Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Detection of Potential Clones from Software using Metrics

Geetika*, Rajkumar Tekchandani
CSED, Thapar University,
Patiala, India

Abstract— In code cloning, we copy a chunk of code from one part of the software and then paste it with or without doing some amendment into other part of the software. Although it is easy to do coding using code cloning but at the same time code cloning may cause several problems. It causes a lot of maintenance related problems for softwares. In order to deal with the problems caused by cloning, clones need to be detected from software. The process of identifying code clones from software is called clone detection. There are several existing techniques for detecting clones that give us quite good results. But these techniques consume a lot of time and are very complex, if we apply them on very large softwares. In this paper, an approach to detect potential clones from software is presented. Potential clones are those parts of the code which are the candidates for clones but are not necessarily clones. This approach is quite simple and can be used to reduce the complications with other techniques. The detection approach explained in this paper gives results on the basis of method level metrics extracted from source code. A tool SourceMonitor is used to calculate the required method level metrics. After getting the required metrics, they are compared to detect the potential clones. The result of applying this potential clone detection to a chat server system developed in java language is shown as example.

Keywords— Code cloning, Clone detection, Software metrics, Potential clones, Clone pairs, Clone classes

I. INTRODUCTION

In code cloning, we copy a chunk of code from one part of the software and then paste it with or without doing some amendment into other part of the software. The chunk of code which is copied with or without amendments is called code clone as shown in Fig. 1. Code cloning is a frequent activity throughout the development phase of software. According to earlier researches about 7% to 23% of code in a software system is cloned code [1]. Although it is easy to do coding using code cloning as it gives us opportunities to reuse the code but at the same time code cloning may cause several problems. It causes a lot of maintenance related problems for softwares because if there is a fault in cloned code, then we have to discover and correct the same fault from the clones of that code in a consistent way. In order to deal with the problems caused by cloning, clones need to be detected from software. The process of identifying code clones from software is called clone detection.

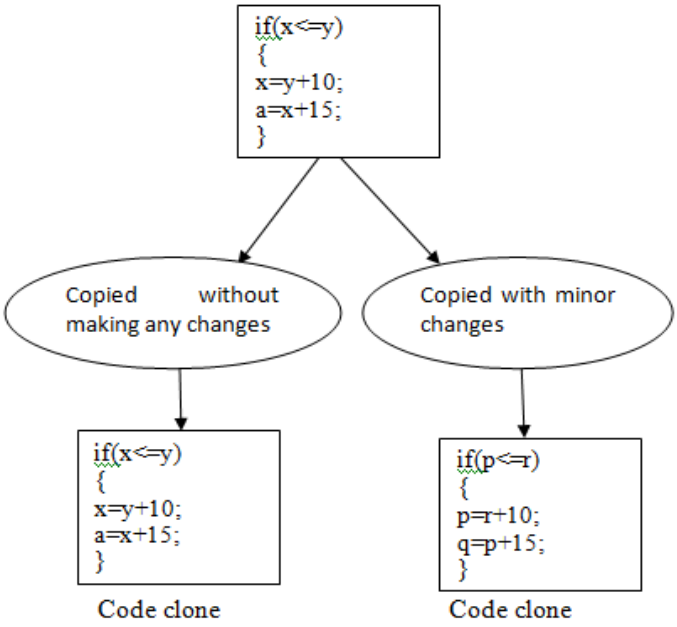


Fig. 1 Code with clones

Some terms related to code cloning include clone pair or clone class. Many clone detection tools present outcomes as clone pairs or clone classes. If there is an equivalence relation between two code segments, then they form a clone pair. Clone class is defined as a collection of similar code segments. Each code segment in a clone class forms a clone pair with other code segments of that class.

Clone detection techniques can be classified into 6 categories: text based techniques, token based techniques, abstract syntax tree (AST) based techniques, program dependence graph (PDG) based techniques, metrics based techniques, and hybrid techniques. Text based techniques ([2],[3]) are the most basic techniques of clone detection that detect clones by comparing code line by line in the form of strings. Token based techniques ([4], [5], [6]) detect clones by comparing code line by line in the form of tokens. AST based techniques ([7], [8], [9], [10]) change the code into AST and then compare the different parts of the AST to find clones. In PDG based techniques ([11], [12]), code gets converted to PDG and then like subgraphs are tracked to find clones. In metric based techniques ([13], [14], [15], [16]) different metrics of code are calculated and then compared for finding clones. The proposed approach is based on metrics to find potential clones. Hybrid techniques ([17], [18], [19]) are a combination of two or more above defined techniques.

