I J C T A, 9(41), 2016, pp. 555-563 ISSN: 0974-5572

Software Clone Detection: A review

Geetika Chatley*, Sandeep Kaur and Bhavneesh Sohal*****

ABSTRACT

Software cloning is the current issue in industries, making acknowledgement of clones a key bit of programming examination. Existing writing on the topic of software or programming clones is grouped comprehensively into various classifications. Utilization of existing code either by duplication and paste methods or by performing minor adjustments in the current code is known as software cloning. Programming clones may prompt bug engendering and genuine support issues. Clone sorts/types, techniques of clones and different procedures are included in this paper. Also this paper will serve as a guide to a potential client of clone identification strategies, to help them in choosing the right apparatuses or methods for their interests.

Keywords: Code clone, parsing, plagiarism, refactoring, reuse, semantic, syntactic, similarity.

1. INTRODUCTION

Programming code cloning is broadly utilized by designers to create code in which they have certainty and which diminishes advancement costs and enhances the product quality. Programming clone exploration in the previous years was generally centred on the recognition and examination of code clones, whereas exploration lately stretches out to the entire range of clone administration. Reusing programming through facsimileing and pasting is a nonstop torment in programming improvement regardless of the way that it engenders earnest maintenance quandaries. The process of duplicating a code is known as code clone. Some programmers perform code cloning intentionally or unintentionally during the development of an application or software. It has been studied that thirty percent of the code in most of the software companies is copied code. So it is essential to know that why the code has been duplicated, why there is a need to duplicate the code, how the copied or cloned code has a negative impact on the maintenance and development. For maintenance and development purpose, some phases like clone detection, analysis and maintenance have become a major area of research for many researchers. Though cloning has many advantages in software industries. It saves the programmer's time, reusability of code is easy for a beginner in the industry. But when we reuse the code, the overhead also increases. So cloning has a dark side as well. The huge matter of concern is the maintenance of the developed software. Sometimes the cost for maintenance exceeds more than the cost of development. The bug detection, virus recognition may also require the extraction of structured or semantically comparative clones. Every aspect has two faces like a coin has two sides so as the code cloning.

2. CLONE TERMINOLOGIES

2.1. Code fragment

Code section (some portion of a code) is any succession of code lines with or without remarks. It is identified by code fragment filename, code fragment begin line, code fragment end line.

* Computer Science and Engineering Lovely Professional University Phagwara, India, *Email: geetikachatley@gmail.com*

** Computer Science and Engineering Lovely Professional University Phagwara, India, *Email: sandeep.16827@lpu.co.in*

*** Computer Science and Engineering Lovely Professional University Phagwara, India, *Email: sohal.16042@lpu.co.in*

2.2. Code clone

At the point when a code part of document two is a clone of another code section of record one.

2.3. Clone pair

One arrangement of a code section is identical to other whether in a same file or in another file, they are said to be a clone pair.

2.4. Clone class

When many fragments are similar to each other or if there exists a clone- relationship between them then they make a clone class.

3. LITERATURE SURVEY

Software or code cloning has become a major area of research these days. Many researchers diligently exploring this topic and so many approaches have been developed to probe duplicate codes. These approaches are syntax-based [1], text- based[2] , graph based [3] and metric based [4].

Chanchal K. Roy et al. [5] have performed the comparison and evaluations of different techniques and tools. First of all the reorganizations and then evaluations of different approaches are being performed on the basis of some restrictions and on the basis of types of clones. This paper aids to detect different clone detectors. Chanchal K. Roy et al. [6] have performed discovery and examination of near-miss software clones. They have executed their task in four steps. They have developed a hybrid clone discovery technique , proposed a Meta model of clone sorts, furthermore they have given a situation based examination of clone recognition procedures and instruments. They have used NICAD tool which was not able to detect Type 4 semantic clone. A study from Ripon K. Saha et al. [7] have shown automatic detection of evolution pattern of both exact and near-miss clones by constructing their groups and they have developed a prototype "gCad" which is scalable to various clone detection tools. For detecting the change in pattern some of the key similarity factors have been used. They have built up a prototype clone genealogy (bunch) extractor, which is further connected on three open source ventures including the Linux kernel.

Study from Jens krinke et al. [8] has identified similar codes with fine-grained program dependence graphs and this approach works not only on the syntax of a program but also on the semantics. Prototype model is used with the non polynomial complexities which yields high precision and recall. This approach has not worked well with a polynomial time limit. Yaowen chen et.al. [9] presents an experimental study on code cloning in more than twenty open source games by applying a state of the art clone detector, NiCad. They also used VisCad tool for visualization and analysis of clones. This research shows that cloning happens not only at inter-project level, but also at an intra-project. On the basis of different dimensions, such as language category, clone density and the clone location, they analyzed a set of metrics and requirement of adopting clone management systems for game development.

