Volume 5, Issue 8, August 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Analysis of Code Clone Detection in Software's Using Metric Based Methods

Jagjit Singh, Er. Sukhpreet Kaur
CSE, Guru Granth Sahib world University
Fatehgarh Sahib, Punjab, India

Abstract - Software systems are getting more unpredictable as the framework develops where keeping up such framework is an essential concern for the software commerce. Code clone is a specific type of the elements which are making software maintenance further challenging. The duplicate of code fragments and after that reuse by sticking with or without minor changes or adjustments and this kind of reuse methodology of existing code is called code cloning and the pasted code fragment (with or without alterations) is known as a clone of the first one. The significant danger of cloning is that it improves the maintenance process. Cloning is fundamentally the method for software reuse and also this becomes the fundamental need of today's surroundings. That is the reason why code cloning has been widely utilized as a part of vast software businesses. So to identify clones and refactor them is a noteworthy concern. Numerous methodologies have been created to distinguish clones. In this paper we have surveyed about clone detection its various types, clone detection process as well as previous techniques used in clone detection in brief.

Keywords-Code clone detection, Code Clones, Code Clone Detection Process, Clone detection techniques.

I. INTRODUCTION

In PC software, we could possibly have dissimilar sorts of repetition. We ought to note that not every sort of redundancy is dangerous. There are diverse types of redundancy in software. Software embodies both programs and information. At some particular times redundant is utilized correspondingly in the sense of unessential in the software engineering works. Redundant code is as well repeatedly misleadingly entitled as cloned code despite the fact that it point towards that one bit of code which is probably imitative from the supplementary one in the original meaning of this particular word. Even though cloning prompts to redundant code, not each particular redundant code is a specific clone. A specific code clone is a particular code share in source records which is indistinguishable or like another. Replication of code happens oftentimes amid the improvement of huge software frameworks. A Clone Detection methodology is to figure out the reused piece of code in any application to keep up. Different sorts of clones are being recognized by clone discovery strategies. Since clone recognition was developed, it gives better results and diminishes the multifaceted nature. Code cloning is a type of software reuse, as well as exists in practically each particular software project. This specific ad-hoc system of reusability comprises in photocopying, in addition to sooner or later altering, a particular block of present code which apply a portion of mandatory functionality. Duplicated blocks are known as clones as well as the action of replicating, which also including slight modifications, is assumed as specific type of cloning. The consequences of numerous studies [5] show that an extensive segment (5-10%) of the basic code in huge software frameworks is duplicate code. Software clone is generally produced by programmer's copy in addition to paste actions. Programmers frequently duplicate and paste a current comparable code and further adjust it as indicated by their need. Code cloning or the demonstration of replicating code parts and making minor, nonfunctional adjustments, is a no doubt understood issue for advancing software frameworks prompting to copied code sections otherwise code clones. The standard working of the framework is not influenced. But then again auxiliary improvement possibly will turn out to be excessively costly.

Table I Clone detection Approaches

Name	Portability	Accuracy	Integrability	Scalability
Text	High	High	Low	Relative to comparison algorithm
Token	Medium	Low	High	High
AST	Low	High	Low	Relative to comparison algorithm
PDG	Low	High	Medium	Low
Metric	Relative to defined metric	High	Medium	High

II. VARIETIES OF CLONES

Code clone could be of any sort that all rely on upon the developer's method and ability of utilizing the code which differs from replicating as it is to duplicate the code however with some change which would be done at diverse level in the technique. In software system code pieces predominantly demonstrates two sort of similitudes. They are said to be comparable if their code content matches or they can be comparative on their functionalities bases if the conduct among them coordinate. Primarily clones are of four sorts out of which initial three sorts are under textual similarity and the last sort is under functional similarity.

A. Textual Similarity

This is based on the text based resemblance we differentiate the subsequent sorts of clones:
Type I (exact clones): Undistinguishable code portions with the exception of the varieties in whitespace (may be likewise varieties in design) and remarks.
Type II (retitled/parametrized clone): Structurally/grammatically indistinguishable sections aside from varieties in layout, identifiers, remarks, literals, and sorts.
Type III (gapped clone): Copied parts with further alterations. Statements could possibly be added, transformed, or detached along with various dissimilarities in layout, identifiers, remarks, literals, and sorts.

B. Functional Similarity

If in any condition, the functionalities of the two distinct code parts are indistinguishable or else comparative i.e., they have comparable pre as well as post circumstances, we particularly call them as a semantic clone sand which is eluded as Type IV clones.
Type IV (semantic clone): Two or more code fragments which could accomplish the same computation however actualized through diverse syntactic variations.
The result of a clone detection device is typically in terms of code part groupings that are moreover done through clone pair or else through clone cluster alongside their location data. A Clone Pair (CP) is a couple of code parts that are like one another under a characterized similarity measure. A Clone Cluster (CC) is a gathering of code partitions or parts which are pair-wise comparative under a characterized similitude measure.