The objective of this paper is to present an approach to detect potential clones in a software system. Potential clones are those parts of the code which are the candidates for clones but are not necessarily clones. The detection approach explained in this paper gives results on the basis of method level metrics extracted from source code. A tool SourceMonitor [20] is used to calculate the required method level metrics. Firstly, metrics are calculated using this tool and then the resulting metrics are exported as a CSV (Coma Separated Value) file. The CSV file is then stored into the database and the metrics are then compared to detect potential clones. This approach provides an efficient way of detecting clones. Using this technique, we don't need to apply clone detection on whole software, but only in that part of software in which potential clones are detected. The next section explains the proposed approach, its implementation and result.

II. PROPOSED APPROACH, IMPLEMENTATION AND RESULTS

The proposed approach is a language independent technique to find out potential clones from software.The detection approach explained in this report gives results on the basis of method level metrics extracted from source code. A tool SourceMonitor [9] is used to calculate the required method level metrics. Firstly, metrics are calculated using this tool and then the resulting metrics are exported as a CSV (Coma Separated Value) file. The CSV file is then stored into the database and the metrics are then compared to detect potential clones. The result of applying this potential clone detection approach to a chat server system developed in java language is provided as example. The metrics which are determined by using the tool are in the form of file level metrics and method level metrics as described below:

A. File Level Metrics

1) *Lines:* This metric includes the total number of physical lines in a source file. If ignore blank lines option is set for a project, then this metric will count only those source lines that contain source code text.
2) *Statements*: It includes all the computational statements. Branches such as if, for and while, all attributes and expression control statements such as try, catch and finally are counted as statements.
3) *Branches*: This metric include branch statements such as if, else, switch, case, default, for, do, while, break and continue.
4) *Calls:* It includes total number of calls to other methods or functions found inside each method or function of the source file.
5) *Comments:* It includes those lines of the file that contain comments.
6) *Classes:* It includes the total no. of classes found the source file.
7) *Methods/class:* This metric gives the total number of methods in a project divided by the total number of classes.
8) *Average Statements/Method:* The total no. of statements found inside methods found in a file divided by the number of methods found in the file.
9) *Maximum Complexity:* The complexity value of the most complex method in a file.
10) *Maximum Block Depth:* is the maximum nested block depth level found in the file.
11) *Average Complexity:* This metric is a measure of the overall complexity measured for each method in a file. It is computed as arithmetic average of all complexity values measured for a file.

B. Method Level Metrics

1) *Complexity:* It is equal to one plus the total number of branch statements in the method. In case of switch, complexity is one more than the number of cases in the switch block.
2) *Statements:* The total number of statements found inside each method or function.
3) *Maximum Block Depth:* The maximum nested block depth level found within each method or function. Nested block depth is depth of nested statement blocks.
4) *Calls:* The total number of calls to other methods or functions found inside each method or function. This is also called fan out.

The implementation part is shown below. Fig. 2 shows the first page of potential clone detector and here we need to enter the name of the CSV (Coma Separated Values) file that contains the method level metrics detail of the project. This potential clone detector is implemented in PHP.

