International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 2 273 - 277

A novel approach for Software Clone detection using Data Mining in Software

G. Anil Kumar
Sr. Asst. Professor
Dept. Of CSE, Mahatma Gandhi Institute of Technology

Abstract **-** The Similar Program structures which recur in variant forms in software systems are code clones. Many techniques are proposed in order to detect similar code fragments in software. The software maintenance is generally helped by maintenance is generally helped by the identification and subsequent unification. When the patterns of simple clones reoccur, it is an indication for the presence of interesting higher-level similarities. They are called as Structural Clones. The structural clones when compared to simple clones show a bigger picture of similarities. The problem of huge number of clones is alleviated by the structural clones, which are part of logical groups of simple clones. In order to understand the design of the system for better maintenance and reengineering for reuse, detection of structural clones is essential. In this paper, a technique which is useful to detect some useful types of structural clones is proposed. The novelty of the present approach comprises the formulation of the structural clone concept and the application of data mining techniques. A novel approach is useful for implementation of the proposed technique is described.

Index Terms—*Design concepts, maintainability, structural clones, restructuring, reengineering.*

*****

1. INTRODUCTION

The program structures which are of considerable size and remarkable similarity are code clones. Many studies have indicated that 20-50 percent of large software systems consist of cloned code [1], [2], [3]. If the location of the clones is known, it helps in understanding and maintaining a program. Refactoring [4] helps in removing the clones i.e. clones are replaced by function calls or macros. Aspect Oriented Programming [5] which is an unconventional metal level technique can be used in order to avoid the harmful effects of clones.

An active area of research is cloning. Many clone detection techniques have been proposed in the literature [1], [6] [7],[8],[9][10]. The major drawback of the present research on code clones is that it focuses more on the fragments of duplicated code and doesn't focus on the aspect that the fragments of duplicated code are possibly part of a bigger replicated program structure.

The larger granularity similarities are called as structural clones. The location of structural clones helps to identify forest from the trees and there is magnificent value for program understanding, evolution, and reuse and reengineering.

The application domain patterns, design technique or mental templates used by the programmers induce the structural clones. In order to solve the similar problems similar design solutions are applied repeatedly. These solutions are generally copied from the code which is existing. The modern component platforms like NET and J2EE encourage architecture-centric and pattern-driven development. This paves way for standardized highly, uniform and similar design solutions. For instance, process flows and interfaces of the components within the system may be similar which results in file or method –level structural clones. Another reason for the higher-level of similarity is the feature combinatory problem [11]. The detection of large-granularity structural clones is really very useful in the reuse context [12]. At the time of creation, the knowledge of structural clones is evident whereas the formal means for the visibility of structural clones in software lacks. During the subsequent software development and evolution, the knowledge of differences among the structural clone instances is implicit and they can be lost easily.

Several attempts have been made to move beyond the raw data of simple clones. In order to enable the user to make sense of cloning information, application of classification, filtering, visualization and navigation have been proposed [13] [14].

The idea of applying a follow up analysis to simple clones' data is explained in this paper. It has been observed that at the core of the structural clones, there are simple clones which coexist and relate to each other in certain ways. This forms the basis of this work on defining and detecting structural clones. A technique to detect some specific types of structural clones from the repeated combinations of collocated simple clones is proposed. A mining based clone detection [15], which is a structural clone detection technique (implemented in C++), can be implemented. The information of simple clones which arise from a clone detection tool enables the structural clone

detection to work. The knowledge of simple clone sets and the location of their instances in programs is only required.

The following are the unique contributions made by the structural clone concept. The advantages of knowing structural clones reach beyond simple clones because structural clones consist of much bigger parts of a program. It is more meaningful to analyst and programmers compared to just similar code fragments. The domain or design concepts which are represented through structural clones help in understanding the program and their detection gives scope for recovery of the design which is not, only practical but also scalable. The representation of repeated program structures of large granularity in a generic form also offers interesting opportunities for reuse [16]. The detection of reuse is useful in the reengineering of legacy systems for better maintenance. If the cloned portions undergo arbitrary changes at the time of evolution, they are scattered in a program. There is every possibility, for this to happen when the code which is plagiarized is purposefully changed to hide cloning. Due to small size, such clones escape detection by simple clone detectors. The detection of structural clones enables the effectiveness of clone detection. This contributes to a more complete picture of the cloning situation.