Study from Robert Tairas, Jeffy Gray [10] shows the expanding clone up support by consolidating clone identification and rewriting an existing source to improve its readability, reusability (refactoring) activities simply by modifying the structure of the code yet without changing the conduct of code. They have proposed CeDAR (clone recognition, investigation and refactoring) code yet without changing the conduct of code. This tool focuses only on Type1 and Type 2 clones. The results of clone detection techniques and refactoring activities for eliminating duplicate code and for maintenance of code clones have been accumulated. Mark Gabel et al. [11] have performed scalable detection of clones on the basis of semantic clones. Millions of lines of code have been evaluated using their algorithm. The program dependence graphs (PDG) [12] problem which has been used to implement program slicing [13], have been reduced to

a simple tree similarity problem. Some of the productive clone recognition methods which are utilised to discover fundamentally comparative clones are DECKARD [14], CP-Miner [15], and CCFinder [16] .

Study from Madhulina Sarkar et al. [17] used clone detection technique to forecast the resource requirements, feedback guided by automatic job modelling methodology which has been founded on the metric based clone discovery [21]. When the job is entered in a system, its execution is bolstered and assets are included or evacuated on the premise of a versatile execution plan displayed in [18]. A tool called PRAGMA is used to implement this scheme. Dhavleesh Rattan et al. [19] have reviewed the programming clones. In their literature review, near about 100 studies from literatures were based on software clone detection. The result of these studies is also categorized as types of clones, internal representation of clones, semantic clones, model clones, code clone management, different approaches of clone detection. Study from Gehan M.K. Selim et al. [20] represents an enhancement in clone detection, which are based on source-based by using intermediate representation. They have used a hybrid approach for detection of Type 3 clones. In clone genealogies, their technique has higher accuracy on the correlation with standalone string based and token based clone detector.

4. TYPES OF CLONES

4.1. Type 1

These types of clones are also known as exact clones. In this type of clone, there is a little bit chance of variation in whitespaces and comments, but as the name suggests they are exact or identical clones.

```
int add(int num[],int x){
    int a=0;//add
    for(int m=0;m<x;m++){
    a=a+num[m];
    }
    return a;
    }
```

```
int add(int num[], int x){
int a=0;
for(int m=0;m<x;m++){
a=a+num[m];
}
return a;
}
```

```
int add(int num[],int x){
int a=0;//add
for(int m=0;m<x;m++){
a=a+num[m];
}
return a;
}
```

Figure 1: Exact clone

4.2. Type 2

These types of clones are known as renamed or parameterized clones. The structure or the syntax of this type of clone is same but there can be exceptions of layouts, variables, literals and in comments.

```
int add(int num[], int x){
    int a=0;//add
    for(int m=0;m<x;m++){
    a=a+num[m];
    }
    return a;
    }
```

```
Int doadd(int no[], int x){
int a=0;
for(int m=0;m<x;m++){
add=add+no[m];
}
return add;
}
```

```
int addition(int s[], int x){
int a=0;//add
for(int m=0;m<x;m++){
a=a+s[m];
}
return a;
}
```

Figure 2: Syntactic clone

4.3. Type 3

These types of clones are known as Near- Miss Clones. Some amendments are done in the code like adding or removing new statements, modification in layouts, modification in literals, changing the name of variables. If there is deletion of a statement in another code fragment, then they is termed as Near-Miss clone.

```
int addition (int
num[], int n){
 int sum=0;//sum
 for(int
i=0;i<n;i++){
 sum=sum+num[i];
 }
 return sum;
 }
```

```
int doadd(int
no[], int n){
int s=0;
for(int
i=0;i<n;i++){
s+=no[i];
}
return s;
}
```

```
int sum(int a[],int
n){
int p=0;//sum
for(int i=0;i<n;){
p=p+a[i];
i++;
}
return p;
}
```

Figure 3: Near-Miss clone

4.4. Type 4

These sorts of clones are known as semantic clones. If two code fragments have similarity in their function or their behaviour is similar, then they would be considered as semantic clones. Textual similarity is not the necessity. But it is not necessary in every case that code fragment is copied from the native code.

```
int add(int no[],int n){
int sum=0;
for(int p=0;p<n;p++){
sum=sum+no[i];
}
return sum;
}
```

```
int add(int no[],int n){
if(n==1)
return no[n-1];
else
return no[n-1]+add[no,n-1];
}
```

Figure 4: Semantic clone

5. REASONS OF CLONING

5.1. Reuse mechanism

One can reuse requirement, code, design, test case in any phase of the software development life cycle (SDLC).

