IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

Design and Analysis of a Hybrid Technique for Code Clone Detection

Jai Bhagwan[1], Kumari Pramila[2]

Assistant Professor, CSE Department, GJU S&T, Hisar, Haryana, India[1]

PG Student, CSE Department, GJU S&T, Hisar, Haryana, India[2]

Abstract: Nowadays, the software industry is getting more complex since the software systems are growing massively. In order to meet the software demand, a developer often reuses the existing code. This reuse of code results as a code cloning. Code cloning can be defined as a copying and pasting activity of code sections by doing a minor or no modification. These modifications can be done in terms of addition, removal, renaming. Tremendous demands from software industry could be one of the reasons for a developer to use the cloning beside his lazy behavior in writing a code from new scratch. Though the cloning reduces the development time and efforts but it impacts the maintenance cost of the software in terms of software readability and changeability. So, the demand for code clone detection arises in the software industry to improve the readability and changeability in software. A number of clone detection methods exist today to find out code clones in a software system. This paper presents a research work carried to design a hybrid technique that combines the metric based clone detection approaches with text-based clone detection approaches and gives a better result in accounts of precision, recall, and accuracy.

Keywords: Code clone, duplicate code, cloning, clone detection

I. INTRODUCTION

We can define the code clone, as a computer programming term that is used when there is a multiple occurrence of a sequence of source code either within the program itself or in some other programs [1][8][12]. Code clones are the semantically and syntactically similar results of copy-and-paste activities [4]. The reason behind the cloning can be intentional or unintentional [2]. In software development, a developer usually, copies a section of code fragment and pastes this code fragment to another code section by doing a no or minor modification. This whole copy-and- paste activity can be termed as software cloning and pasted code (modified code) as well as copied code can be termed as cloned code of each [2][4]. The term "code clone" does not have a generic or precise definition for code clones, each researcher defines cloning as their own.

As, a canonical example of code cloning, we often take the example of copy and paste activity but cloning is not a result of this copy-paste alone. Code clones may be invoked in software programming as idioms of language or libraries, common library API's or framework usage, or even on common examples based on implementations. Likewise, all copy-and-paste activities need not be considered as code cloning. Copying and pasting of trivial code sections like block statement or for loops are not considered as code clone [8].

Today, the software industry is getting more complex since the software systems are growing tremendously, so the software companies need a huge amount of the maintenance in terms of cost and efforts of existing software systems [4][10].

Software maintenance in software engineering is defined as the modification (corrective, adaptive, perfective, or preventative) of a software product after delivery to correct faults and improve the performance or other attributes various research studies have shown that maintenance of the software systems with code clones is more difficult than a non-cloned code system.

In a software system, typically, about 60% of cloned code is the modified code [1] and around 7% to 23% of the code is the copied and pasted code [2] [3][4].

Generally, it is believed that cloning introduces additional maintenance efforts like the maintenance cost will be affected if a change made to one code fragment is to be propagated in the another fragment of the program. Further, problems are raised in the location and maintenance.

There is no doubt that, code cloning is a "bad smell" kind of [10] software design approach. So, there is an insistence of code clone detection approaches for precise and effective information of clones in system software.

A. Terminology associated with Code Cloning.

1) Code Fragment (CF). A code fragment is a sequence of code lines of any granularity, for example, the sequence of statements, begin-end block or function definition etc. [8].
2) Code Clone (CC). A code fragment (CF1) is a clone of another code fragment (CF2), if f (CF1) = f (CF2), where f is a predefined function of similarity [8].

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

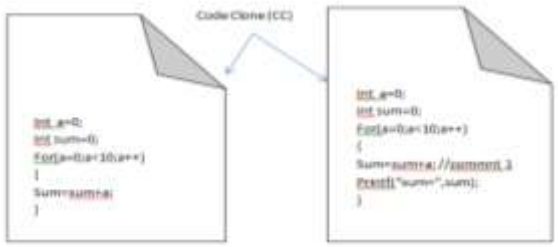

Fig 1. A Code Clone Example [1]

3) Clone Pair (CP). A pair of identical code fragments [4].
4) Clone Set (CS). A set of identical fragments [2].
5) Clone Relation (CR). A clone relation is an equivalence relation defined on code portions. This pair of clone portion is called clone pair. A clone class is a maximal set of code portions in which an equivalence clone relation exists between any pair of code portions [3].

