International Journal of Technology and Science, ISSN (Online) 2350-1111, (Print) 2350-1103 Volume 3, Issue 3, 2016 pp. 22-26

Study on various Software clone types and Detection mechanism in forked software project

A. Prof. Sanjay B.Ankali[1], B. Dr. Latha Parthiban[2], and C. Prof. Bahubali M Akiwate[3]

[1] Asst. Professor, Department of CSE, KLECET, Chikodi, India
sanjay.ankali@yahoo.com

[2]Asst. Professor, Dept. of CS, Pondicherry University CC
lathaparthiban@yahoo.com

[3]Asst. Professor, Department of CSE, KLECET, Chikodi, India
bahubalimakiwate@gmail.com

Abstract:**Several studies on clones or porting show that about 14% to 21% of software systems can contain duplicated code, which are basically the results of copying existing code fragments and using them with or without minor modifications. One of the major drawback of such code fragments is that if a bug is detected in a fragment, all similar fragments should be investigated to check the possible existence of the same bug in the similar fragments. In this paper, we first describe the cloning terminologies and commonly used clone type. Second, we provide a review of the existing clone taxonomies, detection approaches. Finally, this paper concludes by pointing out several open problems related to clone detection research.**

Key terms: **Cloning, forked projects.**

I.INTRODUCTION

Copying code fragments from existing software and then pasting with or without minor modifications or adaptations are common activities in software development. One of the examples is split of FreeBSD and netBSD from 386BSD families. This type of reuse to existing code is called code cloning or forking. It is difficult to say which fragment is original and which is duplicate, fragments of code which are exactly same or similar to each other are called code clones. Several studies show that software systems with code clones are more difficult to maintain than the ones without them [1]. The cloning produces code that is difficult to maintain and introduce subtle errors [2]. Cloning in software development has several adverse effects on the maintenance life-cycles of software systems.

In most of the cases clones are the result of copy-paste activities. In software development this is common as it minimizes coding effort and time, especially in device drivers of Linux operating systems where the algorithms are similar. There are several other factors such as performance enhancement and coding style because of which large systems may contain a significant percentage of duplicated code. There is also accidental cloning, which is the result of using the same set of APIs to implement similar protocols [6]. Code cloning is found to be a more serious problem in software development [1]. Clones may not hamper normal functioning of the system, but they will have a negative impact on evolution [3]. Counter measures are needed to decrease the cost of development.

Code clones will directly affect the quality, maintainability and comprehensibility of software systems[21]. Clones are more error prone as same bug in code fragment should be inspected in all similar code patches. Too much cloning increases design problems such as missing inheritance or procedural abstraction. The costs of changes carried out after delivery is estimated at 40% - 70% of the total costs during a system's lifetime. Existing research shows that a significant amount of code in a software system is cloned code and this amount may vary depending on the domain and origin of the software system. For instance, Baker [1] has found that on large systems between 13% - 20% of source code can be cloned code. Lague et al. have studied only function clones and reported that between 6.4% - 7.5% of code is cloned code whereas Baxter et al. have reported that 12.7% of code being clones of a software system.

II.REASONS FOR CODE DUPLICATION

Several factors influence the developers and/or maintenance engineers in making cloned code in the system. Some clones are introduced by accidents. Below we provide the various factors for which clones can be introduced in the source code [1]

A. Development Strategy

Clone may be introduced in software systems due to the different reuse and programming approaches. Examples are:

B. Reusing

Reusing an entire system or logic, procedure/ function and design are the prime reasons of code clones.

C. Copy-Paste: Copying the existing code and pasting (with or without minor modifications) to form new system is the simplest form of reuse mechanism in the development process which results code duplication. Famous syntax and semantics can be repeated [4].

D. Logic and design reuse: For the similar product/solutions developers can use existing function and logic. For example Linux kernel device drivers contain large rates of duplication because all the drivers have the same interface and most of them implement a simple and similar logic. Moreover, the design of such systems does not allow for more sophisticated forms of reuse.

