Various Detection Techniques and Types of Code Clone Maintenance of Software

Jasmandeep Kaur[1], Rakesh Kumar[2], Sukhjot Kaur[3]
[1]*Research (Scholar), Department of CSE, SECG, Gharuan*
[2]*Principal and Associate Professor, SECG, Gharuan*
[3]*Assistant Professor, Department of CSE, SECG, Gharuan*

Abstract—Large-scale software systems are expensive to build and, are even more expensive to maintain. In PC software, we could possibly have dissimilar sorts of repetition. We ought to note that not every type of redundancy is unsafe. There are diverse types of redundancy in software. Software embodies both programs and information. At particular times redundant is used correspondingly in the sense of unessential in the software engineering works. Sometimes, developers take easier way of implementation by copying some fragments of the existing programs & use that code in their effort. This kind of work is called code cloning. Redundant code is as well recurrently misleadingly permitted as cloned code despite the fact that it point towards that one bit of code which is possibly imitative from the additional one in the original meaning of this particular word. Even though cloning stimuli to redundant code, not each specific redundant code is a specific clone.

Keyword: *Code cloning, Textual Similarity, Functional Similarity, Textual comparison, Token based comparison, Abstract Syntax Tree Based Comparison.*

I. INTRODUCTION

A code clone is a code helping in source files that is equal or similar to another [1]. Duplication of code occurs frequently during the development of large software systems. Code cloning is a procedure of software reuse and exists in almost every software project. This ad-hoc form of reuse consists in copying, & eventually modifying, a block of existing code that apply a piece of required functionality. The main issue with code clone is coupled only with their similar code that is indirectly rather than directly which creates it problematic to identify them. The code quality declines and modification becomes more expensive and error-prone.

Copied blocks are called clones and the act ofcopying, counting slight modifications, is said cloning. Cloning mainly occurs because programmers find that it is cheaper and quicker to use the copy and paste feature then writing the code from scratch. Sometimes programmers intent on applying new functionality find particular working code that completes a computation nearly matching to the one desired copy it entirely & then modify in place[2]. While this is considered a serious problem in industrial software [3]. The quality of code analysis, virus recognize, facet mining and error exposure are the other tasks which need the knowledge of syntactically verified code part to facilitate code detection importance for software detection tool analysis.

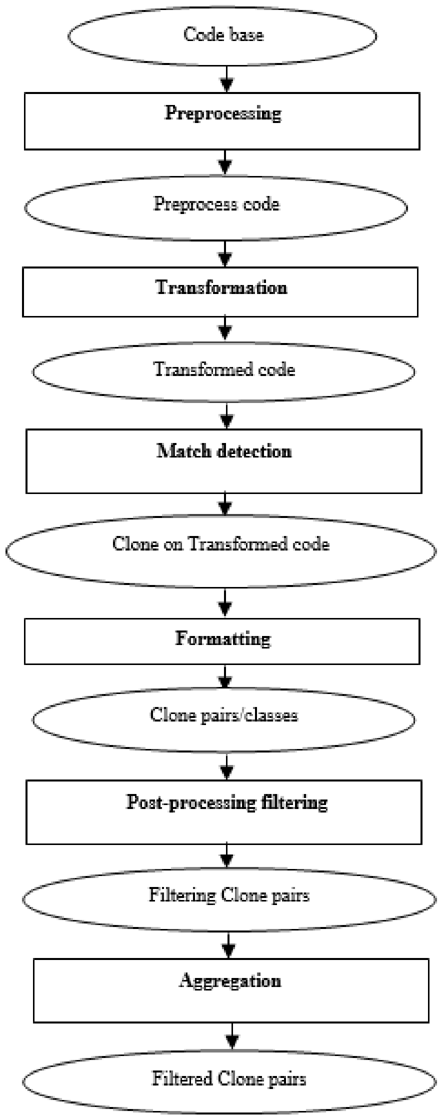

Fig.1: Generic Clone Detection Process [13]

A. Pre-processing

At the beginning of any clone detection approach, the source code is partitioned and the domain of the comparison is determined. There are three main objectives in this phase:

Remove uninteresting parts: All the source code uninteresting to the comparison phase is filtered out in this phase.

Determine source units: After removing the uninteresting code, the remaining source code is partitioned into a set of disjoint fragments called source units. These units are the largest source fragments that may be involved in direct clone relations with each other.

Determine comparison units / granularity: Source units may need to be further partitioned into smaller units depending on the comparison technique used by the tool. For example, source units may be subdivided into lines or even tokens for comparison.

