12th International Conference on Recent Development in Engineering Science, Humanities and Management
The International Centre Goa, Panjim, Goa (India)
14th April 2019 www.conferenceworld.in ISBN: 978-93-87793-85-9

SOFTWARE CODE CLONES- A SURVEY

Sandeep Bal[#1], Sumesh Sood[*2]
[#]Research Scholar, Dept of RIC, IKG-PTU, Kapurthala, Punjab, India
**Asst. Prof., Department of Computer Applications,*
IKG-PTU Campus, Dinanagar, Punjab, India

Abstract: *The field of Code Clone detection in software research area is prevailing and influential problem which must be dealt very carefully for survival of software. Many researchers have defined clones differently depending upon their usage and threshold used. The definition of suitable output has not been given precisely. There is a crucial need to have detailed information about the types and definitions of what exactly a clone means, and the purpose of its detection. In this paper, we provide the different definitions and types of clones from the literature.*

Keywords: code clone, code fragment, exact clone, renamed clone, near-miss clone, semantic clone.

1. DEFINITIONS OF CODE CLONES IN LITERATURE

When a code is copied from another code portion from same or different source files, then they are termed as code clone. Code cloning is a form of software reuse, and exists in almost every software project. The developers can use these code clones directly or modify them by editing.

A definition by Baxter et al. [1] says that code clones are the segments of code that are similar according to some definition of similarity. Kamiya et al. [2] have defined clones as the portions of source file(s) that are “identical" or “similar" to each other where “identical" means “exact copy clones", but there is no formal definition for the term “similar".

These ambiguous terms lead to the formation of taxonomies for defining clones and their categories. For instance, an ordinal scale of eight different types of clones has been provided by Mayrand et al. [3] giving simple, crisp definitions. For example, the category “DistinctName" refers to the clones where only identifiers names can be differed between the cloned segments. However, their ordinal scale is not sufficient towards a sound definition of
clone. For instance, they define a category “SimilarExpression" to identify clones with expressions that differ but yet are still “similar".

Another way of defining clones is on the basis of clone size. Some studies show that a threshold of 30 tokens is reasonable as the minimum clone size for a token-based technique. Other studies argue that measuring clone size with respect to the number of lines could be a better option. However, there is also disagreement among people on the minimum clone size with respect to number of lines. For instance, in Bellon's tool comparison experiment [4, 5] the minimum clone size was set to 6 unprocessed lines of code. On the other hand, Baker [6] has set the minimum threshold to 15 non-commented lines while Johnson [7] set it to 50 lines. Some studies [8, 9] consider the no. of AST/PDG nodes as the measure of clone size and provide a measure of thresholds. Some studies [3, 10] work with only function clones and limit their clone size to the function body of any size.

Another problematic area in defining code clones is Language-specific semantic issues. For example, whether there is any relation between the Java *clone( )* method and the actual code clones. Java *clone( )* method is used for *object* duplication and treated as a reuse mechanism. Here the concentration is on code duplication and thus, even if the Java *clone( )* method copies an *object*(actually `by reference'), there is very little chance that the original *object* (e.g., a class) and the code that uses the *clone( )* method would be similar in their text. Thus, it is assumed that they should be considered clones. However, it is still questionable whether they are semantically similar. As both the original object and code segment using the *clone( )* method are similar in their functionalities, so, they should be considered as clones when considering semantic similarity. Similar other such situations may arise depending on the programming language of interest.