E. Generative programming approach**:** Generating code with a tool using generative programming may produce huge code clones because these tools soften use the same template to generate the similar logic.

III. TYPES OF CODE CLONES

Two code fragments can be similar based on the similarity of their program statements or in their functionalities. In this section, we consider clone types based on the types of similarity in code fragments. Based on the textual similarity we have following types of clones [7]:

A. Type I Clones

Type I is also called Exact clones, original code ca be copied as it is with some variations in whitespace new line(s), blanks, tabs, comments and/or layouts.

Let us consider the following code fragment,

```
if (a>=b) {// Comment1
M = a + b;
N = a-b;}
else
Z = a*b;      //Comment2
```

An exact copy clone of this original copy could be as follows:

```
if (a>=b) {
// Comment1
M=a+b;
N=a-b ;}
else          // Comment2
Z=a*b;
```

B. Type II Clones

Two code fragment that are similar except for some possible variations in names of user-defined identifiers (name of class, method and variables), types, layout and comments. below code is an example of Type II clones.

```
if(a1!=b1)
{ d1 = d1 + 1;}
  else
  c1 = d1 – a1; //Comment2
```

A *Type II* clone for this fragment can be as follows:

```
if (m1 >= n1) // Comment1
{ y1 = x1 + n1;
x1 = x1 + 5; }//Comment3
else
y1 = x1 – m1; //Comment2'
```

above two code segments have change in their shape, variable names and value assignments. However, the syntactic structure is similar in both segments.

C. Type III Clones

In *Type III* clones, the copied fragment is further modified by addition and deletion of statements. Consider the original code segment,

```
if (x >= y) {
m = d + y;
n = d + 1;
}else
m = d - x;
```

now we add alpha=1 to get following code

```
if (x>= y) {
m = d + y;
alpha = 1; // This statement is added
d = d + 1; }
else m = d - x;
```

Without this inserted statement, this copied fragment could be a *Type II* code clone.

D.Type IV Clones

These are the results of semantic similarity between multiple code fragments. Two code fragments may be developed by two different programmers to implement the same kind of logic making the code fragments similar in their functionality. Functional similarity reflects the degree to which the components act alike, i.e., captures similar functional properties and similarity assessment methods rely on matching of pre/post-conditions. Let us consider the following code fragment 1, where the final value of 'j' is the factorial value of the variable n.

Fragment 1:

```
inti, fact=1;
for (i=1; i<=n; i++)
fact= fact * i;
```

fragment1 iterative code can be implemented recursively as shown in fragment 2 which calculates the factorial of its argument *n.*

Fragment 2:

```
int factorial(int n) {
if (n == 0) return 1 ;
else return n * factorial(n-1) ;
}
```

The code having no lexical/syntactic/structural similarities between the statements of the two fragments are called *type IV clones.*

IV.CLONE DETECTION TOOLS AND TECHNIQUES

Below literature presents various clone detection techniques. Most of them are for research purposes aiming to help the software development and maintenance processes. Most of the tools also detect different types of clones primarily based on the detection techniques and comparison level of granularity. In this section, we provide the different clone detection techniques in the form of taxonomy.

A. Taxonomy of Detection Techniques

Each of the clone detection techniques consists of several properties by which that particular technique can be explained, for example, what it does, how it does etc.

Code Transformation: Instead of working directly on the fresh source code, this approach performs kind of transformation or filtering before applying the actual comparison. Filtering just remove whitespace or comments while other use transformation to get another form of code representation suitable for the comparison algorithm and for detecting target clone types for the reengineering purpose.

Granularity Comparison: Different algorithms work on different code representations on different levels of granularity. Some algorithm works on the granularity of one statement while others work on Abstract Syntax Tree.

Comparison Algorithm: In detecting clones of different types, the choice of the algorithm is also a major

concern. Some approaches use the sequence matching algorithm which is commonly applied in the biological science for DNA-sequence matching while others apply several data mining/information retrieval algorithms.
Computational Complexity**:** The efficiency of the clone detection technique is a major concern as the technique should be scaled up to detect clones in large software of millions of lines of code. The complexity of an approach depends on the type of transformations and the comparison algorithm used.
Language Independency: Now days the major concern for a clone detection tool is a software system can be developed with several languages. Also a language independent tool can be applied to any system of interest without any worries.