B. Classification of Code Clones.

Broadly, code clones can be categorized into two categories i.e. the clones that are identical syntactically and the other types of clones are related semantically [4] [12]. Each of these categories is described below:

1) Syntactically Similar Clones**:** These are the structurally or textually similar code fragments having minor modification (white space removal, adding more comments, adding one or more sequence of code to the copied code fragments etc.) Type-I, Type -II and Type-III clones fall under this category [12].
   i. Type-I (Exact clones) - Textually identical code segments except for variations in layout, whitespace, and comments [2][3].
   ii. Type-II (renamed/parameterized) - Textually identical code segments except for variations in literals, identifiers, whitespace, types, layout and comments [1][3].
   iii. Type-III (near-miss clones) - Copied segments with further modifications such as added, changed or removed statements, in addition to variations in literals, identifiers, types, whitespace, layout, and comments [3][4].
2) Semantically Similar Clone: These are code fragments that are similar in computation but have syntactic variation. These are also known as Type-IV code clones [8].

C. Clone Detection Approaches.

Clone detection has been an active area of research since 1990's. A number of clone detection approaches have been proposed in the literature. The clone detection approaches can be classified into four main categories: textual, lexical, syntactic and semantic [8].

Each of these approaches with their related research is described below:-

1) Textual Approaches: Textual approaches are text-based approaches that are using a little or no transformation on the source code before its actual comparison. In most cases, the detection processes directly employ source code in their detection method [1] [8].

Limitations of text-based Approaches [4][8]:

   i. A line-by-line method cannot handle identifier renaming.
   ii. Code segments having line breaks are not recognized as clones.
   iii. Adding or removing brackets can create a problem during comparing two code portions when one of the two portions has brackets and the second portion does not have brackets.
   iv. The text-based approaches cannot be used in source code transformation, so it needs some normalization to improve recall without reducing precision rate.

2) Lexical Approaches: Lexical approaches are token-based approaches that transform source code into a sequence of "tokens" with the usage of a lexical analyzer. The transformed token sequence is then run for duplicated subsequences of tokens and the comparable original code is returned as clones. Lexical approaches are robust over minor code changes like renaming, formatting, and spacing than text-based approaches. The approach can detect Type-I and Type-II clones and, Type-III clones can be further detected by concatenating Type-1 and Type-2 clones [8].

Limitations of Lexical Approaches:

   i. Token-based approaches rely upon the order of program lines. Whenever the order of statements is modified in copied code, copied code can't be detected [1][3].
   ii. Code clones with added or removed tokens along with the swapped lines can't be detected using these techniques as the clone detection technique is more focused on tokens [3].
   iii. Token-based approaches cost more in terms of space and time complexity than textual approaches since a source line comprises of several tokens [1].

3) Syntactic Approaches: A parser is used to convert the source programs into a parse tree or abstract syntax trees (AST) [8] [11], which are then, processed either by using a tree match or structural metrics match to find clones.
   i. Tree matching approaches - These are tree-based approaches that detect clones by detecting similar sub-trees. Literal values, variable names and other tokens in the source code are abstracted in a tree representation, for detection of clones [9].
   ii. Metrics-based Approaches - Metrics-based approaches calculates a number of metrics from code fragments and then compares metrics vectors directly. Metrics are calculated for syntactic units such as classes, loops, functions and statements [1][2][3].
   These metric values can now be used to detect clones. In most cases, AST [8] or control flow graphs (CFG) are used to parse the source code, on which the metrics are then calculated. [3].

DOI 10.17148/IJARCCE.2016.51181

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

Limitations of Syntactic Approaches:

i. Tree-based techniques can't handle literal and identifiers values for clone detection in ASTs.
ii. Tree-based techniques cannot detect reordered statement clones.
iii. A metric-based technique requires a parser or a PDG generator for metrics values computation.
iv. Based on matrices alone two code fragments may not found to be similar code fragments even if they have similar metric values.

4) Semantic Approaches: Static program analysis is used to provide more precise information in semantics-based clone detection approaches. In some approaches, a PDG (program dependency graph) represents a program. The nodes are representing statements and expressions, while the edges are representing control and data dependencies [4][8][9].

Limitations of Semantic Approaches:

i. PDG-based approaches are not scalable for large systems [8].
ii. A PDG generator is required in PDG-based approaches. Graph matching that is used in PDG-based techniques is expensive [8].

5) Hybrid Approaches: Hybrid approaches are the combination of any two earlier discussed approaches [1][4][8]. For example, syntactic approaches can be merged with the semantic approach to achieve their combined goals [7] [11].