Therefore, the detection approach helps in defining clone and its minimum size but with an additional burden of associated vagueness. Giesecke [11] attempted to define a detection independent definition of exact and near- miss clones with following desirable properties of a clone definition:

- Independent of a Programming Language: Rather than finding code clones which are based on the text, syntax and structure of a particular programming language, logic clones must be found i.e., duplication of logic must be detected, the essential property of a program. If such a generic modelling of code clones can be determined, code clone detection problem will be independent of programming languages and the most of the limitations of language-based approaches can be overcome.
- Independent of a Detection Approach: The detection of a particular type of clones should be independent of the detection approaches. A developer can identify whether two code fragments form a clone or not. The detection approach should perform in such a way that it can replicate the detection capability of the human arbiter in algorithmic form.
- Describe a Continuum of Clones from Exact to Non-Exact: Once a code fragment is copied, it can be used without being changed or there might be different levels of editing in order to fit the programmer's need. As a results of extensive editing two fragments may be completely different. But even for fragments where the common origin is almost unrecognizable, similarity knowledge is still valuable for a range of maintenance tasks.

2. TYPES OF CLONES

The similarity in code fragments can be based on the similarity of their program text or similar in their functionalities without being textually similar. The direct copying & pasting a code fragment from one location to another location result in similar code fragments based on the similarity of program text i.e. Textual Similarity. Following are the types of clones under this category [5, 12, 13]:

- Type I: Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

```
If (x<=y) {
z= x + y; // Comment1
p= p + q; }
else
z= p-x; // Comment2
```

```
If (x<=y) {
// Comment1
        z= x + y;
p= p + q; }
else // Comment2
z= p-x;
```

Figure 1: Exact clones

- Type II: Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.

```
if (a <= b) {
z = x – y; //
Comment1
25
c = c + 1; }
else
c = x + b ;
// Comment2
```

```
if (p <= q)
{ // Comment1
s= t - k;
w = w + 5; //Comment
3 }
else
w = t + p;
//Comment2
```

Figure 2: Renamed clones

- Type III: Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments.

```
if (w <= x) {
y = w * 2; // Comment1
z = y - u;
p=p+1 //statement is deleted in
clone
}
Else
y= w * 2; //Comment2
```

```
if (w<=x)

{

// Comment1

y= w* 2;

z=y- u;

}

else // Comment2
```

Figure 3: Near Miss clones

The second category of Functional Similarity exists when the functionalities of the two code fragments are identical or similar i.e., they have similar pre and post conditions, they are known as semantic clones [9, 14, 15, 16] and referred as Type IV clones.

- Type IV: Two or more code fragments that perform the same computation but implemented through different syntactic variants.

```
int i, j=1;

for ( i = 1; i <= VALUE;  i++)

j = j * i ;
```

```
int factorial ( int n)  {

if (n == 0)  return 1 ;

else return n * factorial ( n - 1)  }
```

Figure 4: Semantic clones

3. CODE CLONE TERMS

Different terms are used in literature when referring to clones and their relation. These terms can be related to the four commonly used clone types mentioned in above section i.e. Type I, II, III and Type IV.

3.1 Exact Clones

Two or more code fragments are called exact clones if they are identical to each other with some differences in comments and whitespace or layout. The differences could be like changing the comments, restructuring in layout i.e., changing the positions of language elements through adding/removing tabs, blanks, and new lines. Exact clones are a part of Type I clones.

3.2 Renamed Clones

When renaming is performed for identifier names or literals values in a copied code fragment, it is called as Renamed clone which is also a Type II clone. A renamed clone does not always require consistent renaming.

3.3 Parameterized Clones

A parameterized clone or p-match clone is a renamed clone with systematic renaming. The clone detector looks for consistent name matching rather than normalizing all identifiers and/or literals to a special symbol. Parameterized clones are a subset of Type II clones.

3.4 Near-Miss Clones

All parameterized and renamed clones can also be called as Near-miss clones as the copied fragments are very similar to the original due to the editing activities such as changing in comments, layouts, changing the position of the source code elements through blanks and new lines, changing the identifiers, literals, macros may have been applied in such clones . A copied fragment which is not exact copy of the original due to slight changes but the syntactical structure is still the same as the original. However, many of the authors also assume that a slight modification within a statement(s) or even addition and

deletion of statement(s) in the copied fragment may not make the copied fragment different from the original and hence can be treated as near-miss clones. In this sense, Type III clones may also be termed as near-miss clones.