V. OPEN PROBLEMS IN CLONE DETECTION

There are several open problems concerning the various issues discussed in this paper. A list of 30 open questions on clone detection research is presented in table1. 1st shows issues, 2nd column shows method and 3rd shows status to solution.

*A. Types and Taxonomies of Clones :*There are several types and taxonomies unclear issues to be taken care of. In the following, we point out some of these:
Formal definition of clones: There is no proper definition to clone. All the available definitions are either vague to a certain extent or incomplete. A formal clone definition should be established overcoming the current limitations and associated vagueness. The definition should take into account different types of similarities such as representation similarity (textual, syntactic and structural), semantic or behavioral similarity, execution similarity, metrics similarity and feature-based similarity. While it is easy to define similarity for some types, it is still an open problem to define and measure semantic similarity.

Clone length: there are two ways to measure clone length either by the number of tokens or by number of lines such as number of Abstract Syntax Tree. Once the unit of measurement is chosen, it should be decided what could be the appropriate minimum threshold for clone length.

Clone types based on origins and or risks: There are still no categorizations of clones based on origins or risks. It is also not yet known the statistical distribution of such types in larger systems. Empirical studies can be undertaken for finding such distribution of clone types.

Removal, avoidance and risks: For each of the clone types, the risk and cost of removal and avoidance should be studied.

Relevance ranking: Once the clones are identified, a relevance ranking should be established for removal or other maintenance activities.

B. Better Clone Detection Techniques

There are several clone detection techniques available today; there is always a need to have a better one. The open issues concerning a new technique are:
Higher precision and recall: To date no clone detection technique is found sound enough in terms of precision, recall, robustness and scalability. Most tools show complementary behaviors for precision and recall. Scalability and robustness are also challenging in almost all cases. Therefore, there is a crucial need to develop a new clone detection technique that can overcome the existing limitations.
Detecting Type III Clones: No tool is reported to do well in detecting *Type III* clones. Moreover, attempts in detecting semantic clones (*Type IV*) are very few or none. Both *Type III* and *Type IV* clones are important from a maintenance point of view. When developing a new technique especial treatments should be considered for detecting *Type III* clones. Again, a classification or taxonomy of such clones is crucial.
Semantic clone detection tool: Semantic clone detection technique is used to find *Type IV* clones. Clone [16] definition and detection by semantic similarity are undecidable problems in general. Nevertheless, attempts can be made to detect semantic clones by applying extensive and intelligent normalizations to the code.

VI. APPLICATIONS FOR CLONE DETECTION RESEARCH

In addition to clone refactoring, avoidance and management, there are several other domains in which clone detection techniques seem helpful. There are other areas related to clone detection from which clone detection techniques themselves can get benefited. In this Section, we provide a list of applications and related works of clone detection research.

A. Plagiarism Detection

Plagiarism is one of the closely related areas of clone detection [11]. For plagiarism detection, copied code is disguised intentionally and therefore, it is more difficult to detect. Clone detection with the extensive normalization on source code can detect plagiarism. But such normalization may produce lots of false positives. Clone detection tools such as token-based *CCFinder* and metrics-based CLAN have been applied in Detecting plagiarism.[18] Unfortunately, to
date it is not clear how good they are in doing so. Clone detection techniques, on the other hand may benefit from plagiarism detection tools Burd and bailey gave 2 plagiarism detection and 3 clone detection tools, *JPlag*[12] and Moss [5]. From their study, it is found that plagiarism detection tools show more or less similar precision and recall compared to the clone detection tools even though these tools detect clones across files only. However, plagiarism detection tools are designed to measure the degree of similarity between a pair of programs. Clone detection tools work within the scope of intra and inter-file levels even with various clone granularities. If the plagiarism detection tools are directly used to find code clones within a single

program, they need to compare all possible pairs of code fragments. A system with *n* statements requires a total of *O*(*n*3) pair wise comparisons. This level of computational complexity is impractical for very large systems.