II. LITERATURE REVIEW

Manpreet Kaur et al. [1] proposed a code clone detection technique for efficient detection of type I, type II and type III clones. They segmented source code into a number of functions for clone detection purpose. Their proposed tool is built in MS.Net framework version 4.0 by using visual studio 2010.
Potential clones were detected by calculating a number of effective lines, the number of loops used, the number of function calls, etc. Gitika et al. [2] presented an approach to detect potential clones from software. Potential clones are those parts of the code which are the candidates for a clone but are not necessarily being cloned. This approach can be used to reduce complications with other approaches and is quite simple to use.

The proposed clone code detection approach gave results on method level metrics extracted from source code. Source Monitor is the name of the tool which was used to calculate the required method level metrics. After calculating the required metrics, the potential clones were detected. The authors had used a chat server system developed in java language to detect potential clones. This code clone detection approach was applied only to a part of the software system in which potential clones had been detected rather than applying on the whole system. Amandeep Kaur et al. [3] devised an algorithm which is used to identify duplicate code piece.

The proposed algorithm is based on metrics, which are being used to determine the complexity of a program related to the number of operands and operators in the program. The objective was to merge the metric based and text based techniques to design and analyse a new hybrid approach. In textual comparison, a line by line code comparison is used in post-processing rather than by taking token or word.

Visual Basic 6.0 programming language was used in user interface design for detecting code clone in an application. The software metrics which are used to compute and analyse were the number of operands, number of operators, the number of source lines of code etc.

The proposed algorithm gave a light-weight technique to detect functional clones by computing metrics values and then combining with simple textual analysis technique. With the employment of metrics in the proposed approach, a signified reduction was observed with the existing one. A higher amount of recall was obtained as a result of string matching and textual comparison. K. Raheja et al. [4] had used the concept of hybrid clone detection approach. The proposed approach used an algorithm for detecting duplicity in the software.

The algorithm was used to calculate sufficient information by computing software metrics that were required for the software application and then potential clone could be detected depending on the metrics that found a match. MCD Finder was the proposed tool that used to calculate the metrics of Java byte-code rather than using any transformed representation.

Also, the researchers found that semantic clones can be detected as byte code which was used for metric calculation. Token based approach was applied on potential clones for the detection of code clones. Perumal et al. [6] proposed a combined approach of the textual based and metric based code clone detection. A set of twelve metrics were used in this proposed technique to improve the precision and recall values. Use of metrics had reduced the total overhead in comparisons.

Metrics and the textual comparison were performed over different java source code fragments and it provided less complexity in finding the clones and gave accurate results. Kodhai E. et al. [7] proposed a light-weight metric-based approach combined with the textual comparison of the source code for the detection of functional clones. C source code was used as an input.

The method comprised of four steps namely, input and pre-processing, template conversion method identification, and metric computation. Various metrics (a set of 7 existing metrics) had been formulated and their values were utilized during the detection process. The obtained results were compared with the two other existing tools (Phoenix and NICAD) for the open source project

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

(Weltab). Phoenix-based tool reported six clone class matches while the proposed method reported eight exact match clone classes. Results were found to be similar with NICAD. The only difference with NICAD and proposed method was that the proposed method was built-in hand-coded parser without any external parser deployment while NICAD employed an external parser. C.K.Roy et al. [8] had provided a qualitative comparison and evaluation among various code clone detection techniques and tools.

The work was organised into a large volume of information for a coherent conceptual framework that began with background concepts and proceeded to a generic clone detection process and thereafter it gave an overall taxonomy of current techniques and tools. Further, the classification, comparison, and evaluation of the techniques and tools were discussed.

Jens Krinke et al. [9] proposed a new algorithm, KClone for clone detection that incorporated a combination of lexical and local dependence analysis to achieve precision. It also presented a report on the initial case study implementation result of KClone, which was used in experimenting.

The results indicated that the KClone was more capable of finding types-I, type-II, and type-III clones as compared to token-based and PDG-based techniques. Rainer Koschke et al. [11] compared the existing techniques and showed that token-based clone detection methods relied on suffix trees were extreme fast, but clone candidates yielded by this technique are often no syntactic units.

Current techniques based on abstract syntax trees (AST) were considerably less efficient but could find syntactic clones. The research described how suffix trees could be used to detect clones in abstract syntax trees.

The proposed approach could found the syntactic clones in linear time and space. K.Kontogiannis [13] had performed an examination and evaluation on the use of five data and control flow related metrics for identifying similar fragments. The metrics were used as code fragment signature.

