DETECTING FUNCTIONAL SIMILARITY BETWEEN JAVA FILES USING METRICS

Vidhya.K
Assistant Professor, Dept of CSE, Dr.MCET, TamilNadu, India.
E.Mail:vidhyasridhar@drmcet.ac.in

Thirukumar .K
Assistant Professor(Selection Grade), Dept of CSE, Dr.MCET, TamilNadu, India.
E.Mail:Thirukumar@drmcet.ac.in

***Abstract* - Two similar pieces of software code is called clones. Software Developers often copy a section of code, and then paste it with or without modification .Software clone detection is employed to lessen the software maintenance cost and to improve understandability of the system. It also helps in plagiarism detection. Many code clone detection techniques exist and they detect and identify various types of clones. Many such systems primarily focus on the line-by-line comparison method, token-based, PDG detection methods to find out the clones in the system, which are costly in terms of computation time and complexity. Software clones of small size (4-5 line) are called simple clones. Frequent occurrence of simple clones may lead to higher-level clones, for example method clones, file clones etc. These existing systems will not figure out the fragment, which does not have an exact code match but functionally similar to each other. The proposed system captures higher level (File Level) as well as the functional clones (Even with some modification in code). The best part of this system is that, it uses the combination of metric and textual analysis of a source code for the detection of file level similarity in JAVA files. Number of metrics are been identified and values of those metrics are used in detecting similarity between files. The proposed system detects all types of clones with high precision with less complexity.**

***Index Terms* -- Metric Calculation, Metric Comparison, Text-based Comparison, High Precision.**

1. INTRODUCTION

Recent research states that software system unavoidably contain a large amount of similar code, with up to 30 percent of the total amount of code, mostly due to the copy-and-paste programming practice, the framework-based development, or design patterns. These similar code fragments, called code clones, create several difficulties in software maintenance and affect software quality. For example, many bugs occur due to Inconsistent modifications made to cloned code. These bugs could go unnoticed for a long time, reducing the integrity and quality of the software [12].

Duplication of code occurs recurrently during the development of hefty software systems. Code cloning is a form of software reuse, and exists in almost every software project. This informal form of reuse consists in copying, and in due course modifying, a block of existing code that implement a piece of essential functionality. Duplicated blocks are called clones and the act of copying, including slight modifications, is said cloning. [22].

Two code fragments can be similar based on the similarity of their program text which is often the result of copying a code fragment and then pasting to another location or they can be similar in their functionalities without being textually similar [25].Many Techniques have been proposed to identify the simple clones. Repeated occurrence of simple clone may lead to higher level clones such as method, file level and directory clones [11].

As the requirement is growing day by day coding is becoming larger and complex. Extensive software systems are pricey to build and, are even more costly to maintain. Sometimes, developers take uncomplicated way of implementation by copying some fragments of the existing programs and use that code in their work. This type of work is called code cloning.

The following clone types were identified based on the kind of similarity two code fragments can have: [17]

Type I: Identical code fragments except for variations in white space (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.

Type IV: (Functional Similarity) If the functionalities of the two code fragments are identical or similar and referred as Type IV clones. This type detects two or more code fragments that perform the same computation but implemented through different syntactic variants.

The best part of the paper is detecting the file level similarity in JAVA files by combining both the textual analysis and the metric based approach .This is done with the help of a tool designed in JAVA. This paper contains 5 major sections. Section II discusses the related work, Section III describes the implementation of the proposed system, In section IV the results are been discussed the last section concludes the paper.

II. RELATED WORK

Code clones have no consistent or precise definition in the literature. Most consider code clones to be identical or near identical fragments of source code. Software clone detection is an active field of research. The following section describes the different types of approaches; each uses different representation of source code in detecting the clones.

A. Text based technique

It takes each line of source code as code representation. Two code fragments are compared with each other to find the matched sequences of text or strings. When a match is found i.e. two or more code fragments are found to be similar, then they are returned as clone pair by the detection technique[16][17]. It is one of the fastest clone detection approaches. It does not perform any syntactical or semantically analysis on source code

B. Token based technique

Each line of code is converted into a sequence of token. Then the token sequences of lines are compared efficiently through a suffix tree algorithm [11][14]. This technique is slightly slower than text based method, because of the tokenization step. This can easily detect both type 1 and type 2 clones.

