IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-12, Dec 2018 E-ISSN: 2347-2693

Metaphorical Analysis of Software Clone Detection Techniques based on Dimensions and Metrics

Sarveshwar Bharti[1*], Hardeep Singh[2]

[1, 2]Department of Computer Science, Guru Nanak Dev University, Amritsar, India

[*]*Corresponding Author: sarveshwar.dcsrsh@gndu.ac.in, Tel.: +91-9906129214*

Available online at: www.ijcseonline.org

Accepted: 26/Dec/2018, Published: 31/Dec/2018

***Abstract*—** In spite of having limited benefits, software clones mostly have negative impact on software quality, more specifically on software maintenance and thus diminishing software quality and raising the maintenance cost. Not all the clones are possible to remove, but, if possible clones need to be removed from the software system. To remove clones, we need to first detect this duplication in the code base. Literature lists various clone detection techniques that are used to detect duplication in software system. To have a better clone detection technique in future or to select from the available clone detection technique, these available techniques found in literature need to be analyzed. This paper attempts to comparatively analyze the clone detection techniques available in literature and thus will present a future scope as well as the recourse based on the analysis for selection of any particular technique.

***Keywords*—**Code Clone Detection, Clone Detection Techniques, Comparative Analysis

I. INTRODUCTION

Software Clones are basically defined on the notion of significant similarities between code fragments. In literature, the widely used definition of clones was given by Baxter et al. [1], stating "a clone is a program fragment that [is] identical to another fragment". Similarity between these code fragments can be of syntactic or semantic type. Bellon et al. [2], presented the classification of clones based on the similarity between code fragments. Based on the syntactic similarity, three clone types were defined viz. Type1, Type2 and Type3 and based on the semantic similarity, Type4 clones were defined. There are various intentional as well as unintentional reasons behind the induction of clones in the software system, as discussed in [3]. Literature mentions 9% to 17% [4] of the code of any software system may be a cloned code. There have been various studies that have proved the negative impact of clones on the software system and thus researchers contend that these duplicated code fragments must be polished off and if possible should be averted to be inducted into the software system. Process of code clone detection mainly includes the code transformation and then the match detection. After extraction of the code to be matched, it is transformed into an internal format. This format is used by the implemented algorithm to detect matches more efficiently. To detect clones there are various clone detection techniques available in literature that use different internal format to represent code and accordingly they can be classified into different types based on this internal format viz. text based, tree based, graph based, metrics based and hybrid techniques.

To stimulate a better clone detection technique, these clone detection approaches necessitates to be studied. This paper will endeavor to compare these clone detection approaches based on different parameters.

The rest of the paper is organized as follows. Section II portrays the objective of this paper. Section III discusses the literature related to the present study. Section IV lists various clone detection techniques found in literature. Section V presents various dimensions of clone detection techniques, VI discusses various evaluation metrics and then comparative analysis of the clone detection techniques is performed in Section VII. And, then finally conclusion and future work is presented in Section VIII, along with acknowledgments and references in the support of this paper.

II. MOTIVATION AND OBJECTIVE

As discussed above, it has been empirically evidenced that clones have a negative impingement on the software quality and mainly on the software maintenance, thus, these clones present in the software system needs to be removed from the software system. To remove clones, these must be detected first, and, to detect clones in the software system, literature lists number of clone detection approaches. Different types of code clone detection techniques found in literature are text based, tree based, graph based, metrics based, hybrid etc. that should be comparatively analyzed to help the software clone researchers to select a suitable technique for the system under consideration and also

Table 1. Related Literature

S. No.	Author	Year	Title	Reference
1.	Rysselberghe and Demeyer	2004	Evaluating Clone Detection Techniques from a Refactoring Perspective	[5]
2.	Roy and Cordy	2007	A Survey on Software Clone Detection Research	[3]
3.	Roy and Cordy	2008	Scenario-Based Comparison of Clone Detection Techniques	[6]
4.	Roy and Cordy	2009	Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach	[7]
5.	Ratten et al.	2013	Software Clone Detection: A Systematic Review	[8]
6.	Sheneamer and Kalita	2016	A Survey of Software Clone Detection Techniques	[9]

identify the future potential for a new technique. So, the main motive of this paper is to parametrically analyze and thus evaluate different clone detection techniques that are discussed in the code clone literature.

