International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-3, February 2019

Identification of File and Directory Level Near-Miss Clones for Higher Level Cloning

Sonam Gupta, Vishwachi

Abstract: ***The presence of code cloning, which speaks to duplicate segments of code, has been archived to happen much of the time in programming frameworks. The principle reason for cloning is reusing the part of the code that plays out some usefulness by duplicating and rewriting it to another area in the source code. The concentration here is to detect duplication in software program, which is a noteworthy reason for poor structure in programs. The key, novel part of our duplication-detection approach is detection of near-miss clones at higher level of granularity i.e, the directory and file level. Our work is a progress over past work around there as earlier the granularity of clone detection was method level. The technique adopted incorporates a novel hybrid approach using Abstract Syntax Tree and metrics to achieve precision. The results indicate the ability to detect Type-1, 2, 3 clones at directory and folder level.***

Index Terms: ***Levenshtein distance, re-ordered clones, syntactic clones.***

I. INTRODUCTION

Code cloning is the practice of reusing the source code by the method of copy and paste with or without minor adaptation. It is a common practice in software development as a result of which software systems often contain sections which are quite similar to each other, known *software clones*. Identification of clones is of great importance for the software practitioners as well as for researchers. The reason is that they induce problems at the time of software maintenance. It has been evaluated that out of several phases of software development life cycle like requirement gathering, design, implementation, deployment the majority of the cost involved is in the maintenance phase. It is so due to the revamping needs and requirements of the clients because of changing technical scenario. Cloning in the software makes it difficult to modify. Code clone detection is now a crucial and valuable part of software analysis as many engineering tasks such as program understanding, code quality analysis, aspect mining,
copyright infringement identification, plagiarism detection, code compaction, virus and bug detection may require the extraction of syntactic and semantically similar fragments [5].It is has been revealed in a study [13] that 5-10% of source code of large computer programs is duplicated.

Manuscript published on 28 February 2019.
* Correspondence Author (s)
Sonam Gupta, Associate Professor, Department of Computer Science and Engineering, Ajay Kumar Garg Engineering College, Ghaziabad, India
Vishwachi, Assistant Professor, Department of Information Technology, ABESIT Engineering College, Ghaziabad, India

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC-BY-NC-ND license http://creativecommons.org/licenses/by-nc-nd/4.0/

Clone detection must be adopted as a pro-active measure in the development of software for avoiding the difficulties faced by clones later on. The existence of code clones may be broadly classified into two categories: Accidental and Intentional clones. Intentional clones are the ones which are made because of developer's purposeful copy and pasting exercises. The developer duplicates existing code and pastes it to somewhere else with or without further change which stays like the first, in this manner shaping the clone-pair. Obviously, the code clones can show up in the framework without the software engineer's aim. For instance, because of designer's mental model, habitually utilized phrases are imitated from memory rather than ponder copy and paste[14]. Such comparative code pieces that are not presented by think copy and paste operations are called accidental clones.

II. CLONE GRANUALITY

In writing, proximate bit of code at various granularity levels have been utilized. The most usually utilized granularities are at file level, method level, class level. Code clones granularity can be of following kinds:

A. *Directory Clone*

At the point when two directories are found to have practically identical source code more imperative than or comparable to a specific threshold, they are called directory clones.

B. *File Clone*

At the point when two distinct records are found to have a coveted edge of comparable source code, they are called file clones.

C. *Function Clones*

At least two functions are considered clones when the assemblages of functions comprise of comparable code. The likeness can be syntactic and in addition semantic.

D. *Block Clone*

At the point when two blocks of code are sufficiently comparable, they are called block clones.

E. *Class Clone*

In a protest situated source code, the classes can be considered as clones on the off chance that they have indistinguishable or close indistinguishable code.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

International Journal of Advanced Technology and Engineering
IJEAT
www.ijeat.org
Exploring Innovation

F. *Statements Clone*

At the point when two gatherings of articulations, situated at discretionary area of file are observed to be sufficiently comparative they are said to be statement clones.

III. NEGATIVE IMPACTS OF CLONING

Code cloning produces extensive number of negative effects which must be taken into the light. The significant disadvantages of code cloning are as per the following:

•Duplicating a code piece which encapsulates any defect obscure to the designer may bring about proliferation of that defect to every one of the duplicates of the program.

•Cloning generates issue at the time of upkeep as modification in one code piece may require alterations in all clones of that section else it might prompt irregularities.

•After formation of clone, it might be hard to hound which code section is the parent code and which one is the child code that is, dupliacte code.

•Resource necessity additionally increments because of code cloning as when the framework measure increment, execution time and memory prerequisite likewise gets expanded.