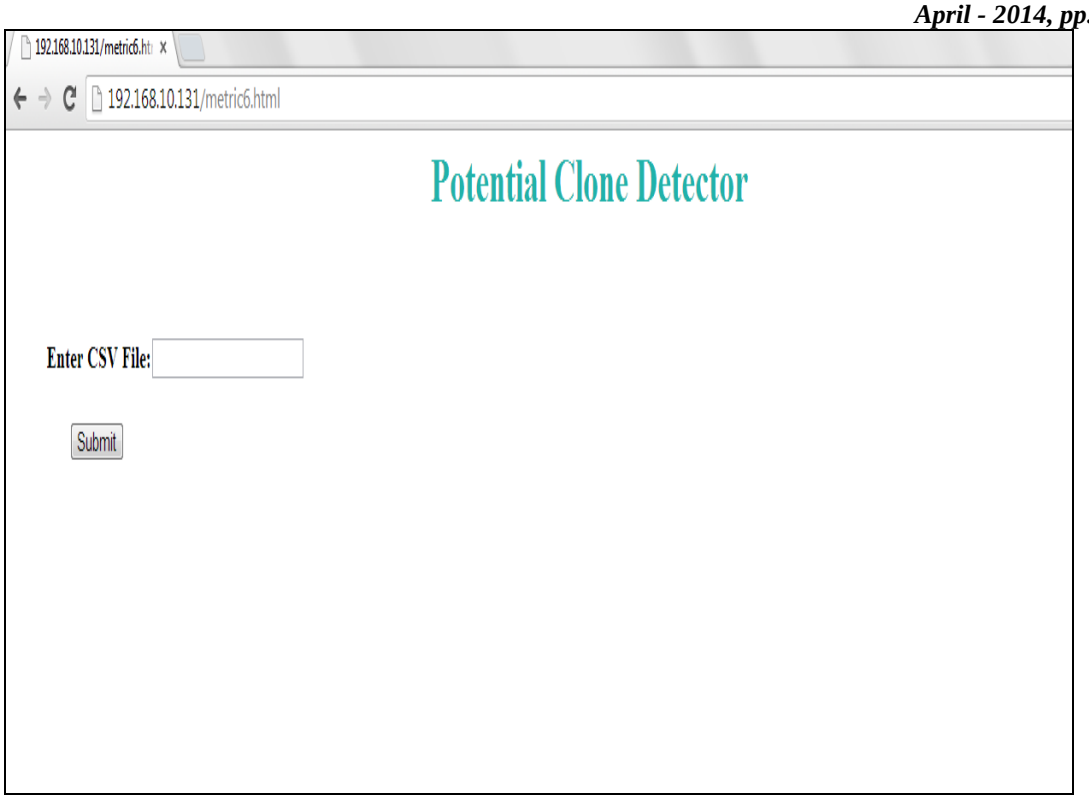
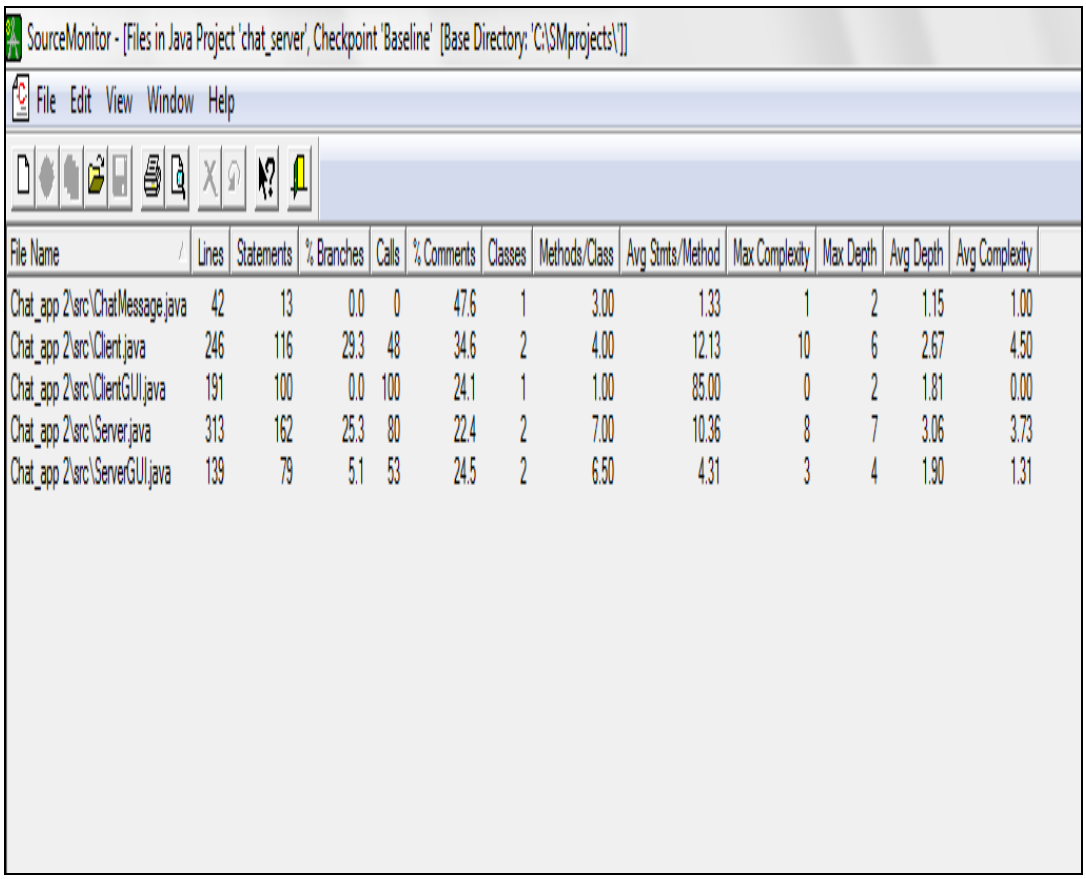


