International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Performance Evaluation of Clone Detection Tools

Shilpa Dang[1], Shahid Ahmad Wani[2]

[1]Assistant Professor, Department of MMICT&BM, Maharishi Markandeshwar University Mullana, Amballa, Haryana, India

[2]Research Scholar, Department of MMICT&BM, Maharishi Markandeshwar University Mullana, Amballa, Haryana, India

Abstract: ***Code clones are code portions which are similar in syntax or semantics. Copy-paste activity is the main reason for introducing clones in a software system. These clones must be identified and removed from a system to improve the quality of a software system. Various clone detection tools have been proposed that provide assistance to professionals in identifying code clones and once identified clones can be removed from software systems. These clone detection tools implement different clone detection techniques and have different approach in detection of clones. This study presents an investigation of clone detection tools to understand the performance of each tool and to judge the usefulness and accuracy of clone detection tools. This study can be helpful while making a selection of a particular tool for detection of clones in a software system.***

Keywords: Abstract Syntax Tree (AST), Program Dependence Graph (PDG), String Matching, Precision and Recall.

1. Introduction

In software programs a code segment which is similar to another code segment is known as code clone. The main reason for introducing clones in software systems is the commonly practiced copy-paste activity. There are other reasons through which clones can also exist in system such as risk avoidance, accidental cloning, etc. It has been agreed that clones have a serious effect on software system. Shahid et. al. performed an industrial study that and found that clones have a harmful impact on software quality [1]. Therefore it is significant to identify and remove clones from a software system. There are various clone detection techniques and their corresponding tools available in literature proposed by the renowned researchers which assist people to identify clones in software systems and once identified clones can be removed from these systems by source code refactoring to improve quality of software system. Refactoring is the process of changing a software system to improve its internal structure without modifying the external behavior of source code. The clone detection tools available are based on different techniques and thus have a distinct approach in finding code clones [2]. For example, a clone detection tool which is based on text-based technique can only detect clones that are similar in syntax, a tool which uses token based approach can identify modified copy pasted code and an Abstract Syntax Tree (AST) based tool can identify variations in variable names and identifiers of the similar code. Each tool has its own advantages and limitations. Therefore in order to find the best tool for a particular purpose of interest evaluation of clone detection tools is important. These tools were evaluated and the results found are presented in this paper.

2. Clone Detection Tools

In literature many clone detection tools are available which are used to detect clones in software systems. These clone detection tools implement various clone detection techniques such as Abstract Syntax Tree (AST), Program Dependence Graph (PDG), code metrics, program tokens, visualization and query based techniques which provide an automated assistance to identity code clones in source code. In spite of great success of clone detection tools, little work has been done to present the comparison of these tools. The comparison of these tools can help software professionals in making the right decision while selecting a tool of their interest. Table 1 provides a list of few clone detection tools available in literature and presents the significant detail about each tool such as the author who proposed the tool, language supported by tools, technique implemented and application field of tools.

Table 1: List of Clone Detection Tools

Tool	*Proposed By*	*Language Supported*	*Technique*	*Application*
CloneDr	Baxter et. al., [3]	C, C++, Java and Cobol	Abstract Syntax Tree Method	Clone Detection
CCFinder	Kamiya et. al., [4]	C, C++, Java	Token Based Method	Clone Detection
CP-Miner	Li et. al., [5]	C, C++, Java	Frequent Subsequent Mining	Clone Detection and copy-pasted bug identification
Bauhaus	Bellon [6]	C, C++, Java	Abstract Syntax Tree	Clone Detection
Coogle	Sager et. al., [7]	Java	Abstract Syntax Tree	Finding identical java class
Deckard	Jiang et. al. [8]	C, Java	Tree Matching, Euclidean space	Clone Detection
CCFnderX	Kamiya et. al., [9]	C, C++, Java, COBOL, VB, C#	Token Based Approach	Clone Detection
PMD	Sourcefourge community [10]	Java, C, C++, JSP, Ruby, PHP, PLSQL etc	String Matching	Clone Detection
PDG-Dup	Komondor et.al., [11]	C,C++	Program Dependence Graph	Clone Detection
Duplix	Krinke et. al., [12]	C	Program Dependence Graph	Clone Detection

3. Evaluation of Tools

In Table 1 a total of 10 clone detection tools are listed, however this study investigated four clone detection tools which are CP-Miner, PMD (Programming Mistake Detector), CCFinderX and Bauhaus. The reason for selecting these tools is that they implement different techniques for clone detection and thus produce distinct results.