C. Abstract Syntax Tree (AST)

Based Technique: Here, the program (source code) is parsed into a parser tree or an abstract syntax tree (AST) with a parser of language of interest. Then, using a tree matching technique, similar sub trees are searched in the tree. When a match is found corresponding source code of the similar sub trees are returned as clone pairs or clone classes [13]. By using AST as code representation gives this technique a better understanding of the system structure. However parsing source file is still a very expensive process on both time and memory.

D. Metric – Based Technique

In Metric based technique, instead of comparing the code directly, different metric of code are gathered and these metrics were compared to detect clones [16][17]. The advantages of technique are it is more scalable and accurate for large software system and it is a straight forward technique.

III. Metric Based Clone Detection System

The objective of the system is to detect the functional similarity between JAVA files using identified metrics. A tool is developed in JAVA for the system and it detects the higher-level clone called file clones in JAVA. The novelty of this system is that it combines both the metric based and text based techniques in detecting the file clones in JAVA. Various metrics have been formed and their values are used in the detection process.

If match exists in the metric values then the textual comparison is performed to confirm the clone pair. Fig 1 shows the architecture of the proposed system. Each part of the system is described in detail

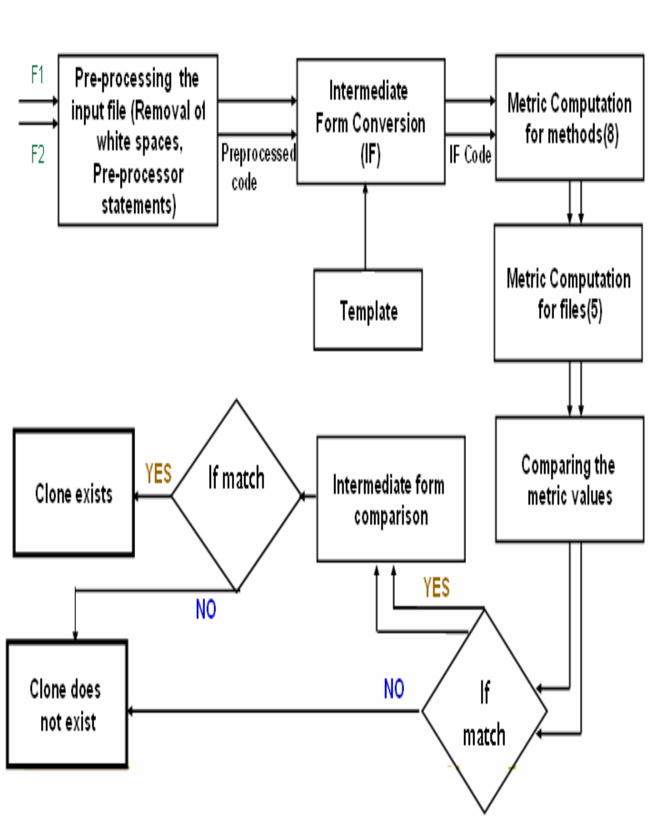
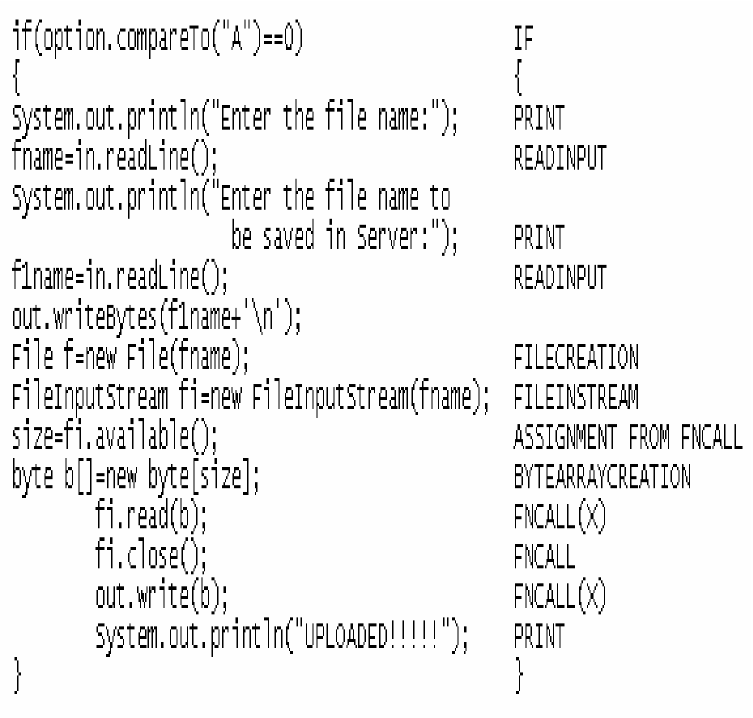