III. Related Literature

This section will attempt to summarize various studies that compare and evaluate various clone detection techniques. Various related studies found in literature as shown in table 1 are discussed below.

In the year 2007, Roy and Cordy [3] presented a detailed survey on software clone detection research. They compared and evaluated various clone detection techniques based on the various properties viz. comparison granularity, code representation, transformation, refactoring opportunities etc. To evaluate various tools they first did the high level evaluation of detection approaches. This high level comparison was done by comparing different approaches using different parameters viz. portability, scalability, precision, recall and robustness. In 2008, Roy and Cordy [6], presented the scenario based comparison of various clone detection techniques. Roy et al. [7] in the year 2009 qualitatively compared and then evaluated clone detection techniques found in literature. Ratten et al. [8] discussed the systematic literature review followed by them in identifying various tools and then the identified tools and implemented techniques were compared. Rysselberghe and Demeyer [5] presented the comparison of three representative detection techniques. Sheneamer and Kalita [9], in the year 2016 came up with a detailed survey of various clone detection techniques.

Complementing the above mentioned surveys, this paper compares and evaluates the clone detection techniques and identifies the future potential by integrating the empirical observations from the previous surveys. In contrast to the previous surveys, this study presents simple and easily adaptable observations with the emphasis on various strengths as well as weaknesses.

IV. Clone Detection Techniques

The corpus of the software clone research incorporates number of clone detection techniques by proficient researchers of research community. Researchers detected clones using various clone detection tools that implements different detection approaches. This section will discuss various clone detection techniques found in literature. Literature study revealed different types of approaches that were employed to detect clones as shown in figure 1 are discussed below:

A. Text Based Clone Detection Techniques

Various clone detection techniques detect clones by comparing the program text directly, considering it as a sequence of lines/strings. This technique involves very little transformation of the source code that becomes an input to the comparison algorithm. In this technique number of lines is taken as a clone size.

B. Token Based Clone Detection Techniques

This technique involves the transformation of source code involving lexing/parsing/transformation into the sequence of tokens. After scanning this sequence of tokens to detect duplicated fragments, the duplicate sub sequence pairs of the tokens are reported as clones. This technique is more robust than text based approach.

C. Tree Based Clone Detection Techniques

Using parser of the language under consideration, the source code in parsed into an Abstract Syntax Tree (AST) or parse tree. After the source code transformation, the sub tree searching algorithm is applied on the tree representation of

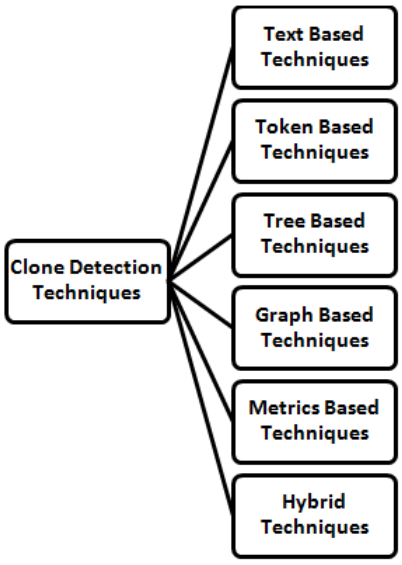

Figure 1. Clone detection techniques

Table 2. Implementation under each Detection Approach (Only one publication)

S. No.	Author	Approach	Tool	Year	Reference
1.	Brenda Baker	Text Based	Dup	1995	[10]
2.	Kamiya et al.	Token Based	CCFinder	2002	[11]
3.	Baxter et al.	Tree Based	CloneDr	1998	[1]
4.	Jens Krinke	Graph Based	Duplix	2001	[12]
5.	Mayrand et al.	Metrics Based	Mayrand et al.	1996	[13]
6.	Hummel et al.	Hybrid	ConQAT	2011	[14]

the source code and the matching sub trees are returned as clone pairs.

D. Graph Based Clone Detection Techniques

To go step further, semantic information of the source code is obtained using program dependency graphs (PDG). These PDGs contain the semantic information as the control flow and data flow information of the source code. Similar sub graphs are returned as clone pairs by applying sub graph matching algorithm.