2. STRUCTURAL CLONES IDENTIFIED

All kinds of large granularity repeated program structures are covered in the concept of structural clones. This novel approach can trace some specific types of structural clones which are listed in Table 1.

Level 1	Repeating groups of simple clones	
	A	In the same method
	B	Across different methods
Level 2	Repeating groups of simple clones	
	A	In the same file
	B	Across different files
Level 3	Method clone sets	
Level 4	Repeating groups of method clones	
	A	In the same file
	B	Across different files
Level 5	File clone sets	
Level 6	Repeating groups of file clones	
	A	In the same directory
	B	Across different directories
Level 7	Directory clone sets	

Table 1. Types of Structural Clones Found by proposed method

TYPES OF STRUCTURAL CLONES

The specific types of structural clones are focused because their detection required only lexical analysis. This makes our method minimally language dependent. The structural clones can be easily detected by well-known data mining techniques. Lastly, these types of clones can be represented in generic form with XVCL

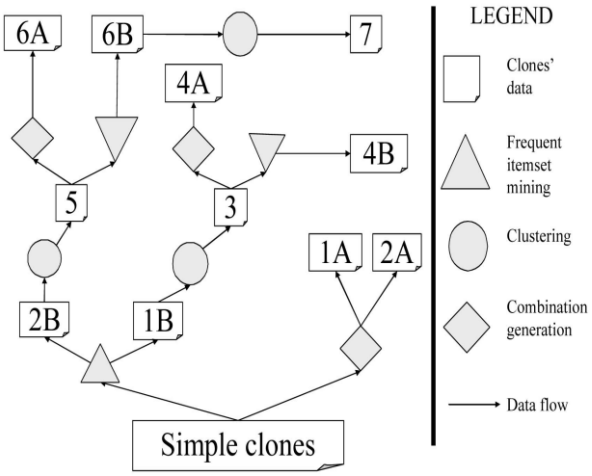

Fig. 3. A hierarchy of structural clones detected by proposed method and the overall detection process.

Fig. 3 depicts the hierarchical process of the detection higher level structural clones given in Table 1 based on the corresponding lower-level clones. The process begins from simple clones which are shown at the bottom of the figure. There are method clone sets (MCSets at level 3), file clone sets (FCSets at level 5), and directory clone sets (DCSets at level 7) which are similar to simple clone sets (SC Sets). These consist of groups of cloned entities at successively higher levels of abstraction. The other types of clones listed in Table 1 include recurring groups of simple clones, method clones, or file clones.

3. STRUCTURAL CLONES

The structural clone detection performed by proposed approach helps to detect simple clones first, and then increases the level of clone analysis to larger similar program structures. Fig 3 shows the overall algorithm for structural clone detection at various levels.

Simple Clone Detection

The output from some simple clone detectors is in the form of clone pairs. The locations of the methods should be provided for the method based structure. There is a possibility of obtaining information directly, if the simple clone detector is based on parsing or lexical analysis. Repeated Tokens Finder (RTF) which is a token based simple clone detector is used by the proposed method as the default front-end tool. The input source code is tokenized into a token string by RTF, from which a suffix-array based string matching algorithm directly computes the SCSets, rather that computing them from the clone pairs. RTF currently supports JAVA, C++, PERL and VB.net. In order

to detect method or function boundaries, RTF performs some simple parsing.

4. Reorganizing the Simple Clone Data

At Levels 1 and 2 Structural clones are found from simple clones by manipulating the data extracted from a software system. We first need to reorganize this data to make it compatible with the input format for the data mining technique that is applied on this data. Depending on the analysis level, we list simple clones for each method or file. The method level analysis only works when we know the method or function boundaries in the system and the simple clones are contained within those boundaries, without straddling them. With this, we get a different view of the simple clones' data, with simple clones arranged in terms of methods or files.