F1
F2
Pre-processing the input file (Removal of white spaces, Pre-processor statements)
Preprocessed code
Intermediate Form Conversion (IF)
IF Code
Metric Computation for methods(8)
Template
Metric Computation for files(5)
Comparing the metric values
Intermediate form comparison
If match
YES
Clone exists
NO
YES
NO
If match
Clone does not exist

Fig 1:.Architecture of the Proposed System

A. **File preprocessing & Transformation**

Source codes of the 2 files are given as the input. In preprocessing the statement which does not have any Effect during analysis like comments, white spaces and pre-processor statements are removed. Source code is re- structured to a standard format.[16] Then the structured code is transformed to a standard intermediary form based on the template. The intermediate form comparison provides better results and precision than comparing the source code as such [10]. This form is used in the textual comparison of the candidates. The following figure shows the template of the given source code.



```
if(option.compareTo("A")==0)                   IF
{                                              {
System.out.println("Enter the file name:");    PRINT
fname=in.readLine();                           READINPUT
System.out.println("Enter the file name to
                   be saved in Server:");      PRINT
f1name=in.readLine();                          READINPUT
out.writeBytes(f1name+'\n');
File f=new File(fname);                        FILECREATION
FileInputStream fi=new FileInputStream(fname); FILEINSTREAM
size=fi.available();                           ASSIGNMENT FROM FNCALL
byte b[]=new byte[size];                       BYTEARRAYCREATION
       fi.read(b);                             FNCALL(X)
       fi.close();                             FNCALL
       out.write(b);                           FNCALL(X)
       System.out.println("UPLOADED!!!!!");    PRINT
}                                              }
```

Fig 2: Template

B. Computing the metric values [12**]**

The methods in the given file are identified by the hand coded parser. Then the metrics are computed for each of the methods identified and the values are stored in a database. Then the metrics are computed for the complete file.

The following table lists the metrics computed for methods[17]

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 Assignment statements

The following lists the metrics computed for file level clone detection

1. No of Effective lines of code
2. Total number of used variables
3. Number of methods defined
4. Total number of function calls
5. Sequence of function call

In a file all the methods defined may be or may not be called and the order in which they are been called also matters. So the metrics are framed in those aspects also

C. Detecting method level similarity [12]

The computed metric values of two files are given as the input for this phase. The Method level metric values are compared. The metric values are stored as numeric values in a data structure. The following table gives a sample which is calculated for a method.

Table 1.Metric Values for a Method

No. of effective lines of code in method	54
No. of arguments passed to the method	2
No. of function calls in the method	1
No. of local variables declared in the method	6
No. of conditional statements in the method	5
No. of looping statements in the method	4
No. of return statements in the method	1
No. of Assignment statements	27

If match exists between Metric values of methods in 2 files, then the clone may exist in the file so it is proceeded to detect the file level clones, otherwise declared as clone does not exist.