3.5 Gapped Clones

A gap clone code is partly similar to the original segment. In this type of clones, there is some different code portion between the segments. This different code portion is known as a gap [17] . These gaps may exist in different forms such as

- No renaming/renaming and code insertion: Identifiers can be renamed, values can be changed and some new statements can be inserted into the copied fragment but syntactic structure is same for the existing statements.
- No renaming/renaming and code deletion: Identifiers can be renamed, values can be changed and some statements can be deleted from the copied fragment but syntactic structure is same for the existing statements.
- No renaming/renaming and code modification: Identifiers can be renamed, values can be changed and some statements can be modified in the copied fragment. As statement(s) is modified there might be different syntactic structure between the two code segments.

3.6 Structural Clones

Structural clones are simple clones within a syntactic boundary following syntactic structure of a particular language. Structural similarity refers to the different software where coding exists with similar design structures. The syntactic boundaries can be function boundary, statement boundary, class boundary etc. depending on the programming language of interest.

Structural clones can be based with any level of similarity, i.e., a structural clone can be an exact clone, parameterized clone, renamed clones, gapped clones and so on. Structural clones focus on finding similar design structures after identifying the basic similarities like textual, lexical, syntactical and/or semantical similarities. While the types of clones are based on the level of similarity between the code fragments, structural clones are based on the level of clone granularity where the granularity can be any syntactic boundary (e.g., *begin-end* block) of the language. A structural clone can be any of the four types (Type I, II, III, IV) of clones based on its similarity level.

3.7 Function Clones

Function clones are simply clones that are restricted to refer to entire functions or procedures. Function clones are therefore, a subset of structural clones. So, a function clone can also be any of the four types (Type I, II, III, IV) of clones based on its similarity level.

3.8 Non-contiguous Clones

Non-contiguous clones are kind of near-duplication where gaps are allowed between the code fragments and therefore, non-contiguous clones are basically gapped clones. All the editing activities that allowed for gapped clones are also allowed for non-contiguous clones. Like gapped clones, non-contiguous clones are Type III clones.

3.9 Reordered Clones

When the reordering of statements and renaming of variables does not affect the functionalities of two fragments they are called as Reordered clones. On the basis of semantic similarity, reordered clones are said to be of Type IV clones. However, considering the lexical similarity, there are essentially gaps in the copied fragments and reordered clones can be a sort of gapped clones and therefore, are classified as Type III clones.

3.10 Design Level Structural Clones

When the clones are related to the design of the software system, these clones are called as design level structural clones. For example, two different clone sets that often occur together in program files are examples of structural clones. Simple clones are found first and then an item set data mining algorithm is applied to correlate simple clones for finding design level similarities.

3.11 Ubiquitous clones

The term ubiquitous clones [18] is used to refer to the short clones that are present in the multiple source files in an application i.e., short clones with high frequency across files of a system are called ubiquitous clones. These clones are usually short methods that perform a specific task, such as returning a new rectangle object, drawing two ovals (including setting their colors) and setting the undo and redo flags for the drawing views.

3.12 Volatile Clones

Sometimes maintenance activities performed during evolution of the software system helps in removing certain clones. These clones are called volatile clones. Any clones of Type I to Type IV could be under the classification of volatile clones [18].

3.13 Long-lived Clones

While volatile clones may disappear in maintenance activities (e.g., refactoring), there might be several clones which cannot be re-factored and hence remain in all versions of the systems. These clones are called long-lived clones [18].

3.14 Problematic Clones

Some clones when detected by the detection methods are meaningless w.r.t. the maintenance activity and therefore, must be filtered out. Following are the examples of such clones.