E. Metrics Based Clone Detection Techniques

Instead of comparing the program code directly, metrics based approaches calculate various program metrics and then compare these metrics values to identify the similar code fragments. Most of the times, to calculate metrics, the source code is transformed into an intermediate representation e.g. AST/PDG representation. Metrics can be calculated at any granularity like at statement level, method level, class level etc.

F. Hybrid Clone Detection Techniques

When more than one clone detection technique is applied or more than one transformation is carried out for detecting clones, we call this type of technique a hybrid approach. For example, in software clone literature there are various clone detection techniques that uses AST as well as Suffix tree representation to detect similar fragments.

Table 2 presents the one implementation as example of the above mentioned techniques discussed in the clone literature. Dup is a text based tool implemented by Baker in the year 1995, CCFinder is a token based tool implemented by Kamiya et al. in the year 2002, tree based clone detection tool CloneDr was implemented by Baxter et al. in the year 1998, in the year 2001 Krinke came up with Duplix, a graph based tool and in the year 2011 ConQAT was developed by Hummel et al.

V. DIMENSIONS OF CLONE DETECTION TECHNIQUES

To efficiently analyze the clone detection techniques, C. K. Roy and J. R. Cordy [3], presented various dimensions (properties), based on which various detection approaches can be elucidated. Figure 2 shows the various dimensions used to compare different techniques that are discussed below:

Figure 2. Dimensions of clone detection techniques

A. Source Transformation/Normalization

Clone detection approaches first transform the source code into the suitable format and then the clone detection is applied. Source code may also be normalized by removing white spaces and comments.

B. Source Representation

After the transformation, detection algorithm works on the transformed code, represented in a desirable format.

C. Comparison Granularity

To detect clones, detection algorithms may compare code at only few lines of granularity or may compare at the level tree or graph node etc.

D. Comparison Algorithm

Different sub fields of the clone research, applies different algorithms, as per the requirement.

Table 3. Comparison of Various Clone Detection Approaches over 11 Dimensions

Dimension	Clone Detection Technique					
	Text Based	*Token Based*	*Tree Based*	*Graph Based*	*Metrics Based*	*Hybrid*
Source Transformation / Normalization	White space & comments removed	Transformed into tokens through lexical analysis	AST created by parsing the source code	PDG created through parsing the source code	Generates metric values	More than one transformation
Source Representation	Normalized code	Token sequence	(AST) Tree representation, vectors	(PDG) Graph representation	Metrics Values, vectors	More than one representation
Comparison Granularity	Lines	Tokens	AST Nodes	PDG Nodes	Metrics calculated from each block	Comparison at more than one granularity
Computational Complexity	Depends on algorithms	Linear	Quadratic	Quadratic	Linear	As per the approach
Comparison Algorithm	Suffix Tree, Array, data mining, IR, fingerprinting, DMP, Sequence matching	Suffix Tree, Array, data mining, IR, Sequence matching	Suffix Tree, Array, data mining, IR, hash value comparison, Sequence matching	Graph Matching	Euclidian Distance	As per the approach selected
Clone Similarity	Exact Match, Near miss, others	Exact Match, Renamed match, Near miss	Exact Match, Renamed Match, Near miss, others	Exact Match, Near miss	Exact Match, Near miss	Any two or more
Clone Granularity	Free, Fixed	Free, Fixed	Free, Fixed	Free	Fixed	Free/Fixed
Language Independency	Adapts easily	Lexer needed	Needs parser	Needs PDG Generator	Parser needed mostly	As per the approach selected
Output/Groups of Clones	Clone pair, Clone class	Clone pair, Clone class	Clone pair, Clone class	Clone pair, Clone class	Clone pair, Clone class	Clone pair, Clone class
Clone Refactoring	Exact match help in refactoring	Post processing needed	Syntactic clones help in refactoring	Supports refactoring	Requires manual inspection	As per the approach selected
Language Paradigm	Procedural, OOP	Procedural, OOP	Procedural, OOP	Procedural	Procedural, OOP	Procedural/OOP