The following figure shows the flow chart of the proposed system

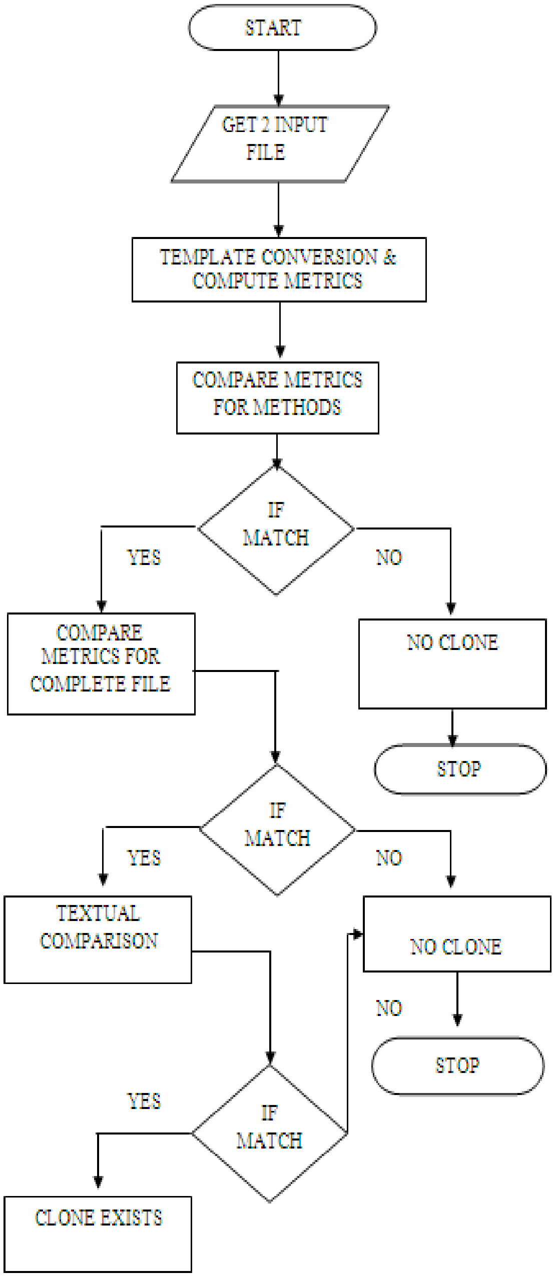


Fig 3: Flow chart of the system

D. Detecting similarities between files

In this phase, the computed metric values of 2 files are given as the input. Because the method level clone exists it cannot be declared that the files are similar. So the file level metric values are compared. The following figure shows the sample metric values calculated for a file.

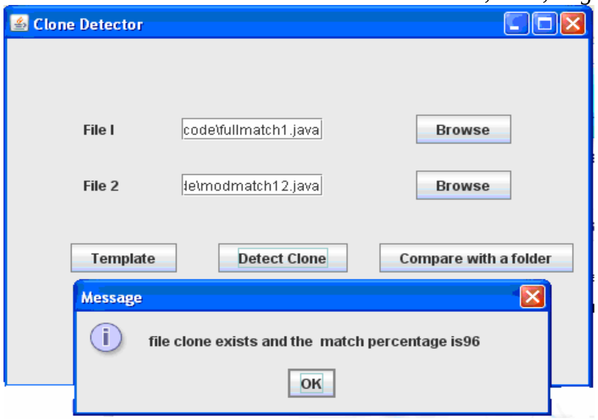
Table 2 Metric Values For A File

No of Effective lines of code	180
Total number of used variables	12
Number of methods defined	3
Total number of function calls	3
Sequence of function call	2 1 3

While comparing the values, similarity between two methods is matched. For example, the 1st method of file 1 may match with 3rd method of file 2. These similarity measures are again stored temporarily and it is used while checking the sequence of function call. Sometimes the same function may be called twice or a function defined may not be called at all. In some cases the number of function call may be same but the order in which they called may be different, which makes the file to produce the different output. All these cases are checked .If match exists it is followed by the textual comparison of the intermediate form code, to confirm the clone pairs; otherwise it is declared that the two files are not similar.