B. Transformation

Once the units of comparison are determined, if the comparison technique is other than textual, the source code of the comparison units is transformed to an appropriate intermediate representation for comparison. This transformation of the source code into an intermediate representation is often called extraction in the reverse engineering community.

Extraction: Extraction transforms source code to the form suitable as input to the actual comparison algorithm. Depending on the tool, it typically involves one or more of the following steps.

Tokenization: In case of token-based approaches, each line of the source is divided into tokens according to the lexical rules of the programming language of interest. The tokens of lines or files then form the token sequences to be compared.

Parsing: In case of syntactic approaches, the entire source code base is parsed to build a parse tree or (possibly annotated) abstract syntax tree (AST).

Control and Data Flow Analysis: Semantics-aware approaches generate program dependence graphs (PDGs) from the source code. The nodes of a PDG represent the statements and conditions of a program, while edges represent control and data dependencies.

C. Normalization

Normalization is an optional step intended to eliminate superficial differences such as differences in whitespace, commenting, formatting or identifier names.

Removal of whitespace: Almost all approaches disregard whitespace, although line-based approaches retain line breaks. Some metrics-based approaches however use formatting and layout as part of their comparison.

Removal of comments: Most approaches remove and ignore comments in the actual comparison.

Normalizing identifiers: Most approaches apply identifier normalization before comparison in order to identify parametric Type-2 clones.

Pretty-printing of source code: Pretty printing is a simple way of reorganizing the source code to a standard form that removes differences in layout and spacing.

Structural transformations: Other transformations may be applied that actually change the structure of the code, so that minor variations of the same syntactic form may be treated as similar.

D. Match Detection

The transformed code is then fed into a comparison algorithm where transformed comparison units are compared to each other to find matches. Often adjacent similar comparison units are joined to form larger units.

E. Formatting

In this phase, the clone pair list for the transformed code obtained by the comparison algorithm is converted to a corresponding clone pair list for the original code base. Source coordinates of each clone pair obtained in the comparison phase are mapped to their positions in the original source files.

F. Post-processing / Filtering

In this phase, clones are ranked or filtered using manual analysis or automated heuristics.

Manual Analysis: After extracting the original source code, clones are subjected to a manual analysis where false positive clones or spurious clones [72] are filtered out by a human expert.

Automated Heuristics: Often heuristics can be defined based on length, diversity, frequency, or other characteristics of clones in order to rank or filter out clone candidates automatically.

G. Aggregation

While some tools directly identify clone classes, most return only clone pairs as the result. In order to reduce the amount of data, perform subsequent analyses or gather overview statistics, clones may be aggregated into clone classes.

II. RELATED WORK

Iman Keivanloo et.al (2011) [4] presented real-time code clone search was an emerging family of clone detection research that goals at finding clone sets matching an input code fragment in fractions of a second. For these techniques to meet actual real world requirements, they were to be scalable and provide a short response time. Our research presented a hybrid clone search approach using source code pattern indexing, information retrieval clustering, & Semantic Web reasoning to respectively achieve less response time, handle false positives, and support automated grouping/querying. **Yang Yuan et.al (2011) [5]** introduced CMCD; a Count Matrix based method to detect clones in program code. The key model behind CMCD was Count Matrix, which was created while counting the occurrence frequencies of every variable in conditions specified by pre-determined counting situation. Because the characteristics of the count matrix do not change due to variable name replacements or even switching of statements, CMCD works well on various hard-to-detect code clones, such as exchange statements or deleting a few lines, which are difficult for other state-of-the-art detection techniques. **Debarshi Chatterji et.al (2011) [6]** described a study that investigates the usefulness of code clone information for performing a bug localization task. In this study 43 graduate students were practical while identifying defects in both cloned & non-cloned portions of code. The goal of the study was to understand how those developers used clone information to perform this task. **Yoshiki Higo et.al. (2011) [7]** proposed a PDG based incremental code clone detection technique for improving practicality of PDG-based detection. A prototype tool developed, and it applied to open source software. And confirmed that detection time was extremely shortened and its detection result almost the same as one of an existing PDG-based detection technique.

III. CLONE TYPES

Code clone could be of any sort that all rely on upon the developer's method & ability of utilizing the code which varies from replicating as it is to matching the code however with some change which would be done at diverse level in the method. In software scheme code pieces predominantly demonstrates two sorts of similarities. One clone type of similarity considers textual similarity), & other second considers the semantic level that the clone code essential to have the identical behaviors, means the functional similarity.

A. *Textual Similarity:* Two code fragments can be similar based on the similarity of their program text we differentiate the subsequent sorts of clones. The following types of clones are discussed in order to find textual similarity [2].