Fig. 2 Startup page

Now to generate the CSV file containing the metrics detail, we have used the tool SourceMonitor [20] that provides a provision to export the metrics detail as a CSV file. Using this tool, we got the file level metrics and method level metrics for the project chat server as shown in Fig. 3. The method level metrics are exported as a CSV file and the name of the CSV file is given as input to the potential clone detector shown in Fig. 2. After we submit the CSV file we get the method level metrics detail of the project in the form a table as shown in Fig. 4.



File Name	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Chat_app 2\src\ChatMessage.java	42	13	0.0	0	47.6	1	3.00	1.33	1	2	1.15	1.00
Chat_app 2\src\Client.java	246	116	29.3	48	34.6	2	4.00	12.13	10	6	2.67	4.50
Chat_app 2\src\ClientGUI.java	191	100	0.0	100	24.1	1	1.00	85.00	0	2	1.81	0.00
Chat_app 2\src\Server.java	313	162	25.3	80	22.4	2	7.00	10.36	8	7	3.06	3.73
Chat_app 2\src\ServerGUI.java	139	79	5.1	53	24.5	2	6.50	4.31	3	4	1.90	1.31

Fig. 3 File level metrics

192.168.10.131/metric7.ph ×

192.168.10.131/metric7.php

id	Project_name	checkpoint_name	created_on	filename	method	complexity	statements	depth	calls
1	chat	Baseline	17 Dec 2013	Chat_app 2srcChatMessage.java	ChatMessage.ChatMessage()	1	2	2	0
2	chat	Baseline	17 Dec 2013	Chat_app 2srcChatMessage.java	ChatMessage.getMessage()	1	1	2	0
3	chat	Baseline	17 Dec 2013	Chat_app 2srcChatMessage.java	ChatMessage.getType()	1	1	2	0
4	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.Client()	1	4	2	0
5	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.Client()	1	1	2	1
6	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.disconnect()	8	14	3	4
7	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.display()	3	4	2	2
8	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.main()	10	36	5	18
9	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	ListenFromServer.run()	7	13	6	6
10	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.sendMessage()	2	4	3	2
11	chat	Baseline	17 Dec 2013	Chat_app 2srcClient.java	Client.start()	4	21	3	15
12	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	ClientThread.ClientThread()	3	13	4	10
13	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	ClientThread.close()	7	13	4	3
14	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	Server.display()	1	2	2	5
15	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	show().main()	5	16	5	5
16	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	show().remove()	3	5	4	3
17	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	ClientThread.run()	8	5	7	4
18	chat	Baseline	17 Dec 2013	Chat_app 2srcServer.java	Server.Server()	1	4	2	2

Fig. 4 Tabular view of method level metrics

When we click the Detect_Clones button, the metrics values shown in Fig. 4 are compared and we get the resulting clone classes as shown in Fig. 5 and Fig. 6. Each clone class consist of a set of potential clone methods and the name of the file in which these methods exist.

192.168.10.131/clone5.php

CLONE_CLASS 1

1:

METHOD = ChatMessage.getMessage()
IN FILE Chat_app 2srcChatMessage.java

2:

METHOD = ChatMessage.getType()
IN FILE Chat_app 2srcChatMessage.java

CLONE_CLASS 2

1:

METHOD = Client.Client()
IN FILE Chat_app 2srcClient.java

2:

METHOD = Server.Server()
IN FILE Chat_app 2srcServer.java

Fig. 5 Resulting clone classes


```
3:

METHOD = ServerGUI.main()
IN FILE Chat_app 2srcServerGUI.java


CLONE_CLASS 3

1:

METHOD = ServerGUI.appendEvent()
IN FILE Chat_app 2srcServerGUI.java

2:

METHOD = ServerGUI.appendRoom()
IN FILE Chat_app 2srcServerGUI.java


CLONE_CLASS 4

1:

METHOD = ServerGUI.windowActivated()
IN FILE Chat_app 2srcServerGUI.java

2:
```

Fig. 6 Resulting clone classes

III. CONCLUSIONS AND FUTURE SCOPE

The proposed approach detects potential code clones on the basis of metrics comparison. In this approach the potential code clones at the method level are detected. This approach is quite simple and can be used to reduce the complications with other techniques and provides an efficient way of detecting clones. Using this technique, we don't need to apply clone detection on whole software but only on that part of software in which potential clones are detected. In future this technique can be further extended to find more number of metrics on the basis of which we are finding the potential clones so that we can get more accurate results and the output of potential clone detector can be integrated with other clone detection approaches to confirm whether the detected potential clones are actually clones or not.