B. Origin Analysis
Clone detection can be applied to origin analysis in which two versions of a system are analyzed to gain the knowledge as how and why structural change has occurred. As of plagiarism detection, the critical difference between the scopes of detection approaches makes them infeasible to assist one another directly.

C. Merging
Merging works with two different variants of similar systems. Establishing the relation between the pair of systems, clones from both the systems are compared and analyzed. Again, the comparison is only between systems. Unlike clone detection, clone analysis within a system is irrelevant for merging.

Table 1: below table shows unsolved issues related to clones

Sl.no	Un solved issues related to clones	Classification
1	Possibility to detect clones in XML/HTML files	Detection
2	Do we have IDE specific clone detection tool?	Detection
3	Can we detect higher level clones?	Detection
4	How can we identify clone requirements from the code	Definition
5	Do we understand clone for other programming paradigm?	Detection
	What kinds of clones are in functional languages	
6	For what levels of code do definitions of clone exists	Definition
7	Are clones defined subjectively or objectively?	Definition
8	Are clones relations symmetric?	Definition
9	Size of the system affect rate of occurrences of clones?	Definition
10	What are clone types	Definition
11	Do we have standard difference measure between similar code	Definitions
12	fragments?	Detection/ Taxonomy
13	Do we have framework to classify & evaluate clones?	
14	Do we understand other level clones?	Taxonomy
15	Can we describe the difference between similar code fragments?	Taxonomy
16	Do we understand context sensitive clones?	Taxonomy
17	What kinds of clones are good and bad	Taxonomy
18	Do we know detection of what kind of clones are too complex to	Ranking
	beneficial?	Ranking
19	Can we deal with generated code?	Clone based actions
20	Can we measure clone rates in system?	
21	What do we do with clones	Clone based actions
22	Must clones be removed?	
23	Which clone must be refactored, kept or encouraged?	Clone based actions

However, both merging and clone detection require robust comparison techniques and each of them can be benefited from other by sharing their comparison algorithms and analysis approaches.

D. Software Evolution : Software evolution analysis is closely related problem to origin analysis and merging. As of origin analysis, two or more different versions of a software systems are mapped for finding a relation between them with a view to observe their evolution behavior. By detecting clones from each of the versions and then mapping the similar clone groups such a relation can be established [2].

E. Multi-version Program Analysis
Multi-version program analysis is the process of mapping element of one program version to the elements of other version of that program. For such mapping, a matching between the program elements of the two versions should be established. Clone detection techniques can be used for establishing such matching relation.

VII. RELATED WORK

Juergens et al. [9] detect inconsistent clones using a suffix-tree based, lexical clone detection algorithm. Their interviews with developers confirm that inconsistencies in the found clones are indeed bugs and report that "nearly every second, unintentional inconsistent changes to clones lead to a fault." Chou et al. show that porting is an important source of

bugs in operating systems [4]. In 65% of the ported code, at least one
identifier is renamed, and in 27% cases at least one statement is inserted, modified, or deleted [14]. An incorrect adaptation of ported code often leads to porting errors [8]. This observation is aligned with our findings—where we find 113 and 182 porting errors by mining FreeBSD and Linux version histories respectively.
SPA [16] detects a broader scope of inconsistent renaming by tokenizing function names, file names, and identifier names using a camel case naming convention and mapping corresponding tokens. Our algorithm detects an inconsistency when a token in one context maps to multiple tokens in the other context. For example, when code is ported from Export.java SPA checks whether all names related to export are updated to import.

CONCLUSION

Clone detection is an active research in Software Engineering and Artificial Intelligence the literature provides plenty of work in detecting and removing clones from software systems. Research is also done in maintaining clones in software systems in its evolution life cycle. In this paper, a survey on the area of software clone detection research is made putting emphasis which shows the definitions of clones.