5.2. To meet the deadline

Time constraint leads to the software or code cloning. Most of the programmers in companies do copy and paste or make certain amendments in code in order to meet deadline and to achieve desired functionality.

5.3. Lack of Interpretation of requirements

Here and there, it is hard to translate and make an orderly approach for every single prerequisite as a result of the high number of determinations in extensive frameworks.

5.4. Tested code

As there is always risk associated with new code because programmer can develop the code which might be more mind boggling or more inclined to bugs and errors. So to copy code is always preferable choice.

5.5. Matter of chance

By co-incidence, codes can be similar.

1) *Preferring Developer's Credibility:* In most of the company's developer's performance is measured by checking how much number of lines he is producing in one hour.
2) *Little knowledge of the new language:* Sometimes programmer does not have the better command over the programming language; at that moment they prefer copy and paste technique.

6. ADVANTAGES OF CLONING

6.1. Quick process

When a programmer starts a code from the scratch, it takes lots of time and effort. So, copy and paste mechanisms are easier to develop a system.

6.2. Foundation for templates

Template building is supported by code cloning. For example:- same types of design are followed in all pages of many websites.

6.3. Encouraging reuse

To achieve already existing Functionality of the tested code, reusing is done by copy and paste technique.

7. DISADVANTAGES OF CLONING

7.1. Rise in the need of resources

Program becomes bigger and complex with the cloning. The more number of hardware and software are needed to meet the requirements.

7.2. Likelihood of poor design increases

Modular and structural programming approach is not being followed. When a clone is used in the program, it leads to poor design and ultimately it hampers the quality.

7.3. Maintenance becomes a tedious task

To maintain the cloned code which complicates the understanding of the code, becomes a difficult work for the maintenance team.

7.4. Rise in cost and time

If bug is detected, then to remove it in the entire code takes a huge amount of time and effort as well as the cost increases for modification.

8. TECHNIQUES OF CLONE DETECTION

8.1. Text based clone detection technique

Detection is not performed on the basis of syntactic and semantic similarity. Line by line comparison will be done on the two code fragments. If textual similarity exists between them, then they are counted as clones.

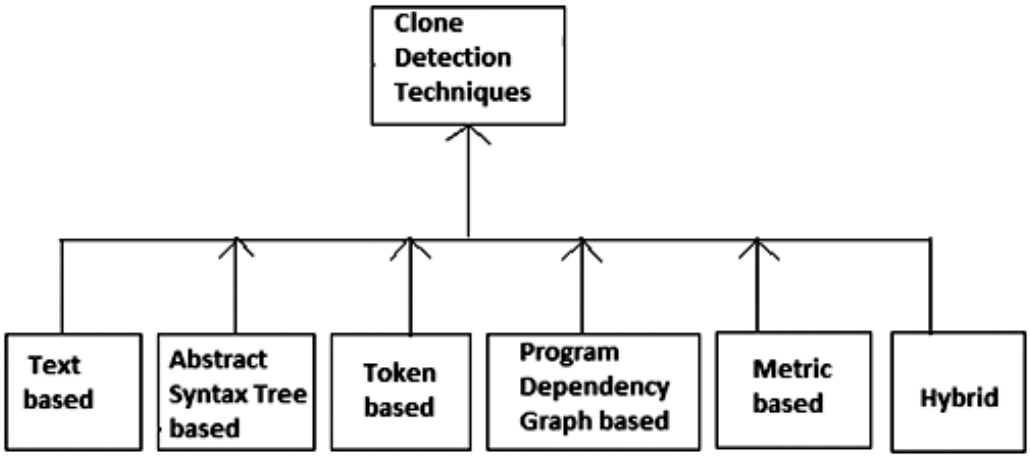

Figure 5: Clone Detection Techniques

8.2. Abstract-syntax tree based clone detection technique

Codes are parsed into a tree based algorithm [21] or tree based matching, if match is detected then the result would be a clone. Generally, near- miss clones are represented in the abstract syntax tree and then on the result, pattern matching is applied.

8.3. Token-based clone detection technique

By using the concept of lexical analysis or study, source code is converted into the tokens. Exact clones and syntactic clones are traced out with this technique.