In this section the results of an experiment conducted with the four clone detection tools are presented. Each one of the four clone detection tools was run with an open source software program to evaluate them. The tools were evaluated based on the factors like number of clones detected, types of clones identified by the tools, precision and recall of the clone detection tools. Precision means that tool should be good enough so that it detect less number of false positives i.e. the tool should find duplicated code with higher precision and recall means that the tool should be capable of locating and finding most (or even all) of the clones of a system of interest. This study used the source code of EIRC (Eteria Internet Relay Chat) program as a subject system. It is written in java and contains 65 files with 11 thousand lines of code.

3.1 Number of Clones Identified

The tools did not find the same number of clones. Table 2 shows the number of clones identified by each tool. The distinction in the number of clones detected is based on the type of clones a tool can identify. A tool that cannot detect type-3 clones will identify less number of clones.

From table 2 it can be seen that PMD identified largest number of clones 957, however CP-Miner identified the least number of clones 783. It can be seen that there is less difference in the detection of clones by Bauhaus and CP-Miner, as Bauhaus detected only 14 clones more than CP-Miner and thus these two tools are comparatively similar in number of identified code clones. CCFinderX detected 26 clones more than Bauhaus and 40 number of clones more than CP-Miner. If number of clones detected is considered as the base of comparison among these tools then PMD is found to be the best tool. Figure 1 shows the percentage of clones identified by each tool.

3.2 Type of Clones Identified

Code clones are classified in four different types which are type-1, type-2, type-3 and type-4. A brief description of each type is given as under:

- **Type-1:** Identical code fragments except small variations in white space, layout, and comments.
- **Type-2:** Syntactically identical code fragments except for variations in literals, identifiers, types, layout, comments and whitespaces.
- **Type-3:** These are copied fragments with additional modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, layout, comments and whitespaces.

Table 2: Number of Clone Identified by Tools

Tool	Number of Clones Identified	Percentage of Clones Identified
PMD	957	7.36%
CCFinderX	823	6.33%
CP-Miner	783	6.02%
Bauhaus	797	6.13%