III. METRIC SPACE BASED CLONE DETECTION

In this section, we will give a brief knowledge related to metric space, metric space distance and how to conduct proximity query reliant on specific provided metric space distance formula.

A. Metric space and metric space distance

A metric space is a pair (A, d), where A stands for an infinite set of valid objects, and d (a,b) stands for a function calculating distance between a and b, which are two members in X . d (a, b) should have below properties so it can be used to measure metric space distance:

$$(x^1)\ \forall a, b \in A, d(a,b) \geq 0, positiveness;$$
$$(x^2)\ \forall a, b \in A, d(a,b) = d\,(y,x), summary;$$
$$(x^3)\ \forall a \in A, d(a,a) = 0, reflexivity;$$
$$(x^4)\ \forall a, b, c \in A, d(a,b) \leq d(a,c) + d(c,b),$$

B. Triangle inequality

Given a k - dimension metric space, its members are identified using k real valued coordinates (a1… ak). For specific member (a1,…..,an) and (b1,…..,bn), there present several paths to calculate, metric distance among them on diverse norm numeral.

IV. CLONE DETECTION PROCESS

A. Preprocessing

In the initial stage of clone detection process the objective source is distributed as well as comparison area is determined. The principle targets to be considered in this stage are evacuating uninteresting parts, deciding the source units and deciding the correlation unit.

B. Transformation

In the following stage, the source code's correlation unit is changed to another intermediary internal demonstration of ease of comparison or else for separating the similar properties.

C. Match Detection

The altered code is specified as input in the direction of an appropriate comparison unit, wherever it is matched and equated with one other with the intention of finding the appropriate match. The order of given comparison units are utilized to add up the adjacent comparable units to transform in greater units. The amount produced of the comparison unit is a record of matches concerning the changed code.

D. Formatting

In this phase, the clone pair list acquired as for the changed code is then changed over to a clone pair records got as for the first code base.

E. Pre-processing

This phase assists in the false positives pass through a filter in two methods for instance manual analysis as well as visualization tool.

F. Manual Analysis

Once the first source code has been extricated, the underdone code of the clones of the clone couples imperiled to the particular manual analysis, with the intention of filtering out all the possible false positive.

G. Visualization

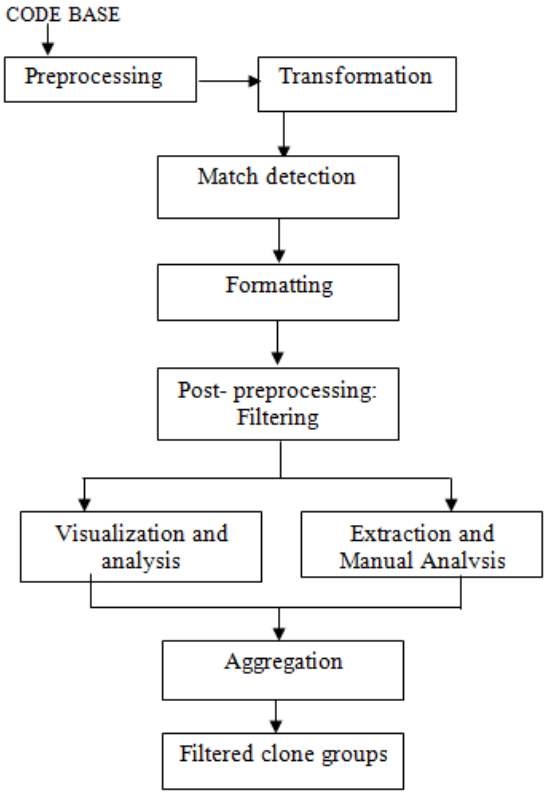

Fig 1: Clone detection process

To accelerate the manual investigation in sifting through the false positives, visualization tool are utilized to imagine the clone pair.

H. Aggregation

To perform certain investigation, there is a need to decrease the measure of information, so the clone sets ought to be collected to groups, classes; clubs of clones or clone bunches and so forth.

V. PREVIOUS TECHNIQUES

Clone Detection is a dynamic research territory since 1990"s [1] [11]. Code clone location is specifically identified with upkeep of software, code overhauling and along these lines making the code more effective. The overview on clone identification demonstrates the different strategies and calculations to distinguish clones [4], [12]. Clone location methodologies are comprehensively characterized into five procedures which are portrayed underneath:

A. Textual Technique
B. Token based
C. Abstract Syntax Tree Based
D. Program Dependency Graph
E. Metric Based

A. *Textual Technique:* This procedure is merely based on the text or string methods, so in this approach the raw source code is measured as sequence of lines and strings. Code segments are matched with each other to perceive the same sequences of text or strings, which not correlated to structural elements of the language. The detected sequences are

return as clone pair by the detection technique. Some text based approach perform a slight of transformation or normalization on the code remains before setting off the assessment process, whereas in general the row source code is straight used in the coordinated process[15]. The following are some usually used filtering/transformation or normalization in some approaches:

- *Normalization:* basic normalization can be functional on the raw source code.
- *Whitespaces:* considers and removes all whitespaces counting tabs, new line and other blanks spaces.
- *Comments:* eliminate all comments used in the source code.