IV. RESULTS AND DISCUSSION

The system has been tested with 2 JAVA files as input and the results are produced based on the similarity between files. A sample result is shown below which states that the file level similarity exists and 96% of similarity exists during the textual comparison



File I code\fullmatch1.java

File 2 le\modmatch12.java

file clone exists and the match percentage is96

Fig 4: The system detects the existence of clone between 2 files

The percentage of the similarity is computed by performing the line by line comparison of the intermediate form of the files and having the following 2 parameters, Number of similar lines and the Total number of lines (Max.No.Lines (file1, file2)).While Comparing the IF of the code, to detect the intentional addition and deletion of the code, line n of the file 1 is first compared with line n of file 2, if no match then it is compared with n+1, n+2 so on up to some threshold level .

The system is also tested by comparing a file with a folder. A folder is taken with 11 JAVA files (Files 6-8 have file level match with the sample file and Files 9,10 and 11 have method level clone and files 1-5 don’t have any match with the sample file) is taken. Then the sample file is compared with all the files in the folder and the results produced are shown below.

Table 3. Result of a file compared with a folder

S.No	**File in the folder**	**Produced Results**
1	File 1	Clone Does not Exist
2	File 2	Clone Does not Exist
3	File 3	Clone Does not Exist
4	File 4	Clone Does not Exist
5	File 5	Clone Does not Exist
6	File 6	File clone exists and the match percentage is 100
7	File 7	File clone exists and the match percentage is 99
8	File 8	File clone exists and the match percentage is 96
9	File 9	Method clone Exists but The number of fn call doesn’t match
10	File 10	Method clone Exists but The sequence of fn call doesn’t matches
11	File 11	Method clone Exists but the file clone doesn’t exist
Total time taken		**3 Seconds**

V. CONCLUSION

This implemented system combines both the text based and metric based techniques. Metric based technique is a straight forward one, so it is a light weight technique. The text based technique is the one which gives high precision. Hence this system is designed to detect the cloned Files in JAVA with high accuracy and reduced complexity. And this system also detects the clones that are not structurally similar but functionally same. This work can also be extended as a generalized tool which accepts different programming language as input and the existence of clone can be detected across the source code of different languages.

REFERENCES

1. Baker,B., “Finding Clones with Dup: Analysis of an Experiment,” IEEE Transactions on Software Engineering, vol. 33, no. 9, pp. 608–621, 2007.
2. Basit, H.A., Stan Jarzabek “A Data Mining Approach for Detecting Higher-Level Clones in Software” IEEE Transactions On Software Engineering, Vol. 35, No. 4, July/August 2009.
3. Baxter,I., Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, “Clone Detection Using Abstract Syntax Trees,” in Proceedings of the 14th International Conference on Software Maintenance (ICSM'98), pp. 368-377, Bethesda,Maryland, November 1998.
4. Brooks, F., 1975. The mythical man-month: essays on software engineering, Reading Mass: Addison-Wesley Pub. Co.5
5. Calefato,F., F. Lanubile and T. Mallardo, “Function Clone Detection in Web Applications: A Semiautomated Approach,” in Journal of Web Engineering, vol. 3, no. 1, pp 3–21, 2004.
6. Ducasse,S., M. Rieger and S. Demeyer, “A Language Independent Approach for Detecting Duplicated Code,” in Proceedings of the 15th International Conference on Software Maintenance (ICSM’99), pp. 109–118, September 1999.
7. Florian Deissenboeck, Benjamin Hummel Elmar Juergens, Michael Pfaehler, Bernhard Schaetz “Model Clone Detection in Practice” IWSC’10, May 8, 2010
8. Gayathri Devi G , Dr.M.Punithavalli “An Effective Software Clone Detection Using Distance Clustering International Journal of Engineering and Technology (IJET), Vol 5 No 1 Feb-Mar 2013
9. Gayathri Devi G , Dr.M.Punithavalli “ Comparison and Evaluation On Metrics Based Approach For Detecting Code Clones” Indian Journal of Computer Science and Engineering (IJCSE) Vol. 2 No. 5 Oct-Nov 2011
10. Gehan M. K. Selim ,King Chun Foo, Ying Zou “Enhancing Source-Based Clone Detection Using Intermediate Representation” 17th Working Conference on Reverse Engineering, 2010
11. Hamid Abduk Basid,Stan JArzabek “A Data Mining Approach for Detecting Higher Level Clones in Software” IEEE Transactions On