Repeating Groups of Simple Clones

The same data mining technique which is used for "market basket analysis" [17] is applied in order to detect recurring groups of simple clones in different files or methods. This helps in the analysis of finding the items which are usually purchased together by different customers from a departmental store. A list of transactions, each one containing items bought by a customer in that transaction is included in the input database. The output includes groups of items which are most likely to be bought together. The objective is to find all those groups of SCSets.

The returning of many frequent item sets which are subsets of bigger frequent item sets can be done by mining all frequent items sets. "Frequent Closed Item Set Mining (FCIM) [18] is more suitable for our problem. The item sets that are not subsets of any bigger frequent item set are reported.

The input parameters for FCIM are the minimum support count. In this context, it is an indication for the minimum number of files or methods that should contain the detected group of SCSets. The standard algorithms are designed so that the minimum support level for FCIM is adjusted. This is because of the general nature of the FCIM problem. In this case, the support value is coded at 2 so that it will report a group of SCSets because of the significance of its length. The unrestricted gapped clones are level 1-B and 2-B structural clones [19] [20] where number of gaps of arbitrary sizes and ordering are allowed. The repeating groups of simple clones across different files and methods only can be detected because of the limitation of the FCIM technique. In order to detect level 1-A and 2-A structural clones, a simple and straight forward follow-up technique is applied in order to compute the locally repeating groups of simple clones separately.

File and Method Clones

The process of clustering enables the location of File Clone Sets(FCSets at level 5) and method clone sets(MCSets at level 3) from the significant level 2-B and 1-B structural clones, respectively. With this mechanism, there is a possibility of finding groups of highly similar files and methods. The larger granularity similarities that level 1-B or 2-B structural clones with more defined boundaries are indicated by the clusters of similar files and methods.

The well studied technique in the domains of data mining, statistics, biology and machine learning is clustering [17]. The process of grouping the data objects into classes or clusters is clustering. This helps to locate the data objects within a cluster which are highly similar to one another and dissimilar to data objects in other clusters. In this analysis, files or methods are considered as data objects. The detected level 2-B and 1-B structural clones contained in them as having descriptive attribute values.

The average values of two metrics at the structural clone instance level are used to measure the significance of a level 1-B or 2-B structural clone set.

In clustering we cannot expect that the files or methods may become part of some cluster. Many of these files and methods need to be ignored as outliers. This is entirely different from the usual clustering scenarios. The former approach is referred as cluster mining instead of clustering.

Repeating Groups of Method Clones

The repeating groups of method clones across different files to form level 4-B structural clones are found. The detection of repeating groups of method clone across directories is another potentially useful analysis. However, this is not being implemented in proposed method. Apart from this the FCSets based on these repeating groups of method clones can be found. But, the results are expected to be close to the clustering of similar files based on SCSets level 4-A structural clones. The forming of the locally repeating groups of method clones within files is again traced by sorting and brute force combination generation.

File Clones to Directory Clones

We can move on to the level 6 and level 7 structural clones from FCSets. For finding level 6-B structural clones, FCSets play the same role as the SCSets in finding level 1-B and 2-B structural clones. The containers for these file clones are the directories. The transition from level 6-B to level 7 is similar to the transition from level 2-B to level 5 via clustering. Lastly, level 6-A structural clones, representing repeating groups of file clones within

directories, are detected in the same way as Level 4-A structural clones.

Method implementation

The structural clone detection techniques which are presented in this paper are implemented by a novel approach using a mining technique. This method is written in C++. It possesses token-based simple clone detector. The algorithm form is used for FSIM. The proposed method makes use of the STL containers from the standard C++ library in order to manipulate the clone's data. The output from this method is generated in the form of text files. This helps in the visualization of the tool developed in the future which can easily interface with the proposed method. Our experiments taken place on java files with 1500 source files in 150 directories, 62000 LOC and 7250 methods are used to evaluate the performance using different values of minimum clone size. In order to from FC sets and MC sets, a value of 20 token is used for the clustering parameter minLen. The value of 50 percent is used in all other cases. A P-IV computer with 2.6 GHz processor and 1 gb RAM are used for the tests. Two to three minutes were taken for the whole process of finding simple and structural clones.