1) Type I: In Type I clone, a copied code fragment is the same as the original. However, there might be particular variations in whitespace (blanks, new line(s), tabs etc.), remarks and/or designs. Type I is widely known as exact clones
2) Type II: A Type II clone is a code fragment that is the same as the original except for some possible differences about the corresponding designations of user-defined identifiers (name of variables. constants, class. methods & so on) layout, identifiers, remarks, literals, & sorts. The specific reserved words & the sentence structures are essentially the same as the original one.
3) Type III: Type DI is copy with further modifications. E.g. a new statement can be added, or some statements can be detached along with various dissimilarities in layout, identifiers, remarks, literals, & sorts. The structure of code fragment may be changed & they may even look or behave slight differently. This kind of clone is hard to be discovered, for the reason that the wholly framework understanding is needed.

B. *Functional Similarity:* Two code fragments can be similar based on the similarity of their functionalities without being textually similar. If the functionalities of the two code fragments are identical or similar i.e., they have comparable pre as well as post circumstances referred as Type IV clones or semantic clones [8].

4) Type IV: Type IV clones are the consequences of semantic similarity between two or extra code fragments which could accomplish the same computation however actualized through diverse syntactic variations. In this category of specific clones, the cloned part is not necessarily copied from the first one. Two code fragments may possibly be established by two different programmers too.

IV. DETECTION OF CLONE AND USES

Clone detection is useful in finding malicious software [8]. By comparing, one malicious software with another, it is possible to find the matched parts of one software system with another. Some applications of clone detection are as follows:

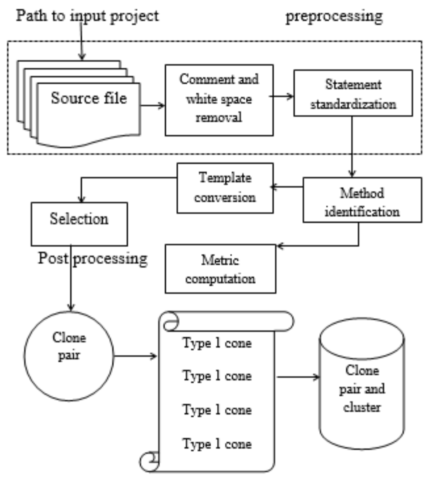

Fig.2: Schematic diagram of Clone Manager [14]

a) **Plagiarism Detection**: In Projects Plagiarism Detection is one of the closely related areas of clone detection [11]. Clone detection techniques can be used in the domain of plagiarism detection. A clone detection tool such as token-based CC Finder has been applied in detecting plagiarism.

b) **Copyright Infringement:** The problem of detecting source code copyright infringement is watched as a code similarity measuring problematic between software systems. Clone detection tools therefore, can be applied or can simply be adapted in noticing copyright infringement [9].

c) **Clone Detection:** in Models Clone detection is also used in models [10]. Phenomenon is not restricted to code, but usually occurs in models in a very similar way. So it is likely that model clones are as unfavourable to model quality as they are to code quality. Clone detection is also used in data flow model. General Model. LIE Domain Model [10, 11].

V. DETECTION CLONE TECHNIQUES

Clone Detection is a dynamic research territory since 1990‟s [6]. Code clone location is exactly identified by upkeep of software, code overhauling & along these lines making the code more effective. The impression on clone identification demonstrates the dissimilar strategies & calculations to distinguish clones [3]. Clone position methodologies are comprehensively considered into five procedures which are portrayed underneath:

A. Text - Based Technique

One of the fastest clone detection approaches. It can easily deal with type 1 clone and with additional data transformation, the type 2 can also be taken care. The newer text- based clone detection technique that is based on dot plots. A dot plot is a two-dimensional chart where both axes list source entities. In this approach the lines of a program are comparison entities. If x and y are equal there is a dot at coordinate (x, y). Two lines are considered equal if they have the same hash value. Dot plots can be used to visualize clone information; diagonals in dot plots are identified as clone. The detection of clones in dot plots can be automated, & string-based dynamic pattern matching is used to compare whole lines. Diagonals that have gaps indicate type 3 clones.

B. Token - Based Technique

Token based technique is similar to text based technique. However, instead of taking a line of code as representation directly, a lexical analyzer converts each line of code into a sequence of token [12]. After data values & identifier are substituted by some special tokens. The token sequences of lines are compared efficiently through a suffix tree algorithm. The result is also presented in dot plot graph. This technique is somewhat slower than text based technique because of the tokenization step. However, applying suffix tree matching algorithm the time complexity is similar as text based technique. By breaking line into tokens, it can easily detect both type 1 and type 2 clone, with token filter applied. The result of clone can be controlled very precisely, for instance, skip any uninterested information.