8.4. Graph-based clone detection technique

From the source code, the program dependency graph is acquired which includes control flow and data flow. It contains behaviour or semantic information of a two codes.

8.5. Metric-based code clone detection technique

Distinctive measurements of codes are computed. Measurements contain data about the name of strategies, formats, literals and control of the project. The parts of code which will demonstrate comparable metric qualities are considered as clones.

8.6. Hybrid clone detection technique

By mixing and using two or more above mentioned techniques clones can be detected. This technique holds better value than normal technique. For example: graph and metrics technique can be used in a combination for best results.

9. CLONE DETECTION PROCESS

There are the generic steps involved in detecting clones whether they are actual clones or not. This process is quite expensive, requires fast computation speed. On the basis of similarity, clones are detected from the clone pairs.

9.1. Pre-processing

This phase follows two steps: one is dividing the source code into the sections also known as segmentation. Secondly, figure out the area of comparison. There are certain objectives of this phase:

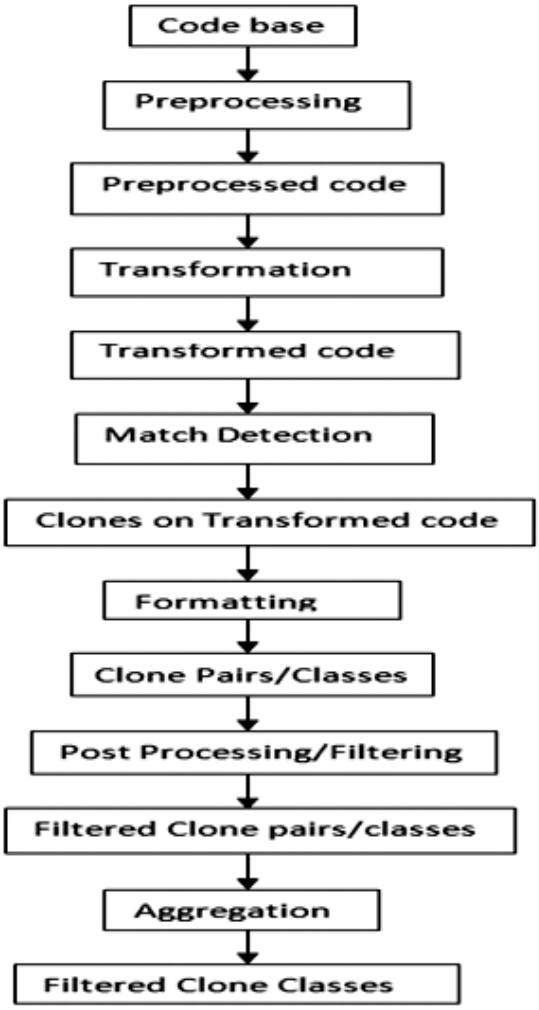

Figure 6: Clone Detection Process

1) *Elimination of unwanted parts:* Source code is segmented and uninterested parts are removed, which may generate false positive values. Reckoning of further steps would be easy.
2) *Figure out source units:* Once the removal of unwanted code is completed, then the rest of the source code is partitioned in such a way so that common portion can be obtained. For an instance: in a program; files, classes, functions/methods, start finish blocks, or source line sequence.
3) *Figure out comparison units:* Segmentation of the source units to further obtain smaller units for the comparison purpose.

9.2. Transformation

For the comparison purpose, the main motive of this phase is to convert the source code units into peculiar intermediate representations. This process is called as extraction. This step is further subdivided into following:-

1) *Extraction:* To make source code appropriate as input to the real algorithm, conversion of source code has done.
2) *Tokenization:* Every line of source code is isolated into tokens.
3) *Parsing:* To indicate the clones in syntactic approach, abstract syntax tree is used to compare algorithms for same sub-trees. Metric-based approach can also be used.

9.3. Match detection

Transformed code which is obtained from the above steps is put into comparison algorithm where all the transformed comparison units are evaluated on the basis of similarity to determine the matches. A set of candidate clone pairs will be obtained. The algorithms used in this phase are: suffix tree dynamic pattern matching and hash esteem examination.

9.4. Formatting

The clone pair list for the changed code acquired by the comparison algorithm is transformed over to a relating clone pair list for the original code base.

9.5. Post processing/filtering

This step is further subdivided into two parts:

1) *Manual analysis:* Here false positives are filtered out by human experts.
2) *Automated heuristic:* Few parameters are already set according to filtering purposes. For example: length, frequency, diversity etc.