Matching on such metric resulted in fast detection, which was used to locate code cloning instances even in the presence of modifications. The paper reported on experiments in three different software systems, conducted for retrieving code clone fragments.

M. Merlo et al. [14] used Metric-based technique to detect functional clones from the source code. A DATRIX tool framework was used to accomplish this. DATRIX tool is a source code analyser tool set that selects only the control flow metrics and data flow metrics. This proposed approach had been applied on two telecommunication monitoring system, for automatic detection, in which function clones were detected.

III. PROPOSED METHODOLOGY

Algorithm:
Input: File1, File2
Output: Clones
1: BEGIN
2: **Metric Calculation at file level**
for i←0, File1.Length **do**
file1Metric.calculateMetric();
for j←0 File2.Length **do**
file2Metric.calculateMetric();
3: **If** file1Metric== file2Metric
for i←0, File1.Length **do**
for j←0, File2.Length **do**
LevSim← LevDistance ($File1_i$, $File2_j$)
if LevSim==1 **then**
Clones ← $File1_i$,
4: **else**
5: Extract File1.class(), File1.method()
6: Extract File2.class(), File2.method()
7: Compute File1_class.metric(), Compute File1_method.metric()
8: Compute File2_class.metric(), Compute File2_method.metric
9: **for** i←0, File1_class.Length **do**
for j←0, File2_class.Length **do**
If classMetric[i]== classMetric[j]
LevSim← LevDistance ($class_i$, $class_j$)
if LevSim==1 **then**
Clones ← $class_i$
10: **for** i←0, File1_method.Length **do**
for j←0, File2_method.Length **do**
If methodMetric[i]== methodMetric[j]
LevSim← LevDistance ($method_i$, $method_j$)
if LevSim==1 **then**
Clones ← $method_i$,
11: END

IV. IMPLEMENTATION AND RESULT

A. Implementations

Fig 2. Architecture of Proposed Model

The proposed tool is an implementation of a hybrid approach that merges metrics-based clone detection approach and textual-based approach to detect clones. In addition to that we have also used Levenshtein Distance method to improve the results. The automated code clone

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

detection tool has been implemented in JDK 1.8, JRE 1.8 and Net Beans IDE 8. It takes two java source code files as input and compares these files based on a hybrid method which is given in proposed methodology sec III. The architecture of proposed tool is given in figure 2.

B. Results

We have done experiments with existing and proposed technique to detect the clones. The results of both approaches have been tested on two java source code files and their resultants are shown in table 1 and table 2. The results show that our proposed approach is better than the existing one in terms of parameters precision rate, recall, accuracy rate and error rates values which are obtained by using the equations (1), (2), (3) and (4) discussed in next section.

In Table 1 and Table 2:

TP is an abbreviation for true positive i.e. these are the actual clones which are detected by the tool.

TN is an abbreviation for true negative i.e. these are the actual clones which are not detected by the tool.

FP is an abbreviation for false positive i.e. these are not the actual clones but are detected as clones by the tool.

FN is an abbreviation for false negative i.e. these are not the actual clones and also the tool didn't detect these.

P is the sum of TP and FN.

N is the sum of FP and TN.

Table 1. Clones found using existing technique

		Detected Clones	
		Yes	**No**
Actual Clones	**Yes**	45 (TP)	33 (FN)
	No	0 (FP)	11 (TN)

Table 2. Clones found using proposed technique

		Detected Clones	
		Yes	**No**
Actual Clones	**Yes**	59 (TP)	19(FN)
	No	0(FP)	11(TN)

C. Performance Measures

For clone detection, the parameters precision, recall, accuracy and error are obtained using the equations given below [9]:

$$\text{Precision (P)} = \frac{TP}{TP+FP} \quad (1)$$

$$\text{Recall (R)} = \frac{TP}{P} \quad (2)$$

$$\text{Accuracy (A)} = \frac{TP+TN}{P+N} \quad (3)$$

$$\text{Error (E)} = \frac{FP+FN}{P+N} \quad (4)$$

Using the above four equations we have compared the performance of our proposed approach and existing approach based on table 1 and table 2. The obtained results are shown in table 3.