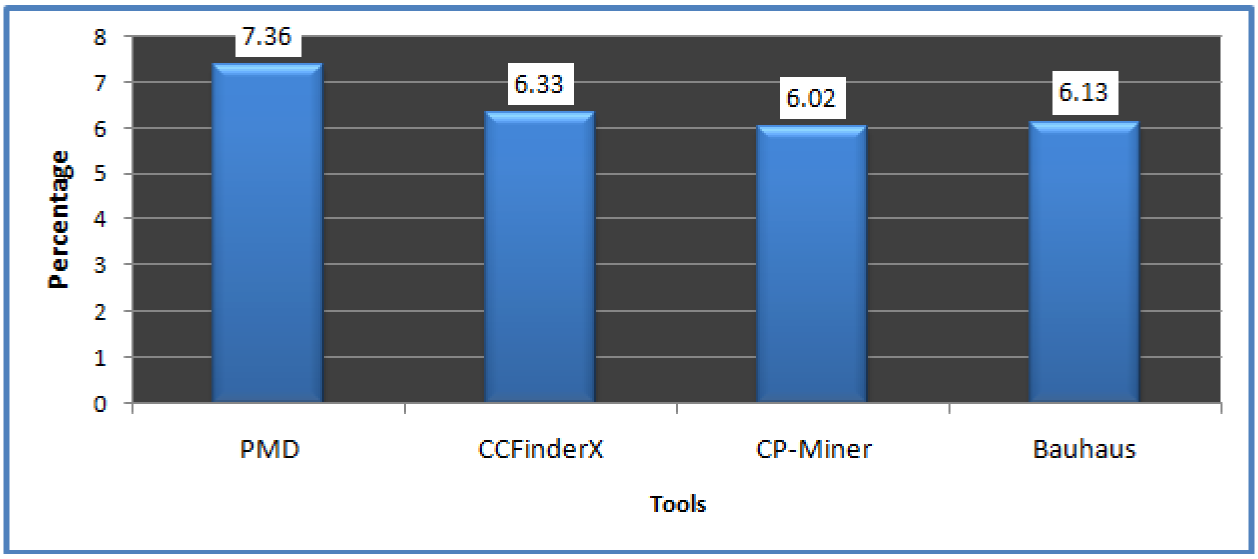

Figure 1: Percentage of Clones Identified

Type-4: Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

It is found that none of the clone detection tool is able to find the type-4 code clones. Table 3 shows the type of clone identified by each tool. The variation in the type of clones detected by each tool is based on the fact that each tool implements a different algorithm or technique. The simpler type clones i.e. type-1 and type-2 are easy to locate and thus are identified by all of the four tools; however type-3 clone is not identified by all tools.

PMD and CCFinderX are able to find the type-3 as well as type-1 and type-2 clones. CP-Miner and Bauhaus tools could only detect type-1 and type-2 clones. If type of clones detected is considered as the basis of comparison of tools then it can be concluded that PMD and CCFinderX are the better tools.

3.3 Precision and Recall of Tools

Precision and recall are the two important factors considered while comparing the clone detection tools. A brief description of each is given below:

1. **Precision:** The tool should be good enough so that it detect less number of false positives i.e., the tool should find duplicated code with higher precision. It is calculated as:

$$\text{Precision} = \frac{\text{Number of correct detected clones}}{\text{Number of clones detected}}$$

2. **Recall:** The tool should be capable of locating and finding most (or even all) of the clones of a system of interest. It is calculated as:

$$\text{Recall} = \frac{\text{Number of correct clones detected}}{\text{Number of possible existing clones}}$$

A tool with higher precision and recall values is considered as a better tool. Table 4 shows the precision and recall of each tool calculated as the result of this study. It can be seen from the table that precision and recall for PMD and Bauhaus tools are complementary i.e. if precision is high then recall is low and vice versa. Figure 2 shows the precision and recall values of each tool

Table 3: Type of Clone Identified

Tool	Type of Clones Identified			
	Type-1	Type-2	Type-3	Type-4
PMD	Yes	Yes	Yes	No
CCFinderX	Yes	Yes	Yes	No
CP-Miner	Yes	Yes	No	No
Bauhaus	Yes	Yes	No	No

Table 4: Precision and Recall of Tools.

Tool	Precision	Recall
PMD	0.46	0.59
CCFinderX	0.56	0.51
CP-Miner	0.41	0.48
Bauhaus	0.81	0.49