Software Engineering, Vol. 35, No. 4, July/August 2009

12. Hoan Anh Nguyen, Tung Thanh Nguyen, Nam H. Pham,Jafar Al-Kofahi, and Tien N. Nguyen “Clone Management for Evolving Software” IEEE Transactions On Software Engineering, Vol. 38, No. 5, September/October 2012

13. Ira D.Baxter,Andrew yashin,Moura l,Sant M,Bier L “Clone Detection Using Abstract syntax Tree” Copyright 1998 IEEE. Published in the Proceedings of ICSM’98, November 16-19, 1998

14. Kamiya,T., S. Kusumoto, and K. Inoue, “CCFinder: A MultiLinguistic Token-Based Code Clone Detection System for Large Scale Source Code,” IEEE Trans.Software Eng., vol. 28, no. 7, pp. 654-670, July 2002.

15. Kamiya,T.,Shinji Kusumoto, and Katsuro Inoue,“A Token-based Code Clone Detection Tool-ccfinder and its empirical evaluation,” Technical report, 2000.
16. Kodhai.E, Kanmani.S, Kamatchi.A, Radhika.R, Detection of Type-1 and Type-2 Clone Using Textual Analysis and Metrics in ITC, 2010.
17. Kodhai.E, Perumal.A, and Kanmani.S, ”Clone Detection using Textual and Metric Analysis to figure out all Types of Clones” International Journal of Computer Communication and Information System(IJCCIS)- Vol2. No1. ISSN: 0976–1349 July – Dec 2010.\
18. Kodhai.E,V.Vijayakumar,G.Balabaskaran, T.Stalin, B.Kanagaraj “Method Level Detection and Removal of Code Clones in C and JAVA Programs Using Refactoring” International Journal of Computer Communication and Information System (IJCCIS) – Vol2. No1. ISSN: 0976–1349 July – Dec 2010

19. Komondoor,R., and Susan Horwitz, “Using Slicing to Identify Duplication in Source Code,” in Proceedings of the 8th International Symposium on Static Analysis (SAS'01), Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.

20. Krinke,J., “Identifying Similar Code with Program Dependence Graphs,” in Proceedings of the 8th Working Conference of Reverse Engineering, pp. 301- 309, Stuttgart, Germany, October 2001.
21. Lakhotia,A., Junwei Li, Andrew Walenstein, and Yun Yang,” Towards a clone detection benchmark suite and results archive,” in Proceedings of the 11th IEEE International Workshop on Program Comprehension, pp. 285, Washington, DC, USA, 2003.
22. Lanubile .F, Mallardo.T., “Finding Function Clones in Web Applications” Proceedings of the Seventh European Conference On Software Maintenance And Reengineering (CSMR’03), 2003.
23. Mayrand,J., C. Leblanc and E. Merlo, “Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics,” in Proceedings of the 12th International Conference on Software Maintenance (ICSM’96), pp. 244–253, Monterey, CA, USA, November 1996.
24. Merlo,E., “Detection of Plagiarism in University Projects Using Metrics based Spectral Similarity,” in the Dagstuhl Seminar: Duplication, Redundancy, and Similarity in Software, 2007.

25. Prabhjot Kaur*, Harpreet Kaur, Rupinder Kaur,”Comarison of Clone Detection Tools:CONQAT and Solid SDD” International Journal of Advanced Research in Computer Science and software Enginering” 2012.

26. Roy, C.K. . and J.R. Cordy, “A Survey on Software Clone Detection Research, Queen’s School of Computing Technical Report No. 2007-541, vol.115, September 2007.

27. Selim M.K., King Chun Foo “Enhancing Source-Based Clone Detection Using Intermediate Representation” 17th Working Conference on Reverse Engineering, 2010

Vidhya K.: Assistant Professor in Dr.Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India, Interested in Software Engineering and Mining

Thirukumar K. Assistant Professor (SG)in Dr.Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India, Interested in Databases and Mining