Code Clone Detection System using hybrid technique along with optimal intelligence techniques

Mehak Shahzadi, Kainat, Saira Munir

University of Central Punjab Lahore, Pakistan 2002

Abstract: A lot of search about code clones, detection systems, techniques and tools are available world widely. Code clones are the duplicate code of source codes which are formed by copying and pasting code fragments of different programs into new programs to avoid writing and wasting time. It also reuses of codes with some modifications in original source codes. So. Due to the copy paste, reuse and minor modifications in original source codes causes many defects into the software. code clones are verified as a main source of defects, which means they cause problems into the implementation and maintenance of software. Due to this reason most of research based on the detecting and removing these code clone's fragments in source code. In this research paper we introduce code clone detection system which uses hybrid approaches along uses optimal and intelligence technique. A hybrid technique which is combination of metric based and text-based detection technique of code clone gives a better result of accuracy, recall and precision whereas optimal and intelligence technique based on metric approach. We implement the metric-based approach extract the code properties i.e. LOC, function Overloading, function repetition, total number of functions, Global and local Variable with the help of PDG and AST tree techniques. These two different techniques give us better result in precision, recall and accuracy with back propagation neural network for 30 instance number from 90% to 98.8%.

---

1. INTRODUCTION

Code clone are defined as a computer programmed term that is a syntactically or systematically similar in original code of any software or program [1]. Code clones are basically duplication of source codes. It is also a reuse of existing code with some modification or some portion of whole code. [2] Code clones are now considered as a main source of defects in software. Which make complexity to handle, implement and maintenance of software [3]. So, a lot research based on detection of code clone in programs and proposed different approaches, techniques and tools for detection of code clone and give better accuracy, recall and precision results. The term 'code clone' does not have generic or precise definition for code clones because each researcher defines cloning as their own [2]. As a canonical example of code cloning, we often take the example of copy and paste activity but cloning is not a result of copy and 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 [4]. There is no doubt that, code cloning is a "bad smell" kind of [5] software design approach. So, there is an insistence of code clone detection approaches for precise and effective information of clones in system software [8]. The main issue in code clone is associated only with their similar code that is indirectly rather than directly which creates it problematic to identify them. Although, modifies like updates or covers that are often meant to affect every clone in the same path, are normally not functional to all of them consistent. The code quality declines and modification become more expensive and error-prone [3].

2. RELATED WORK

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. [6] 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 analyses 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 analyses 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. [1] had used the concept of hybrid clone detection approach. The proposed approach used an algorithm for detecting duplicity in the software.

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.

Metrics based techniques are complex because they only require comparison of some numerical data, i.e. metrics values of program units to find code clones. But these techniques may give false positives and even the clones with extra modification could be found by numerous detection techniques and tools. The survey of a systematic approach and analyzed in single type 3 clones and their dissimilar. The main focus, however on the difference in code metrics, variable and hided them only type substitution.

Komondoor et al. (2001) [6] author investigates the duplicate code from a software system with slicing technique. Duplicate modules in a software system are a normal thing. But it increases the software maintenance cost and efforts for stable a software system in production mode. The proposed approach detects all the similar clones and converted into a single module. That single module called for all the places to reduce duplicated code from the modules. This approach working with some graphs technique which helps to represent clone from a software system with the help of similar sub-graphs.

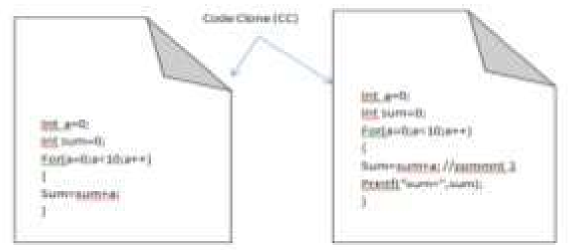
Jasmandeep Kaur implemented metric based technique used with the help of swarm and artificial intelligence techniques that described them from copy and paste code clones in a path that helps clone detection research.
He implemented these approaches or technique in JAVA, C++, and MATLAB for coding challenges (2017) [3].

Jai Bhagwan, Kumari Pramila, design hybrid technique for code clone detection using text-based and metric based approaches to give better results of accuracy, recall and precision. (2016)[8].

3. EXPLAINATION

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. [4].
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 [1].



```
int a=0;
int sum=0;
For(a=0;a<10;a++)
{
Sum=sum+a;
}
```

```
int a=0;
int sum=0;
For(a=0;a<10;a++)
{
Sum=sum+a; //comment 1
Printf("sum=",sum);
}
```

Fig 1. A Code Clone Example [1]

3) Clone Pair (CP). A pair of identical code fragments [1].
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 [6].

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 [1][7]. 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 [7].
i. **Type-I** (Exact clones) - Textually identical code segments except for variations in layout, whitespace, and comments [2][6].
ii. Type-II (renamed/parameterized) - Textually identical code segments except for variations in literals, identifiers, whitespace, types, layout and comments [1][6].
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 [4].

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 [4]. 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 [7] [4].

Limitations of text-based Approaches [1][4]:

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 [4].

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 [7][6].

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 [6].

iii. Token-based approaches cost more in terms of space and time complexity than textual approaches since a source line comprises of several tokens [7].