9.6. Aggregation

With a specific end goal to expel the information, perform ensuing examination or accumulate outline measurements, clones might be collected into clone classes.

10. CONCLUSION AND FUTURE SCOPE

This paper puts a light on all the types of clones and various techniques for the detection of clones. We have also presented the reasons of cloning along with its pros and cons and the process involved in detection of clones. Since the last decade, there has been wonderful contribution of numerous researchers in the field of software cloning. This field has still a lot of scope for new researchers to work upon code clone genealogies, investigating potential clones from the actual clones, detecting type 4(Semantic) clones with more accuracy and precision, refactoring of clones and of course the maintenance of a project which is the most costly phase of SDLC.

REFERENCES

[1] I. D. Baxter, A. Yahin, L. Moura, M. S. Anna, L. Bier, and S. Drive, “Clone Detection Using Abstract Syntax Trees,” 1998.

[2] S. Ducasse, M. Rieger, and S. Demeyer, “A language independent approach for detecting duplicated code,” Proc. IEEE Int. Conf. Softw. Maint. - 1999 (ICSM’99). ’Software Maint. Bus. Chang. (Cat. No.99CB36360), no. c, pp. 109–118, 1999.

[3] R. Komondoor and S. Horwitz, “Using Slicing to Identify Duplication in Source Code,” SAS ’01 Proc. 8th Int. Symp. Static Anal., vol. 2126, pp. 40–56, 2001.

[4] J. Mayrand, C. Leblanc, and E. M. Merlo, “Experiment on the automatic detection of function clones in a\nsoftware system using metrics,” Softw. Maint. 1996, Proceedings., Int. Conf., pp. 244–253, 1996.

[5] C. K. Roy, J. R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Sci. Comput. Program., vol. 74, no. 7, pp. 470–495, 2009.

[6] C. K. Roy, “Detection and Analysis of Near-Miss Software Clones,” pp. 447–450, 2009.

[7] O. Lqj, O. Hqhdorjlhv, R. K. Saha, C. K. Roy, and K. A. Schneider, “An Automatic Framework for Extracting and Classifying Near-Miss Clone Genealogies” 2011, pp. 293–302.

[8] J. Krinke, “Identifying Similar Code with Program Dependence Graphs,” 2001.

[9] Y. Chen and C. K. Roy, “Near-miss Software Clones in Open Source Games/ : An Empirical Study,” pp. 1–7, 2014.

[10] R. Tairas and J. Gray, “Increasing clone maintenance support by unifying clone detection and refactoring activities,” Inf. Softw. Technol., vol. 54, no. 12, pp. 1297–1307, 2012.

[11] M. Gabel, “Scalable Detection of Semantic Clones,” pp. 321–330, 2008.

[12] C. Liu, C. Chen, J. Han, and P. S. Yu, “GPLAG: detection of software plagiarism by program dependence graph analysis,” Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discov. data Min., pp. 872–881, 2006.

[13] M. Weiser, “Program slicing,” Proc. 5th Int. Conf. Softw. Eng., pp. 439–449, 1981.

[14] L. Jiang, G. Misherghi, and Z. Su, “D ECKARD/ : Scalable and Accurate Tree-based Detection of Code Clones “,” no. 520320, 2007.

[15] S. Lu, Z. Li, F. Qin, L. Tan, P. Zhou, and Y. Zhou, “BugBench: Benchmarks for Evaluating Bug Detection Tools,” Proc Work. Eval. Softw. Defect Detect. Tools, no. 3, pp. 1–5, 2005.

[16] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multilinguistic token-based code clone detection system for large scale source code,” IEEE Trans. Softw. Eng., vol. 28, no. 7, pp. 654–670, 2002.

[17] M. Sarkar, T. Mondal, S. Roy, and N. Mukherjee, “Resource requirement prediction using clone detection technique,” Futur. Gener. Comput. Syst., vol. 29, no. 4, pp. 936–952, 2013.

[18] B. S. Baker, “A Program for Identifying Duplicated Code,” Comput. Sci. Stat., vol. 24, pp. 49–57, 1992.

[19] D. Rattan, R. Bhatia, and M. Singh, Software clone detection/ : A systematic review, vol. 55, no. 7. Elsevier B.V., 2013.

[20] G. M. K. Selim, K. C. Foo, and Y. Zou, “Enhancing source-based clone detection using intermediate representation,” Proc. - Work. Conf. Reverse Eng. WCRE, pp. 227–236, 2010.

[21] W. S. Evans and C. W. Fraser, “Clone Detection via Structural Abstraction,” Seminar, 2008.