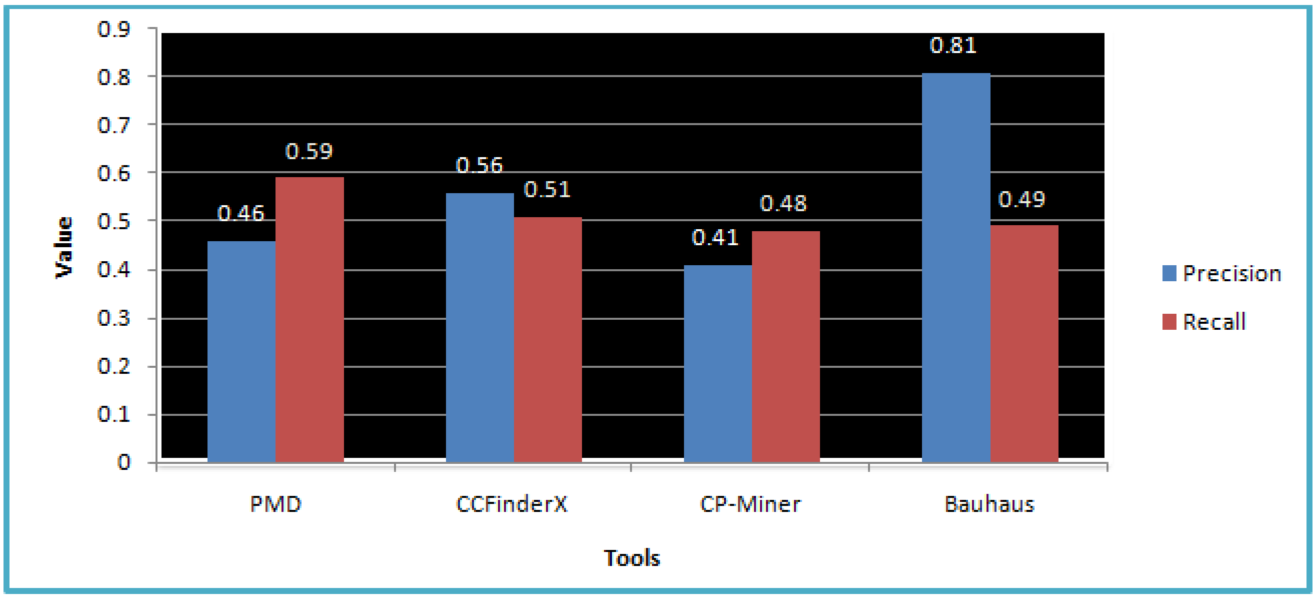

Figure 2: Precision and Recall of Tools

PMD has a highest recall value however it's precision is not that good which means that it finds maximum number of clones but with large number of false positives. Bauhaus has the highest value for precision of all the tools but its recall value is very small compared to its precision which means that most of the clones detected by this tool are true clones with less number of false positives but it fails in detecting most of the clones. There is not much difference in the precision and recall values of CCFinderX and CP-Miner. CP-Miner has the lowest precision and recall values among all the tools. CCFinderX has little difference in the values of precision and recall which means that this tool is detects most of the clones with less number false positives.

If precision and recall of each tool is considered for the comparison of tools then it is very difficult to decide which tool is best as all the tools behave complementary for precision and recall values. PMD has the highest value for precision but its recall is small, similarly Bauhaus has the highest value for recall and its precision is not good. The only tool that has smaller difference in precision and recall values is CCFinderX.

4. Conclusion

This paper presented a comprehensive study of four clone detection tools which are PMD (Programming Mistake Detector), CCFinderX, CP-Miner and Bauhaus. The specification of each tool is presented. The results were obtained by applying clone detection tools to a subject system known as EIRC (Eteria Internet Relay Chat) which is a chat program often used for clone detection studies. PMD and CCFinderX were able to detect type-1, type-2 and type-3 clones and no tool was able to find type-4 clones. The results show that precision and recall of tools behave complementary and each tool may have a different use. The results of the study suggest that each tool has its own advantages and disadvantages and no tool overcome the other.

References

[1] Shahid Ahmad Wani and Shilpa Dang, “Survey Based Analysis of Effect of Code Clones on Software Quality”, IJERT, ISSN: 2278-0181, Vol. 4(3), pp. 371-379, 2015.
[2] Shahid Ahmad Wani and Shilpa Dang, “A comparative study of clone detection tools”, ISSN: 2321-7728, IJARCSMS, Vol. 3(1), pp. 37-41, 2015.
[3] I. Baxter, A. Yahin, L. Moura, M. S. 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] T. Kamiya, S. Kusumoto, Katsuro Inoue, “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code”, Transactions on Software Engineering, Vol. 28(7): 654- 670, July 2002.
[5] Z. Li, S. Lu, S. Myagmar, Y. Zhou, “CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code”, In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04), pp. 289-302, San Francisco, CA, USA, December 2004.
[6] S. Bellon and V. von, “techniken zur erkennung duplizierten quellcodes”, Diploma Thesis, No. 1998, University of Stuttgart (Germany), Institute for Software Technology, September 2002.
[7] T. Sager, A. Bernstein, M. Pinzger, C. Keifer, “Detecting Similar Java Classes Using Tree Algorithms”, In Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR'06), pp. 65-71, Shanghai, China, May 2006.
[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “DECKARD: Scalable and Accurate Tree-based Detection of Code Clones”, In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), pp. 96-105, Minnesota, USA, May 2007.
[9] http://ccfinder.net
[10] http://www.pmd.sourcegefourge.net
[11] R. Komondoor and S. Horwitz, “Tool demonstration: Finding duplicated code using program dependences”, In Proceedings of the European Symposium on Programming (ESOP'01), Vol. LNCS 2028, pp. 383386, Genova, Italy, April 2001.
[12] J. Krinke, “Identifying Similar Code with Program Dependence Graphs”, In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309, Stuttgart, Germany, October 2001.