IV. LITERATURE REVIEW

The work done by different researchers in the techniques developed are discussed in this section. The work done in different fields are as follows:

TEXTUAL COMPARISON:

A tool named Dude was developed by Wettel et al. [16]. Dude is a text based clone detection tool. It has the capability of detecting duplicate clones comprising of various clones of smaller size. Its major drawback is that it do not have the potential of being applied to systems with

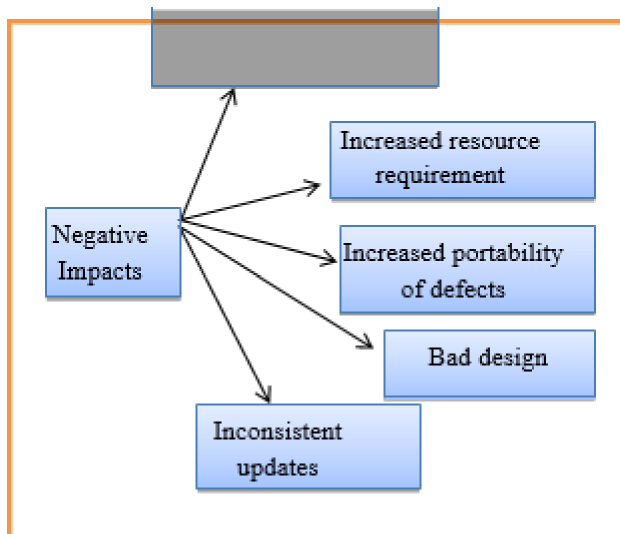

Figure 1. Negative impacts of clones

large lines of codes. The SDD (Similar Data Detection) tool developed by S. Lee et al. [17] is quite useful for detecting code clones in systems with large number of lines of code but its accuracy is not high. NICAD, developed by C.K.Roy et al. [18] is capable of detecting near-miss clones very efficiently but it is not purely text based approach. Duccase et al. [19] introduced a method which reads files with source codes, makes sequences of lines, applies the preprocessing on lines and detects match by a string-based Dynamic Pattern Matching (DPM) algorithm. The output produced involves the line numbers of the clone pairs [20].

TOKEN BASED APPROACH

CCFinder [21] developed by T.Kamia et al. uses algorithm based on suffix tree matching for finding identical sequences and is a token based tool for detecting code clones. CPMiner [22] is another token based tool which incorporates frequent item-set mining for detecting software bugs introduced due to copy and paste. It detects clones at structural level which are abstractions of high level . A tool named “clones” was developed by Koschke et al. [23] uses an abstract syntax tree (AST) and parser. AST is serialized afterwards and is fed as an input to the suffix tree.

METRICS BASED APPROACH

Mayrand et al. [24] developed CLAN which compares metrics which are obtained from an Abstract Syntax Tree of the program code . Computation of metrics are done from expressions, flow, names and control information of functions . Patenaude et al. [25] detected metrics from the program code modified in different themes namely , coupling , classes , functions , hierarchical structure and clones . Kontogiannis et al . [26] developed an approach in which dynamic programming is applied on lines of code using minimum edit distance which detects the match between them . Perumal et al. [27] used fingerprinting and metrics technique for clone detecting but the technique is quite costly.

PDG APPROACH

R. Komondoor et al. [28] developed a tool which incorporated program slicing for finding out the isomorphic sub-graphs which is based on PDG approach . The main feature of this approach is that it helps in finding out the non-contiguous clones. Another tool named Scorpio was developed by Higo and Kusumoto [29] in the year 2011 which applied two-way slicing technique for detecting clones. The aim for developing it was to address the problem faced by the existing of particularly slow detection of the clones especially contiguous.

AST BASED

CloneDR developed by I.D. Baxter et al. [1] used dynamic programming and hashing and is able to detect exact and reordered clones . A compiler generator is utilized to create a clarified parse tree (AST) and looks at its sub-trees by portrayal measurements in light of a hash work through tree coordinating . Source codes of the comparable sub- trees are then returned as clones .

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

Yang [30] proposes accomplice procedure to look out the language structure assortments between two methods; this downside, however not predictable as clone acknowledgment, is said thereto. Baxter et al. [1] propose accomplice AST based strategy to finding clones in supply code. They recognize correct clones by finding vague AST sub-trees and mistaken clones by finding sub-trees that locale unit indistinct once factor names and demanding qualities region unit overlooked.

V. PROPERTIES OF CLONE DETECTION TECHNIQUES

In literature several clone detection technique have been discovered. Code clone detection techniques are composed of several properties. An overview of these properties along with their meaning is given in Table below [31].