This paper may also assist in identifying remaining open research questions, possible area for future research, and choose combinations of existing techniques.

REFERENCES

[1] Brenda Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), pp. 86-95, Toronto, Ontario, Canada, July 1995.

[2] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. R. Engler.An empirical study of operating system errors. In Proceedings of the 18th ACM symposium on Operating systems principles (SOSP'01), pp. 7388, Banff, Alberta, Canada, October 2001.

[3] Reto Geiger, Beat Fluri, Harald C. Gall and Martin Pinzger. Relation of code clones and change couplings. In Proceedings of the 9th International Conference of Funtamental Approaches to Software Engineering (FASE'06), pp. 411-425, Vienna, Austria, March 2006.

[4] Miryung Kim, Lawrence Bergman, Tessa Lau, David Notkin. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In Proceedings of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE'04), pp. 83- 92, Redondo Beach, CA, USA, August 2004.

[5] Cory Kapser and Michael W. Godfrey. \clones considered harmful" considered harmful. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06), pp. 19-28, Benevento, Italy, October 2006.

[6] Raihan Al-Ekram, Cory Kapser, Michael Godfrey. Cloning by Accident: An Empirical Study of Source Code Cloning Across Software Systems. International Symposium on Empirical Software Engineering (ISESE'05), pp. 376-385, Noosa Heads, Australia, November 2005.

[7] Stefan Bellon. Detection of Software Clones Tool Comparison Experiment. Tool Comparison Experiment presented at the 1st IEEE International Workshop on Source Code Analysis and Manipulation, Montreal, Canada, October 2002.

[8] Rainer Koschke. Survey of Research on Software Clones. In Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 24pp.,Dagstuhl,Germany, July 2006.

[9] Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In Proceedings of the 31st International Conference on Software Engineering, ICSE '09, pages 485–495, Washington, DC,USA, 2009. IEEE Computer Society

[10] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2000.

[11] Lerina Aversano, Luigi Cerulo, and Massimiliano Di Penta. How Clones are Maintained: An Empirical Study. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07), pp. 81-90, Amsterdam, the Netherlands, March 2007.

[12] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. Finding plagiarisms among a set of programs with JPlag. In Journal of Universal Computer Science, 8(11):10161038, November 2002.

[13] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. In IEEE Transactions on Software Engineering, Vol. 32(3): 176-192, March 2006.

[14] Andy Kellens, Kim Mens, and Paolo Tonella.A Survey of Automated Code-Level Aspect Mining Techniques. In Transactions on Aspect Oriented Software Development, Vol. 4 (LNCS 4640), pp. 145-164, 2007.

[15] Magiel Bruntink. Aspect Mining using Clone Class Metrics. In Proceedings of the 1st Workshop on Aspect Reverse Engineering, 2004.

[16] Baishakhi Ray, Miryung Kim Detecting and Characterizing Semantic Inconsistencies in Ported Code

[17] Jeffrey Svajlenko Chanchal K. Roy Slawomir Duszynski ForkSim: Generating Software Forks for Evaluating Cross-Project Similarity Analysis Tools978-1-4673-5739-5/13/$31.00 c 2013 IEEE.

[18] Rainer Koschke University of Bremen Germany Frontiers of Software Clone Management ."

[19] Yael Dubinsky_, Julia Rubin_y, Thorsten Bergerzx, SlawomirDuszynski{, Martin Becker{ and Krzyszt of Czarnecki "An Exploratory Study of Cloning in Industrial Software Product LinesYael"

[20] Xiaoyin Wang1, Yingnong Dang2, Lu Zhang1, Dongmei Zhang2, Erica Lan3, Hong Mei"Can I Clone This Piece of Code Here?"ASE '12, September 3 – 7, 2012, ssen, Germany Copyright 2012 ACM 978-1-4503-1204-2/12/09 ...$10.00.

[21] Cory J. Kapser · Michael W. Godfrey "Cloning considered harmful" considered harmful: patterns of cloning in software springer DOI 10.1007/s10664-008-9076-6