3.15 Spurious Clones

Sometimes a particular detection method may detect clones that are not really clones in the maintenance perspective. Clones with the ending lines of one function and beginning lines of another are not useful to a maintenance programmer even though from a lexical point of view these are in fact rightful clones. There may be several such clones that do not follow a syntactic structure and hence become useless from the maintenance point of view and known as spurious clones [13].

3.16 Frequently false positive clones

There are several clones that are not interesting or considered as false positives in the analysis process. Following [19] are the examples:

- Consecutive simple method declarations are found as code clones coincidentally.
- Consecutive method invocations are detected as code clones.
- Consecutive *if-statements* and *if-else statements* are detected as code clones.
- Consecutive case entries are found as code clones coincidentally.
- Consecutive variable declarations are found as code clones coincidentally.

4. SUMMARY

There are several types of clones used in the literature. They have been defined according to the kind of editing activities taking place while copying code. They have been generalised amongst terms according to their working. These clones have been affecting code at method level, class level etc. Their comparison will provide insight for code re-factoring as future activity in order to sustain software architecture.

REFERENCES

[1] Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna. Clone Detection Using Abstract Syntax Trees. In *Proceedings of the 14th International Conference on Software Maintenance (ICSM'98)*, pp. 368-377, Bethesda, Maryland, November 1998.

[2] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder: A MultilinguisticToken-Based Code Clone Detection System for Large Scale Source Code. *Transactions on Software Engineering*, Vol. 28(7): 654- 670, July 2002.

[3] Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In *Proceedings of the 12th International Conference on Software Maintenance (ICSM'96)*, pp. 244-253, Monterey, CA, USA, November 1996.

[4] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. In *IEEE Transactions on Software Engineering*, Vol. 33(9): 577-591, September 2007.

[5] 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.

[6] 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.

[7] John Johnson. Substring Matching for Clone Detection and Change Tracking. In *Proceedings of the 10th International Conference on Software Maintenance*, pp. 120-126, Victoria, British Columbia, Canada, September 1994.

[8] 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.

[9] Raghavan Komondoor and Susan Horwitz. Effective, Automatic Procedure Extraction. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03)*, pp. 33-42, Portland, Oregon, USA, May 2003.

[10] Bruno LaguÄe, Daniel Proulx, Jean Mayrand, Ettore M. Merlo and John Hudepohl. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In *Proceedings of the 13th International Conference on Software Maintenance (ICSM'97)*, pp. 314-321, Bari, Italy, October 1997.

[11] Simon Giesecke. Generic modelling of code clones. In *Proceedings of Duplication, Redundancy, and Similarity in Software*, ISSN 16824405, Dagstuhl, Germany, July 2006.

[12] Stefan Bellon.Vergleich von techniken zur erkennung duplizierten quellcodes. Diploma Thesis, No. 1998, University of Stuttgart (Germany), Institute for Software Technology, September 2002.

[13] Rainer Koschke, Raimar Falke and Pierre Frenzel. Clone Detection Using Abstract Syntax Suffix Trees. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06)*, pp. 253-262, Benevento, Italy, October 2006.

[14] Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01)*, pp. 301-309, Stuttgart, Germany, October 2001.

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

[16] Neil Davey, Paul Barson, Simon Field, Ray J Frank. The Development of a Software Clone Detector. *International Journal of Applied Software Technology*, Vol. 1(3/4):219-236, 1995.

[17] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. On detection of gapped code clones using gap locations. In *Proceedings 9th Asia-Pacific Software Engineering Conference (APSEC'02)*, pp. 327336, Gold Coast, Queensland, Australia, December 2002.

[18] Miryung Kim, Gail Murphy. An Empirical Study of Code Clone Genealogies. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*

(ESEC/SIGSOFT FSE 2005 '05), pp. 187-196, Lisbon, Portugal, September 2005.

[19] Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. Code Clone Analysis Methods for Efficient Software Maintenance. Graduate School of Information Science and Technology, Osaka University, 2006 (published as a report (PhD Thesis?) and paper version unpublished).