B. *Token Based:* Token approaches transform or parsed or flexed the source code into a series of tokens using compiler style lexical analysis. These tokens are scanned [16] to detect duplicate subsequences of tokens, as a product the matched tokens from the original code remains are retrieved as clones. This technique is much superior to the textual approach in term of detecting the minor code change such as formatting and spacing.

C. *Abstract Syntax tree based:* Tree matching approach find clones by finding similar sub trees. Variable names, accurate values and other leaves in the source may be abstracted in the tree depiction, allowing for more complicated detection of clones. One of the pioneering tree matching clone detection technique is Baxter et al.'s Cloned [17]. A compiler generator is used to generate a constructor for annotated parse trees. Sub trees are then hashed into buckets. Only within the same bucket, sub trees are compare to each other by a tolerant tree matching. The hashing is optional but reduces the number of necessary tree comparisons drastically. This approach has been adapted by the AST-based clone detectors. The main differences from Cloned are ccdiml's explicit modeling of sequences, which eases the search for groups of sub trees that together form clones, and its exact matching of trees.

D. *Program dependency:* Program dependence graphs [18], which have proven useful in software engineering application such as testing, debugging, and preservation, model potential semantic dependences between program elements. However, they do not model the strengths of any corresponding statistical dependences between the plan elements. This paper makes the case that by augment program dependence graphs with statistical dependence information in the principled way provided by probabilistic graphical models, it is possible to considerably increase the utility of program dependence graphs in some software manufacturing applications. Probabilistic graphical models have proven useful in several fields due to their ability to model both the presence of sure dependences between variables of interest and the way in which the variables are probabilistically conditioned on other variables. A probabilistic graphical model derived from a program reliance graph provides a natural framework for modeling both the attendance of dependences and their statistical strengths.

E. *Metrics based:* Metrics based technique gather a number of metrics for code garbage and then compare metrics vectors rather than code or ASTs directly [19]. One popular procedure involves fingerprinting functions, metrics considered for syntactic units such as a class, function, method and statement that yield values that can be compare to find clones of these units. In most cases, the source code is first parsed to an AST or control flow graph on which the metrics are then calculated.

Kodhai. E et.al [10] proposed an amalgamation of textual in addition to metric analysis of a source code for the location of a wide range of clone in a given arrangement of section of java source code. Different semantics had been defined and their qualities were utilized amid the discovery process. This metrics with textual analysis gives less unpredictability in discovering the clones and provides exact results

Girija Gupta et.al [11] introduced work metric based methodology which is utilized to distinguish the potential clones and after that upgrading of code is done to diminish the recognized clones. Since the byte code is taken which changes over the source code into uniform representation and it is given as an information to the instrument for computing metric's value, so up to some degree it has the capacity distinguish the semantic clones.

D. Gayathri Devi et.al [12] proposed a strategy planned for identification of control structure. Distinguishing these clones brings more advantages towards support and reuse. They recognize and evaluate the similarity in source code that is imperative for numerous application specifically of code detection. Their paper introduce a clone discovery procedure in view of information mining methodology. Their work recognizes the code clones that happen in control structures, for example, for, while and do statement.

Md. Sharif Uddin et.al [13] took a current code cloning framework and enhanced the time execution by a request of extent utilizing simhash, and exhibited its possibility for utilization with vast frameworks, for example, the Linux Kernel. Their experience affirms that diff-based examination (as utilized as a part of NiCad) functions admirably for discovering Type-3 clones. Be that as it may, this accompanies both an expense in time many-sided quality, and lower review.

Bruntink et.al [14], proposed a few clone identification methods are assessed regarding discovering known cross-cutting concerns in C programs with homogeneous executions.