Property	Detail
Source Transformation	The transformation or filtering applied by each technique before performing the actual comparison.
Comparison Granularity	Each of the techniques works on different levels of code granularity. Some of the techniques work on one line of code while some works on function as code granularity.
Comparison Algorithm	This property means that which comparison algorithm is used by each technique to identify code clones.
Clone Similarity	This property means that which type of clones can be detected by each technique. For example, some of the techniques can detect only Textual clone while others can detect behavioral too.
Clone Refactoring	This property indicates if the clone detection technique supports clone refactoring or not.
Output	It means whether the clone detection technique returns clone pairs or clone classes as output.
Language Paradigm	This property means that for which language a This property means that for which language a

VI. PROPOSED APPROACH

Practically, manual clone recognition is infeasible for extensive programming frameworks along these automated support is vital. As far back as the work in this field has begun, numerous code clone detection methods have been developed. In this paper, we exhibit a novel method in light of AST i.e. Abstract Syntax Trees and metrics. The unmistakable elements of the approach proposed in this paper for code clone detection are the distinguishing proof of directory and folder or file level granularity.

To begin with, the user needs files and directories in which the clones are to be discovered as it is the novelty of our approach. We have developed a tool named MultDup which is detecting the clones in :

Number of input statements: Here, we calculate the total number of input statements in the individual files and directories chosen for clone detection.

Number of output statements: Here, we calculate the total number of output statements in the individual files and directories chosen for clone detection.

Total conditional statements: Computing quantity of conditional statements is important as it helps in determining the overall semantics. We will calculate the total number of conditional statements in an individual file and will later on compare them.

Number of iteration statements: Here, we will calculate the number of iteration statements as it is useful in identifying the pattern of evaluation.

Number of functions called: A function call is an expression that passes control and contentions to a function. By deciding the number of functions brought in an individual file and contrasting it to the number of functions called in another file, will help us in deciding the level of cloning in both the files and directory on a higher level. *Complexity*: Complexity of a program is defined as the number of instructions which are executed by a program during its running time. Here, we have calculated the cyclometic complexity of the system by using formula:

CC= (1+CASEs+LOOPs+IFs)

Where,

CC =Cyclometic complexity

Ifs represents total conditional statements in a file

LOOPs are the total loops in a file(for,while,do- while)

CASEs represents the total number of switch statements in a file.

The fundamental purpose behind figuring the above metrics independently for every one of the documents is to present improvement in the framework. The object is accomplished by comparing the metrics values calculated. Files and directories which will further be involved in the process of clone detection are those whose metrics matches the minimum threshold value. If the metrics values do not match, then they will be discarded to be considered as clones. It helps in bringing optimization in the procedure.

The files and directories whose above computed metrics comes out to be similar will further be involved in the process of AST formation. As soon as we get the Abstract Syntax Tree representation of the files it will be converted into template and levenshtein distance between the files will be calculated.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.
International Journal of Engineering and Advanced Technology
IJEAT
www.ijeat.org
Exploring Innovation

A levenshtein distance is the number of insertions, deletions and substitutions to be done in a file to make it similar to another file. If the levenshtein distance calculated between the files is less than or equal to the set threshold, cloning is confirmed.

VII. ALGORITHM

Input: number of folders selected by user

Step 1: Inputting

Step 2: Preprocessing: removing comments and whitespaces
Step3: Metrics Computation: compute the lines of code, complexity, number of input and output statement etc. of each file

Step 4: Compare metrics of files // detect potential candidates for further processing

Step 5: If metrics matches then goto step 5 else stop Step 6: Perform AST formation of files

Step7: do template conversion

Step 8: Compute the levenshtein distance between files

Step 9: If levenshtein distance <= threshold, goto step 10 else cloning does not exist

Step 10: cloning exist Step 11: End

VIII. RESULTS AND DISCUSSION

The proposed approach is implemented and a tool is created in C#. The input to the tool is various folders of C, C# and Java. The metrics similarity shows that the folders can be clones but the final decision is based on the levenshtein distance. We have successfully found output and the near-miss clones are detected by the tool along with type-1 and type-2 clones. The clones are found between the files and the directories.