CONCLUSION

The need to study code cloning at a higher level is emphasized in this paper. The concept of structural clone has been introduced as a repeating configuration of lower level clones. A technique is presented for detecting structural clones the process begins with finding simple clones. By using data mining technique of locating frequent closed item sets and clustering, increasingly higher level similarities are also found. The structural clone detection technique is implemented. The underlying structural clone detection technique can work with the output from any simple clone detector whereas this method can detect simple clones also.

The querying of the database of clones facilitates the analysis of the clones. A mechanism to create a relational database of structural clones data along with a query system to facilitate the user in filtering the desired information. The detection and analysis of similarity patterns is dependent only on the physical location of clones. The system design recovery can perform in a better way with more knowledge of the semantic associations between clones. A clear picture of the similarity in process can be built and automated by using tracing techniques to find associations between classes and methods.

REFERENCES

[1] B.S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," Proc. Second Working Conf. Reverse Eng., pp. 86-95, 1995.

[2] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," Proc. IEEE Int'l Conf. Software Maintenance, pp. 109-118, 1999.

[3] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," Proc. IEEE Int'l Conf. Software Maintenance, pp. 244-254, 1996.

[4] M. Fowler, Refactoring—Improving the Design of Existing Code. Addison-Wesley, 1999.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," Proc. European Conf. Object-Oriented Programming, pp. 220-242, 1997.

[6] I.D. Baxter, A. Yahin, L. Moura, M.S. Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," Proc. IEEE Int'l Conf. Software Maintenance, pp. 368-377, 1998.

[7] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," Proc. IEEE Int'l Conf. Software Maintenance, pp. 109-118, 1999.

[8] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multi- Linguistic Token-Based Code Clone Detection System for Large Scale Source Code," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 654-670, July 2002.

[9] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," Proc. 13th Working Conf. Reverse Eng., pp. 253-262, 2006.

[10] J. Krinke, "Identifying Similar Code with Program Dependence Graphs," Proc. Eighth Working Conf. Reverse Eng., pp. 301-309, Oct. 2001.

[11] D. Batory, V. Singhai, M. Sirkin, and J. Thomas, "Scalable Software Libraries," Proc. ACM SIGSOFT Symp. Foundations of Software Eng., pp. 191-199, Dec. 1993.

[12] U. Pettersson and S. Jarzabek, "Industrial Experience with Building aWeb Portal Product Line Using a Lightweight, Reactive Approach," Proc. European Software Eng. Conf. and ACM SIGSOFT Int'l Symp. Foundations of Software Eng., pp. 326-335, Sept. 2005.

[13] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, "ARIES: Refactoring Support Environment Based on Code Clone Analysis," Proc. Eighth IASTED Int'l Conf. Software Eng. and Applications, pp. 222-229, Nov. 2004.

[14] C. Kapser and M.W. Godfrey, "Toward a Taxonomy of Clones in Source Code: A Case Study," Proc. Int'l Workshop Evolution of Large Scale Industrial Software Architectures, pp. 67-78, 2003.

[15] H.A. Basit, S. Puglisi, W. Smyth, A. Turpin, and S. Jarzabek, "Efficient Token Based Clone Detection with Flexible Tokenization," Proc. European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng., pp. 513-516, Sept. 2007.

[16] S. Jarzabek, Effective Software Maintenance and Evolution: Reused- Based Approach. CRC Press, Taylor and Francis, 2007.

[17] J. Han and M. Kamber, Data Mining: Concepts and Techniques. Morgan Kaufman Publishers, 2001.

[18] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proc. First IEEE ICDM Workshop Frequent Itemset Mining Implementations, Nov. 2003.

[19] J. Krinke, "Identifying Similar Code with Program Dependence Graphs," Proc. Eighth Working Conf. Reverse Eng., pp. 301-309, Oct. 2001.

[20] Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue, "On Detection of Gapped Code Clones Using Gap Locations," Proc. IEEE Ninth Asia-Pasific Software Eng. Conf., pp. 327-336, 2002.