Table 4. Comparative Metric Evaluation of Code Clone Detection Tools

Approach	Tool	Precision	Recall	F-measure	Reference
Text Based	Dup[1]	3.1% - 9.3%	56% - 81.5%	5.95% - 16.04%	[10]
Token Based	CCFinder[1]	0.8% - 6.6%	44.5% - 100%	1.58% - 12.33%	[11]
Tree Based	CloneDr[1]	6% - 40.3%	14.9% - 48.1%	8.68% - 32.55%	[1]
Graph Based	Duplix[1]	2.9% - 10.5%	17.3% - 45.8%	5.35% - 17.08%	[12]
Metrics Based	Mayrand *et al.*	N/A	N/A	N/A	[13]
Hybrid	ConQAT	N/A	N/A	N/A	[14]

1 Values taken from an experiment conducted by Murkami et al. as mentioned in [9]

Table 5. Summary of Comparative Metric Evaluation of Code Clone Detection Tools and Techniques

Detection Approach	Tool Used	Precision	Recall	Reference
Text Based	Dup[1]	High	Low	[10]
Token Based	CCFinder[1]	Low	High	[11]
Tree Based	CloneDr[1]	High	Low	[1]
Graph Based	Duplix[1]	High	Medium	[12]
Metrics Based	Mayrand et al.	Medium	Medium	[13]

1 Values taken from an experiment conducted by Murkami et al. as mentioned in [9]

E. Computational Complexity

To detect clones efficiently, overall complexity of the algorithm, including extraction, transformation and matching should be taken into consideration, thus depicting the overall complexity of the algorithm.

F. Clone Similarity

It represents the type of the similarity between the code fragments and thus the types of clones viz. exact, near miss etc.

G. Clone Granularity

Pre-defined syntactic boundary (fixed granularity) can be at the function level, block level etc. or there may not be any boundary to report clones.

H. Language Independency

This property will depict the language the clone detection tool supports.

I. Output/Groups of Clones

Clones can be detected as clone pair, clone class or both.

J. Clone Refactoring

This property defines the support of the algorithm towards the refactoring of the detected clones.

K. Language Paradigm

To which language paradigm the clone detection approach targets viz. procedural, object oriented, assembly etc.

VI. EVALUATION METRICS

To evaluate various clone detection tools and thus various techniques, various frequently used metrics are discussed below:

A. Precision

One of the most common metric used to evaluate the clone detection tools is the positive predictive value (PPV), also known as precision. This metric refers to the relevant clone instances detected by clone detection algorithm out of all the retrieved clones. In terms of true positive (TF), true negative (TN), false positive (FP) and false negative (FN), precision is represented as:

$$Precision = \frac{\text{TP}}{(\text{TP+FP})}$$

B. Recall

It is another important metric used to assess the quality of the clone detection results also known as sensitivity or true positive rate. It refers to the relevant code clone instances detected out of all the clones present in the code base. In terms of the TP, TN, FP and FN, recall is represented as:

$$Recall = \frac{\text{TP}}{(\text{TP+FN})}$$

C. F-measure

F-measure also known as traditional F-measure or balanced F-score depicts the harmonic mean of the precision and recall, and is represented as:

$$F\text{-}measure = 2\frac{Precision\ .\ Recall}{Precision + Recall}$$

VII. COMPARATIVE ANALYSIS OF THE SOFTWARE CLONE DETECTION TECHNIQUES

As discussed in the previous sections, clones should be detected and to detect clones there are different approaches. Out of these detection approaches, which one should be selected, is a matter of concern. Thus, these approaches need to be comparatively analysed. This section presents the comparative analysis of all these clone detection approaches. Various dimensions discussed in the previous section can be used to compare various clone detection techniques. Table 3

presents the comparison of various clone detection techniques with respect to the various dimensions mentioned in the previous section. For example, the comparison granularity used by the clone detection techniques are discussed including lines, tokens, AST nodes, PDG nodes, metrics calculated presented under text based, token based, tree based, graph based and metrics based respectively. In the same manner all the other dimensions are discussed in the table 3.

Table 4 presents the comparative metric evaluation of different tools utilizing various clone detection approaches. This table, based on previous survey, obtains precision, recall and F-measure for various mentioned tools.

Table 5 summarizes the comparative evaluation results with high, low or medium measure of clone detection quality.