The working of the tool along with the results is shown in the form of figures below:

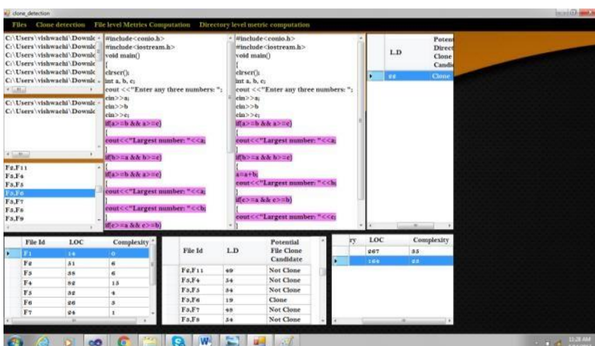

Figure 1. Result of file level cloning

First the information regarding each file will be shown in tabular format about the line of code and the complexity of each file. There can be total n number of files and the mapping is done between every two files in order to detect the file-level clones. The example of the result is shown in tabular format below. There is another section which tells the potential file clone candidates along with the levenshtein distance between each file.

Table 1. Metrics values of different files

File Id	LOC	Complexity
F1	14	0
F2	53	6
F3	82	12
F4	90	15
F5	68	9

Table 2. File level Potential Clone Candidates

File Id	Levenshtein Distance	Potential File Clone Candidate
F3,F10	19	Clone
F4,F3	8	Clone
F2,F1	21	Not Clone
F5,F7	30	Not Clone
F8,F2	23	Not Clone
F7,F3	11	Clone

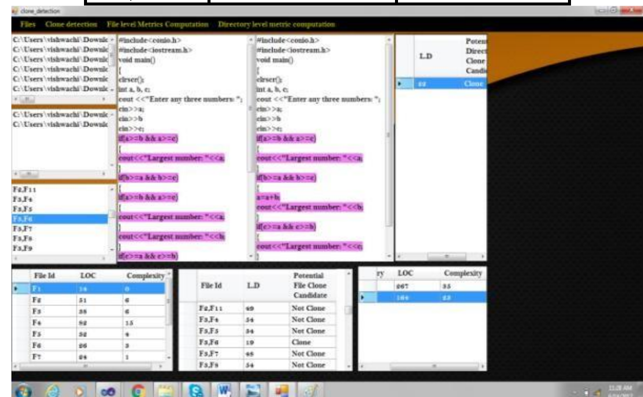

Figure 2. Result of directory- level cloning

Figure 2 is showing the result of directory level code clone detection. In the result the directories are labeled as D1, D2 and so-on. The lines of code and complexity of each directory is shown. The example is shown below:

Table 3. Metrics values of Directories

Directory Id	LOC	Complexity
D1	267	35
D2	164	23

Based on the levenshtein distance between the directories final result is displayed as potential clone candidate.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

Table 4. Directory level potential clone candidates

Did	L.D	Potential clone candidate
D1,D2	22	Clone

PRECISION AND RECALL

Tool	Precision%	Recall%
MultDup	92	91
Dude	91	88

Where,

Precision=number of correctly detected clones bytool

Total number of clones detected by the tool Recall= number of correctly detected clones by tool

Total number of actual clones present

The comparison of the tool designed is done with the previous tool named Dude[]. Dude detects the clones in code at method level. The tool developed here, i.e Multdup is detecting the clones at directory and file level. The comparison clearly shows that the tool developed has better precision and recall.

IX. CONCLUSION

Here, we have proposed an approach for detecting file and directory-level re-ordered by combining metrics with abstract syntax trees technique. Since the formation of Abstract Syntax Tree is done on the candidates shortlisted, a higher amount of recall could be obtained. Prior methodologies were not equipped for recognizing the clones between the files and directories. Rather, they could distinguish the clones just inside them. Here, we are able to find the clones between the files and directories which is quite beneficial from industrial as well as academic perspective.