Table 3. Performance Table

Parameters to compare	Existing Approach	Proposed Approach
Precision	0.80	0.84
Recall	0.57	0.75
Accuracy	0.62	0.78
Error	0.38	0.22

V. CONCLUSION

In this paper, we have presented a hybrid technique that detects software code clones for Java programs on the basis of metrics and text-based approaches. The proposed approach looks for clones in the code at the file level, class level, and method level. The proposed approach detects potential clones on metric-based match. Potential clones are further compared line by line using a text-based approach to check whether the potential clones detected using metric based comparison are actually clones or not. We have implemented the existing and proposed techniques in the form of a tool named JHCCD written in java. Based on the results from this tool, we have observed that our proposed method is better than existing one in terms of parameters such as precision, recall, accuracy, error rate '0.80, 0.84', '0.57, 0.74', '0.62, 0.78', '0.38, 0.22' respectively for existing and proposed method.

The proposed method was tested on Java Source Code only and further it can be enhanced to support several kinds of programming languages. Additionally, more metrics can be introduced to enhance the results correction rate. Soft computing technique can be integrated to get optimization in case of a large dataset.

ACKNOWLEDGMENT

The authors express great thanks and acknowledge all the research scholars and other contributors whose valuable work have been used to understand the background and literature review in this work. Authors have used this information for a successful design and implementation of the proposed approach. At last but not the least we express our thanks to almighty for giving us a rapid motive and inner peace throughout our research.

REFERENCES

[1] Manpreet Kaur and Madan Lal, "Code Clone Detection Using Function Based Similarities and Metrics", International Journal of Emerging Research in Management & Technology, vol. 4, no. 7, pp. 156–159, Jul. 2015.

[2] Geetika and Rajkumar Tekchandani, "Detection of potential clones from software using metrics", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 4, Apr. 2014.

[3] A. Kaur and B. Singh, "Study on Metric Based Approach for Detecting Software Code Clones", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 1, Jan. 2014.

[4] K. Raheja and R. K. Tekchandani, "An efficient code clone detection model on Java byte code using hybrid approach",

Confluence 2013: The Next Generation Information Technology Summit (4th International Conference), 2013, pp. 16–21.

[5] M. Mondal, M. S. Rahman, R. K. Saha, C. K. Roy, J. Krinke, and K. A. Schneider, "An Empirical Study of the Impacts of Clones in Software Maintenance", IEEE 19th International Conference on Program Comprehension (ICPC), 2011, pp. 242–245.

[6] Kodhai, Perumal and Kanmani, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones", International Journal of Computer Communication and Information System, vol. 2, no. 1, July-Dec 2010.

[7] E. Kodhai, S. Kanmani, A. Kamatchi, R. Radhika, and B. VijayaSaranya, "Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics", International Conference on Recent Trends in Information, Telecommunication and Computing (ITC), 2010, pp. 241–243.

[8] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Science of Computer Programming, vol. 74, no. 7, pp. 470–495, May 2009.

[9] M. Matsushita, Jens Krinke, M. Harman, and David Binkley, "KClone: A Proposed Approach to Fast Precise Code Clone Detection," in Int. Workshop Detect. Softw. Clones, pp. 12–16, 2009

[10] C. Kapser and M. W. Godfrey, "'Cloning Considered Harmful' Considered Harmful," in 13th Working Conference on Reverse Engineering, 2006. WCRE '06, pp. 19–28

[11] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," in 13th Working Conference on Reverse Engineering, 2006. WCRE '06, pp. 253–262.

[12] I. Baxter, A. Yahin, L. Moura, M. Anna ,"Clone Detectio Using Abstract Syntax Trees," in Proceedings of 14th International Conference on Software Maintenance (ICSM), November 1998

[13] K. Kontogiannis, "Evaluation experiments on the detection of programming patterns using software metrics," in Proceedings of the Fourth Working Conference on Reverse Engineering, 1997.

[14] J. Mayrand, C. Leblanc, and E. M. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in , International Conference on Software Maintenance 1996, Proceedings, 1996, pp. 244–253

BIOGRAPHIES

Kumari Pramila graduated with B.Tech and presently she is pursuing M.TECH (CSE) in Guru Jambheshwar University of Science & Technology, Hisar, India. Her area of interests includes Software Engineering.

Jai Bhagwan received the M.TECH degrees in Computer Science and Engineering from Maharishi Markandeshwar University, Mullana in 2011. After this, he stayed in Maharishi Markandeshwar University, Mullana on the post of Lecturer in Information Technology department for a period of one year. Now, he is working as Assistant Professor in the Computer Science & Engineering Department in Guru Jambheshwar University of Science & Technology, Hisar, India. He has around 5 years of teaching experience. His areas of research are Cloud Computing and Software Engineering.