VIII. CONCLUSION

Authors conclude that in spite of having advantages in many cases, software clones cannot be left into the software system. Because, Software Clones have an adverse impact on the software quality, these should be removed. This paper first discussed various clone detection techniques available in literature viz. text based, tree based, token based, graph based, metrics based and the hybrid approach, along with various dimensions of these clone detection techniques viz. clone similarity, comparison granularity, language paradigm etc. Then extensive comparative analysis is performed considering various dimensions and evaluation metrics to describe each clone detection technique. Thus this paper gives an overview of the detection approaches and would help the researchers to identify the particular technique of his/her interest, or to develop a new one.

ACKNOWLEDGMENT

Authors would like to acknowledge UGC for the Research Fellowship to the first author and also would like to thank the Department of Computer Science, Guru Nanak Dev University for providing the needed infrastructure.

REFERENCES

[1] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant' Anna, and Lorraine Bier, "Clone Detection Using Abstract Syntax Tree," in Proceedings of 14th International Conference on Software Maintenance(ICSM'98), Bethesda, Mayland, 1998, pp. 368 - 377.

[2] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo, "Comparision and Evaluation of Clone Detection Tools," IEEE Transaction on Software Engineering, vol. 33, no. 9, pp. 577 - 591, 2007.

[3] Chanchal K. Roy and James R. Cordy, "A Survey on Software Clone Detection Research," Queen's University, Kingston, Technical Report 2007-541, 2007.

[4] Minhaz F. Zibran, Ripon K. Saha, Muhammad Asaduzzaman, and Chanchal K. Roy, "Analysing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study," in Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, USA, 2011, pp. 295-304.

[5] Filip Van Rysselberghe and Serge Demeyer, "Evaluating Clone Detection Techniques from a Refactoring Perspective," in Proceedings of the 19th IEEE international conference on Automated Software Engineering (ASE'04), Linz, Austria, 2004, pp. 336-339.

[6] Chanchal K Roy and JamesR Cordy, "Scenario-Based Comparison of Clone Detection Techniques," in The 16th IEEE International Conference on Program Comprehension, 2008, pp. 153-162.

[7] Chanchal Kumar Roy, James Cordy, and Rainer Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Quantitative Approach," Science of Computer Programming, vol. 74, no. 7, pp. 470 - 495, March 2009.

[8] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh, "Software Clone Detection: A Systematic Review," Information and Software Technology, vol. 55, no. 7, pp. 1165-1199, July 2013.

[9] Abdullah Sheheamer and Jugal Kalita, "A Survey of Software Clone Detection Techniques," International Journal of Computer Applications, vol. 137, no. 10, pp. 1 - 21, March 2016.

[10] Brenda Baker, "On Finding Duplication and Near Duplication in Large Software Systems," in Proceedings of the 2nd Working Conference on Reverse Engineering (WCRE'95), 1995, pp. 86-95.

[11] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System For Large Scale Source Code," IEEE Transactions on Software Engineering, vol. 28, no. 7, pp. 654-670, July 2002.

[12] Jens Krinke, "Identifying Similar Code with Program Dependence Graphs," in Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01), Stuttgart, 2001, pp. 301-309.

[13] Jean Mayrand, Clande Leblane, and Ettore Merlo, "Experiment on the Automatic Detection of Function Clones in a Software Systems Using Metrics," in Proceedings of International Conference on Software Maintenance (IWSM'96), Monterey, 1996, pp. 244 -253.

[14] Benjamin Hummel, Elmar Juergens, Lars Heinemann, and Michael Conradt, "Index-Based Code Clone Detection: Incremental, Distributed, Scalable," in IEEE International Conference on Software Maintenance, Timisoara, Romania, 2010.

Authors Profile

Mr. Sarveshwar Bharti is presently working at the Department of Computer Science, Guru Nanak Dev University, Amritsar, India, as a Ph.D. Research Fellow. He has received his Master of Computer Applications (MCA) degree from University of Jammu, Jammu, India. He is a Software Engineering Researcher with research interests including Software Clones, Integrated Clone Management, and Clone Management Plug-in.

Dr. Hardeep Singh is a Professor and Head at the Department of Computer Science, Guru Nanak Dev University, Amritsar, India. His research interests lie within Software Engineering and Information Systems. He has been awarded with various prestigious awards including Dewang Mehta Award for best Professor in Computer Engineering, ISTE Award for Best Teacher in Computer Science and Rotract International Award for best Teacher.