VI. METRIC SPACE BASED CLONE DETECTION

A pair i.e. (X, D) which is matrix based. In this pair X defines for the infinite valid object sets. d(x,y) is the function that defines x and y which is the distance between them. The d(x,y) have the following properties for measuring the metric space distance:

$$(p1)\,\forall x, y \in X, d(x,y) \geq 0; positiveness;$$

$$(p2)\,\forall x, y \in X, d(x,y) = d(y,x); symmetry;$$

$$(p3)\,\forall x, y \in X, d(x,x) = 0; reflexivity;$$

$$(p4)\,\forall x, y, z \in X, d(x,y) \leq d(x,z) + d(z,y); positiveness;$$

The formulas used for calculating the matrix of different norm numbers are as below:

For 1- norm;

$$\sum_{i=1}^{n} x_i - y_i$$

For 2-norm;

$$(\sum_{i=1}^{n} x_i - y_i^{\,2})1/2\text{r}$$

For p-norm;

$$(\sum_{i=1}^{n} x_{i-}\, y_i^{\;p})$$

For Infinite norm:

$$\frac{lim}{p \to \infty} (\sum_{i=1}^{n} (x_i - (y_i)^p$$

VII. CONCLUSION AND FUTURE SCOPE

The proposed paper presents various code cloning techniques to detect functional clones with the computation of metrics like Textual comparison, Token based comparison, Abstract Syntax Tree Based Comparison, Program Dependency Graph Comparison, Metric Based Comparison. In addition to this various types of clones has also been discussed in proposed survey paper. The main aim of this paper is to highlight the code clone issue in software components.

REFERENCES

[1] Gilad Mishne and Maarten de Rijke, "Source Code Retrieval Using Conceptual Similarity", In Proceeding of the 2004 Conference on Computer Assisted Information Retrieval (RIAO'04), pp. 539-554, April 2004.

[2] Chanchal Kumar Roy and James R. Cordy, "A Survey on Software Clone Detection Research", Technical Report No. 2007-541, School of Computing Queen's University at Kingston Ontario, Canada, September 26, 2007.

[3] G. Antoniol, G. Casazza, M. Di Penta, E. Merlo, Modeling Clones Evolution through Time Series, in: Proceedings of the 17th IEEE International Conference on Software Maintenance, ICSM 2001, pp. 273-280 (2001).

[4] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, Comparison and Evaluation of Clone Detection Tools, Transactions on Software Engineering, 33(9):577-591 (2007).

[5] R. Koschke, Survey of Research on Software Clones, in: Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 24pp. (2006).

[6] C.K. Roy and J.R. Cordy, An Empirical Study of Function Clones in Open Source Software Systems, in: Proceedings of the 15th Working Conference on Reverse Engineering, WCRE 2008, pp. 81-90 (2008).

[7] C.K.Roy and J.R. Cordy, WCRE'08 Clones,http://www.cs.queensu.ca/home/stl/download/NICADOutput/ Last accessed November 2008.

[8] C.K. Roy and J.R. Cordy, Towards a Mutation-Based Automatic Framework for Evaluating Clone Detection-Tools, in: Proceedings of the Canadian Conference on Computer Science and Software Engineering, C3S2E 2008, pp. 137-140 (2008).

[9] 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, JULY 2002.

[10] Kodhai. E, Perumal. A, and Kanmani. S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones" International Journal of Computer Communication and Information System Vol2. No1. ISSN: 0976–1349 July – Dec 2010

[11] Girija Gupta, Indu Singh, "A Novel Approach towards Code Clone Detection and Redesigning" © 2013, IJARCSSE Volume 3, Issue 9, September 2013.

[12] D. Gayathri Devi and Dr.M.Punithavalli, "Developing a Novel and Effective Clone Detection Using Data Mining Technique" © 2012, IJARCSSE, Volume 2, Issue 8, August 2012.

[13] Md. Sharif Uddin, Chanchal K. Roy, Kevin A. Schneider, and Abram Hindle, "On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems" 30 June, 2011.

[14] M. Bruntink, A. Deursen, R. Engelen and T. Tourwe, "On the Use of Clone Detection for Identifying Crosscutting Concern Code", Transactions on Software Engineering, 31(10):804-818 (2005).

[15] Roy, Chanchal Kumar and James R. Cordy. A Survey on Software Clone Detection Research. Canada: Queen's University, 2007.

[16] Patenaude, J-F, Ettore Merlo, Michel Dagenais, & Bruno Laguë, (1999) "Extending Software Quality Assessment Techniques to Java Systems", 7th International Workshop on Program Comprehension (IWPC'99), pp 49-56

[17] I. Baxter, A. Yahin, L. Moura, M. Anna, Clone detection using abstract syntax trees, in: Proceedings of the 14th International Conference on Software Maintenance, ICSM 1998, 1998, pp. 368–377.

[18] J. Ferrante, K.J. Ottenstein, and J.D. Warren, "The Program Dependence Graph and Its Use in Optimization," ACM Trans. Programming Languages and Systems, vol. 9, no. 3, pp. 319-349, July 1987.

[19] J. Patenaude, E. Merlo, M. Dagenais, B. Lague, Extending software quality assessment techniques to java systems, in: Proceedings of the 7th International Workshop on Program Comprehension, IWPC 1999, 1999, pp. 49–56.