REFERENCES

1. Baxter, Ira D., et al. "Clone detection using abstract syntax trees." Software Maintenance, 1998. Proceedings., International Conference on. IEEE, 1998.
2. Bellon, Stefan, et al. "Comparison and evaluation of clone detection tools." IEEE Transactions on software engineering 33.9 (2007).
3. Chatterji, Debarshi, Jeffrey C. Carver, and Nicholas A. Kraft. "Code clones and developer behavior: results of two surveys of the clone research community." Empirical Software Engineering 21.4 (2016): 1476-1508.
4. Choudhary V and Gupta S (2017). A novel approach in detecting code clones in Java using DFS. International Journal of Advanced and Applied Sciences, 4(5): 26-29
5. Cordy, James R., and Chanchal K. Roy. "The NiCad clone detector." Program Comprehension (ICPC), 2011 IEEE 19th International Conference on.IEEE, 2011.
6. Cuomo, Antonio, Antonella Santone, and Umberto Villano. "CD-Form: A clone detector based on formal methods." Science of Computer Programming 95 (2014): 390-405.
7. Gupta, Sonam, and P. C. Gupta. "A Novel Approach to Detect Duplicate Code Blocks to Reduce Maintenance Effort." International Journal of Advanced Computer Science & Applications 1.7 (2016): 311-314.
8. Kodhai, Egambaram, and Selvadurai Kanmani. "Method-level code clone detection through LWH (Light Weight Hybrid) approach." Journal of Software Engineering Research and Development 2.1 (2014): 12.
9. Kontogiannis, Kostas. "Evaluation experiments on the detection of programming patterns using software metrics." Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on. IEEE, 1997.
10. Mondal, Manishankar, et al. "An empirical study of the impacts of clones in software maintenance." Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. IEEE, 2011.
11. Rieger, Matthias, Stéphane Ducasse, and Michele Lanza. "Insights into system-wide code duplication." Reverse Engineering, 2004. Proceedings. 11th Working Conference on. IEEE, 2004.
12. Roy, Chanchal K. "Detection and analysis of near-miss software clones." Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. IEEE, 2009.
13. Roy, Chanchal Kumar, and James R. Cordy. "A survey on software clone detection research." Queen's School of Computing TR 541.115 (2007): 64-68.
14. Selim, Gehan MK, et al. "Studying the impact of clones on software defects." Reverse Engineering (WCRE), 2010 17th Working Conference on. IEEE, 2010.
15. Vishwachi and Sonam Gupta. Literature Survey of Software Clones. International Journal of Computer Applications 153(4):1-7, November 2016.
16. Wettel, Richard, and RaduMarinescu. "Archeology of code duplication: Recovering duplication chains from small duplication fragments." Seventh International Symposiumon Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05). IEEE,2005.
17. Li, Z. O., & Sun, J. (2010, April). A metric space based software clone detection approach. Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on (pp. 393-397). IEEE.
18. Roy, Chanchal K., and James R. Cordy. "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization." *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*. IEEE, 2008.
19. Ducasse, Stéphane, Matthias Rieger, and Serge Demeyer. "A language independent approach for detecting duplicated code." *Software Maintenance, 1999.(ICSM'99)*
20. *Proceedings. IEEE International Conference on*. IEEE, 1999.
21. S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD,,03), pp. 7685, San Diego, California, June 2003.
22. Kanagalakshmi, K., and R. Suguna. "Software Refactoring Technique for Code Clone Detection of Static and Dynamic Website."
23. Lozano, Angela, Michel Wermelinger, and Bashar Nuseibeh. "Evaluating the harmfulness of cloning: A change based experiment." *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007.
24. Koschke, Rainer, Raimar Falke, and Pierre Frenzel. "Clone detection using abstract syntax suffix trees." *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*. IEEE, 2006.
25. Neil Davey, Paul Barson, Simon Field, Ray J Frank. The Development of a Software Clone Detector. International Journal of Applied Software Technology, Vol. 1(3/4):219- 236, 1995 Raghavan Komondoor and Susan Horwitz. "Using slicing to identify duplication in source code." International Static Analysis Symposium.Springer Berlin Heidelberg 2001.
26. Koschke, Rainer, Raimar Falke, and Pierre Frenzel. "Clone detection using abstract syntax suffix trees." *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*. IEEE, 2006.
27. Pate, Jeremy R., Robert Tairas, and Nicholas A. Kraft. "Clone evolution: a systematic review." *Journal of software: Evolution and Process* 25.3 (2013): 261-283.
28. Rahman, Foyzur, Christian Bird, and Premkumar Devanbu. "Clones: What is that smell?." *Empirical Software Engineering* 17.4-5 (2012): 503-530.
29. Higo, Yoshiki, and Shinji Kusumoto. "Code clone detection on specialized PDGs with heuristics." Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. IEEE, 2011.
30. Yoshida, Norihiro, Takeshi Hattori, and Katsuro Inoue. "Finding similar defects using synonymous identifier retrieval." *Proceedings of the 4th International Workshop on Software Clones*. ACM, 2010.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

31. S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD„03), pp. 7685, San Diego, California, June 2003.

AUTHORS PROFILE

Dr. Sonam Gupta has received her BE degree from Mody Institute of Science & Technology, Lakshmangarh, Rajasthan (formely affiliated to Rajasthan University and now a deemed university), Mtech degree from SRM University, Chennai and PhD from Suresh Gyan Vihar University, Jaipur. Currently she is working as Associate Professor(CSE) in Ajay Kumar Garg Engineering College, Ghaziabad. Her research interests include software maintenance and evolution, machine learning and artificial intelligence.

Ms. Vishwachi is currently working as an Assistant Professor in ABESIT, Ghaziabad which is affiliated to APJ Abdul Kalam Technical University. She is B.Tech and M.Tech. Her research domain includes Software Engineering, Cryptography and network Security.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.