that our technique is scalable and useful. 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. Our main goal is to identify clones and quantify the amount of similarity present.

REFERENCES

[1]. I. Baxter, A. Yahin, L. de Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In Proc. 14th IEEE International Conference on Software Maintenance, pages 368–377, 1998.

[2]. S. Bellon. Detection of software clones. 2007, http://www.bauhaus-stuttgart.de/clones.

[3]. S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. Transactions on Software Engineering, 33:577–591, 2007.

[4]. P. Bulychev. Duplicate code detection using clone digger. Python Magazine, Sept. 2008.

[5]. P. Bulychev and M. Minea. Duplicate code detection using anti-unification. In Proc. Spring Young Researchers Colloquium on Software Engineering, pages 51–54, 2008.

[6]. W. Evans, C. Fraser, and F. Ma. Clone detection via structural abstraction. In Proc. of the 14th Working Conference on Reverse Engineering, pages 150–159, 2007.

[7]. R. Koschke. Frontiers of software clone management In 24th IEEE International Conference on Software Maintenance, pages 119–128, 2008.

[8]. R. Koschke, R. Falke, and P. Frenzel. Clone detection using abstract syntax suffix trees. In Proc. 13th Working Conference on Reverse Engineering, pages 253–262, 2006.

[9]. Reto Geiger, Evolution impact of code clones, Departmant of Informatics, October 2005.

[10]. C. K. Roy and J. R. Cordy. A survey on software clone detection research. Technical report, Queen's University at Kingston, Ontario, Canada, 2007.

[11]. Richard Wettel and R. Marinescu. Automated Detection of Code Duplication Clusters University of Timisora June, 2004.

[12]. W. Yang. Identifying syntactic differences between two programs. Software Practice and Experience, 21:739–755,1991.

[13]. Fowler M, Beck K, Brant J, OpdykeW, Roberts D. Refactoring: Improving the Design of Existing Code. 1st edn., Addison-Wesley,1999.

[14]. T. Kamiya, S. Kusumoto and K. Inoue. (2002, jul). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Software Eng. 28(7), pp. 654-670.

[15]. C. Kapser and M. W. Godfrey. Cloning considered harmful considered harmful. Presented at WCRE '06: Proceedings of the 13th WorkingConference on Reverse Engineering

[16]. Thummalapenta S, Cerulo L, Aversano L, Di Penta M. An empirical study on the maintenance of source code clones. Empirical Software EngineeringMar2009.

[17]. S. Uchida, A. Monden, N. Ohsugi, T. Kamiya, K. Matsumoto, and H. Kudo. Software analysis by code clones in open source software. The Journal of Computer Information Systems, XLV(3):1–11, April 2005.

[18]. Richard Wettel, "Automated Detection of Code Duplication Clusters" Diploma Thesis, June 2004

[19]. 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, vol. 28, pp. 654--670, July 2002.

[20]. I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, Clone detection using abstract syntax trees, Int. Conf. Softw. Maintenance, 1998.