ACKNOWLEDGMENT

We would like to thank all those who provide us the opportunity to write this paper. We thank thapar university for giving us labs to implement our task and all the faculty members who gave support to us.

REFERENCES

[1] B. S. Baker, "On finding duplication and near duplication in large software systems," in *Proc. 2nd Working Conference on Reverse Engineering*, 1995, pp. 86-95.

[2] S. Ducasse, M. Rieger and S. Demeyer, "A language independent approach for detecting duplicated code," in *Proc. ICSM'99*, 1999, pp. 109–118.

[3] J. H. Johnson, "Identifying redundancy in source code using fingerprints," in *Proc. CASCON: Software engineering*, 1993, pp. 171-183

[4] T. Kamiya, K. Inoue and S. Kusumoto, "CCFinder: A multilinguistic token based code clone detection system for large scale source code," *IEEE Transaction on Software Engineering*, vol. 28, no. 7, July 2002.

[5] Z. M. Jiang, Hassan and A.E, "A framework for studying clones in large software systems," in *Proc. 7th IEEE International Working Conference on SCAM*, 2007, pp.203-212

[6] H. A. Basit and S. Jarzabek, "Efficient token based clone detection with flexible tokenization," in Proc. Sixth joint meeting of the European Software Engineering Conference and the ACM SIGSOFT, 2007, pp. 513-516.

[7] L. Jiang, D. Misherghi, Z. Su and S. Glondu, "DECKARD: Scalable and accurate tree based detection of code clones," in *Proc. 29th international conference on Software Engineering*, 2007, pp. 96-105.

[8] W. Yang, "Identifying syntactic differences between two programs," *Software Practice and Experience*, vol. 21, pp. 739 – 755, June 1991.

[9] R. Koschke, P. Frenzel and R. Falke, "Clone detection using Abstract Syntax Trees*," in Proc. thirteenth Working Conference on Reverse Engineering*, 2006, pp. 253-262.

[10] V. Wahler, D. Seipel, J. Gudenberg and G. Fischer, "Clone detection in source code by frequent itemset techniques," in *Proc. SCAM*, 2004, pp. 128–135.

[11] R. Komondoor and S. Horwitz, "Using slicing to identify duplication in source code," in *Proc. 8th International Symposium on Static Analysis*, 2001, pp. 40-56.

[12] C. Liu, C. Chen, J. Han and P. Yu, "GPLAG: Detection of software plagiarism by program dependence graph analysis," in *Proc. 12th international conference on Knowledge discovery and data mining*, 2006, pp. 872-881.

[13] J. Mayrand, C. leblanc and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *Proc. ICSM*, 1996, pp. 244-253.
[14] K. Kontogiannis, "Evaluation experiments on the detection of programming patterns using software metrics," in *Proc. WCRE*, 1997, pp. 44-54.
[15] J. F. Patenaude, E. Merlo, M. Dagenais and B. Lague, "Extending software quality assessment techniques to Java systems," in *Proc. 7th International Workshop on Program Comprehension* , 1999, pp. 49-56.
[16] Abd-El-Hafiz and S. K, "A metrics-based data mining approach for software clone detection," in Proc. IEEE 36th annual *Computer Software and Applications Conference* , 2012, pp. 35-41.
[17] R. Koschke, R. Falke and P. Frenzel, "Clone detection using abstract syntax suffix trees," *in Proc.13th Working Conference on Reverse Engineering (WCRE06*), 2006, pp. 253-262.
[18] A.M. Leitao, "Detection of redundant code using R^2D^2," *Software Quality Journal*, vol. 12, pp. 361-382, 2004.
[19] R. Tairas, J. Gray, "Phoenix based clone detection using suffix trees," *in Proc. 44th annual Southeast regional conference*, 2006, pp. 679- 684.
[20] The SourceMonitor Homepage. [Online]. Available: http://www.campwoodsw.com