3) Syntactic Approaches: A parser is used to convert the source programs into a parse tree or abstract syntax trees (AST) [4] [8], 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 subtrees. Literal values, variable names and other tokens in the source code is 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 [7][1][6].

These metric values can now be used to detect clones. In most cases, AST [4] or control flow graphs (CFG) are used to parse the source code, on which the metrics are then calculated. [6].

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 have 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 [9][4][1].

Limitations of Semantic Approaches:

i. PDG-based approaches are not scalable for large systems [4].

ii. A PDG generator is required in PDG-based approaches. Graph matching that is used in PDG-based techniques is expensive [4].

5) Hybrid Approaches: Hybrid approaches are the combination of any two earlier discussed approaches [7][6][4]. For example, syntactic approaches can be merged with the semantic approach to achieve their combined goals [10] [11].

6) Classification of code clone:
Code clones are classified on the basis of tri-aspects which are used for expansion re-engineering and detection. They have re-iterated on the major prominent kind of clone, which prevents at the quality of time interval re-engineering. Following are the various code clones based on tri-aspects i.e.

a) Similarities b/w binary code parts.
b) Object code location in program.
c) Re-factor chances with the simulated code [12][3].

The similarity-based fragments are the majority of binary kind's i.e.

i) Binary code part could be verified on the basis of the same code of their execute program data [13]
ii) It could be same in their functionalities without being texture verification. However, texture similarity based clones are of four kinds as type-1, type-2, type-3, and type-4. An instance section the methods which are similar except the name and the techniques which are verified for the kinds of performance parameters integrated with larger similarity code clones. The type-4 code clone is based on the same functionalities, same output but different logics designed. The classification define the quality of methods of content has been copied same and also what kind of syntax tree elements have been changed. [3]

Con-QAT is a steady, free, open-source dash-board tool-kit also used in industry. It is normal aim simulation tool for several kinds of code measurement and analysis study.
ConQAT gives various specific code clone detection configurations for several programming languages, adding JAVA, C/C++, and COBOL. It has divide detection methods for Type-1 or Type-2 clones and Type-3 clones. They employed the previous method. Con-QAT has been described in various analyses in clone detection adding the study, they construct on [14].

Deckard uses an effective method for verifying same sub trees, and applies it to tree re-presentations of source code. It normally generates a parse-tree constructor to construct parse-trees required by its method. By a same parameter, it is possible to control whether only Type-1, Type-2 clones and Type-3 clones are detected. Deckard is a suitable tool described in-other analyses in adding the study, we construct on [15].

4. PROPOSED METHOD

We proposed a technique with the combination of hybrid technique along with optimal and intelligence technique in software engineering.
We first of all use the hybrid technique uses metric approach with text-based approach to detect the code clone fragments and portions on JAVA, C++ and C codes to get the better results of accuracy, recall and precision [8]. After applying hybrid technique we implement metric approach using swarm and artificial techniques of re-engineering. To get more generalized results of defect free program source code from code clones because it is not possible to get 100% defect free codes. First of all we select the JAVA, C++ code files which already detected by hybrid technique. Then we implement the feature approach on these files and then algorithm of optimizing the file code. Then apply the classification approach which is most appropriate for detection. Weather similarity based, object code location in program and refactoring. Then match with file code weather all code clone types detect and remove [3].

5. CONCLUSION & FUTURE WORK

In this research paper we proposed a hybrid technique along with optimal intelligence technique to detect code clone in JAVA, C++ and MATLAB to get more generalized results of the accuracy, recall and precision in source codes because code clones caused defected software program which makes problems in maintaining software. It's highly expensive to maintain the software defects.

Future research can be based on the implementation of this proposed method and on uses of different hybrid technique along optimal intelligence techniques.

6. REFERENCES

1. 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.
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. Jasmandeep Kaur Design “Code Clone Detection System uses Optimal and Intelligence Technique based on Software Engineering” International Journal of Advanced Research in Computer Science Volume 8, No. 5, May-June 2017.
4. 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.
5. C. Kapser and M. W. Godfrey, “„Cloning Considered Harmful‟ Considered Harmful,” in 13th Working Conference on Reverse Engineering, 2006. WCRE ‟06, pp. 19–28
6. 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.
7. 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.
8. Jai Bhagwan, Kumari Pramila”Design and Analysis of a Hybrid Technique for Code Clone Detection” International Journal of Advanced Research in Computer and Communication Engineering ISO 3297:9(2007) Certified Vol. 5, Issue 11, November 2016.
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. 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.
11. 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.
12. Göde, N., and Koschke, R., (2009), "Incremental clone detection."In Software Maintenance and Reengineering, 2009.CSMR'09. 13th European Conference on, pp. 219 228. IEEE.
13. Zhenmin, L., Lu, S., Myagmar, S., and Zhou, Y., (2006), "CP-Miner: Finding copy-paste and related bugs in largescale software code." IEEE Transactions on software Engineering 32, no. 3, pp. 176-192.
14. Koschke, Rainer, RaimarFalke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006.WCRE'06. 13th Working Conference on, pp. 253-262. IEEE, 2006.
15. Kim, M., Sazawal, V., Notkin, D., and Murphy, M., (2005), "An empirical study of code clone genealogies." In

ACM SIGSOFT Software Engineering Notes, vol. 30, no. 5, pp. 187-196.ACM