C. Abstract Syntax Tree (AST) - Based Technique

In AST based technique the program is parsed into a parser tree or an abstract syntax (AST) with a parser of language of interest. Then, using a tree matching method similar sub trees are examined in the tree. When a match is found corresponding so code of the alike sub trees are returned as clone couples or clone classes. The information is available in the parse tree of AST. The variable names & literal value the source code are discarded during the tree representation: still it is possible to employ more sophisticated clone detection tools. 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.

VI. CONCLUSION

Code clone is a big problem. A copy and paste activity which is done by programmer is the main reason of code cloning. It looks like a simple and effective method, these copy and paste activities are not documented. Which create a bad effect on the software quality and duplication also increase the bug probability and maintenance problem?. Cloning of code has become one of the easiest ways to complete a project, who does not want to invest their time on doing programming their project. It’s a loss for those who really work hard for the project coding. The date no such method has present who can evaluate the cloning for several languages with one piece of code. In this paper also present the different types of code clone techniques like text based Token based comparison and Abstract Syntax Tree Based Comparison. In this paper describe how to effective these technique in code clone.

VII. REFERENCES

[1]. Balazinska, Magdalena, Ettore Merlo, Michel Dagenais, Bruno Lague, and Kostas Kontogiannis. (1999) "Measuring clone based reengineering opportunities."In *Software Metrics Symposium, 1999.Proceedings.Sixth International*, pp. 292-303.vol.no.5, IEEE.

[2]. Baker, Brenda S. (1995) "On finding duplication and near-duplication in large software systems." In *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pp. 86-95. Vol.no.95,IEEE.

[3]. Liu, Chao, Chen Chen, Jiawei Han, and Philip S. Yu. (2006) "GPLAG: detection of software plagiarism by program dependence graph analysis."In *Proceedings of the 12th ACM*

SIGKDD international conference on Knowledge discovery and data mining, pp. 872-881. Vol.no. 6, ACM.

[4]. Keivanloo, Iman, JuergenRilling, and Philippe Charland (2011). "Seclone-a hybrid approach to internet-scale real-time code clone search."In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, pp. 223-224.Vol.18. IEEE.

[5]. Yuan, Yang, and Yao Guo. (2011) "CMCD: Count matrix based code clone detection."In *2011 18th Asia-Pacific Software Engineering Conference*, pp. 250-257.Vol no 18 IEEE.

[6]. Chatterji, Debarshi, Jeffrey C. Carver, Beverly Massengil, Jason Oslin, and Nicholas A. Kraft (2011). "Measuring the efficacy of code clone information in a bug localization task: An empirical study."In *2011 International Symposium on Empirical Software Engineering and Measurement*, pp. 20-29. Vol.no. 11,IEEE.

[7]. Higo, Yoshiki, Ueda Yasushi, Minoru Nishino, and Shinji Kusumoto (2011). "Incremental code clone detection: A pdg-based approach." In *2011 18th Working Conference on Reverse Engineering*, pp. 3-12.vol.no 18 IEEE.

[8]. C .K. Roy and JR. Cord (2007), "A Survey on Software Clone Detection Research "*Queen's School of Computing Technical Report No*. 2007-541. vol. 115.

[9]. B. Baker (1992). "A Program for Identifying Duplicated Code." *in Proceedings of Computing Science and Statistics*: 34th Symposium on the Interface. Vol. 24. pp. 49-57.

[10]. Kapser, Cory J., and Michael W. Godfrey (2008). ""Cloning considered harmful" considered harmful." *Empirical Software Engineering* 13, vol.no. 6 : 645-692.

[11]. Merlo, Ettore (2007). "Detection of plagiarism in university projects using metrics-based spectral similarity."In *Dagstuhl Seminar Proceedings*.SchlossDagstuhl-Leibniz-ZentrumfürInformatik,.

[12]. J. H. Johnson (1994) "Visualizing Textual Redundancy in Legacy Source, "*In Proceedings of the 1994 Conference of the Centre for AdvanceclSmclies on Collaborative research*(C.-l$Co.~'v"94). Vol.no 94, pp. 171-183. Toronto. Canada. 1994.

[13]. Roy, Chanchal K., James R. Cordy, and Rainer Koschke(2009). "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach." *Science of Computer Programming* 74, no. 7: 470-495.

[14]. Kodhai, Egambaram, and Selvadurai Kanmani (2014)."Method-level code clone detection through LWH (Light Weight Hybrid) approach." *Journal of Software Engineering Research and Development* 2, no. 1 : 1.