Clones: A Survey

[1]Sonam Gupta, [2]Dr. P. C Gupta
[1]Ajay Kumar Garg Engineering College, Ghaziabad, India
[2]Dept. of CSE, Jaipur National University, Jaipur, Rajasthan, India

Abstract
Various types of clones are available in a code which ultimately leads to a redundant code. But every redundant code is not a clone. It can be thought of as a duplicate code which can be either duplicate or somewhat changed as duplicate. In this paper we have described various types of clones which include both similar and identical clones and their classifications in further sub categories.

Keywords
Clones, Textual Similarity, Semantic Similarity, Structural Similarity

I. Introduction

Redundancies if exists in the database then that can be minimized by implanting the concept of normalization but it may be harmful if it exists in the coding of the software [3, 5, 18]. Every type of redundancy or cloning is not harmful [22]. In this work an empirical approach have been implemented to evaluate the impact of clones on the bugs. Many researches show that cloned code does not introduce more defects. Larger clone groups with less defects are known as ubiquitous clones [5-6]. In this paper an attempt has been made to identify the clone types known so far.

II. Clone Types

Program text-clones can be compared on the basis of the program text that has been copied. The text can be copied either line by line known as textual similarity or may have same pre and post conditions also known as semantic similarity. Based on these criteria clones are categorized as follows:

A. Textual Similarity

These are specified as follows and are also termed collectively as function clones.[2]
Type-I: These are an exact copy of the code just the variation in white spaces, comments and layouts are acceptable.[4]

```
e.g   for(int i=0;i<5;i++) // for loop
      {
          cout<<i;
            }// end of for
              Fragment 1

          for(int i=0;i<5;i++)
      {   // for loop
                    cout<<i; }
            // end of for
                  Fragment 2
```

It can be clearly seen from fragment 1 and 2 that there is change in the position of comment and parenthesis but still both the fragments are clones[5, 8, 13, 19]. If we change the identifier name then it will be different type of clone. Type-I is also termed as Exact Clones.
Type-II: This is similar to Type-I but unlike Type-I it supports changes in variable names, their types and function identifiers but the reserved words should not be changed.

```
e.g   for(int i=0;i<5;i++) // for loop
          {
             cout<<i;
           }// end of for
               Fragment 1

           for(int j=0;j<5;j++) // for loop
           {
            cout<<j;
            }// end of for

                        Fragment 2
```

Example-1
Type-II clone: Change in variable name

```
for(int i=0;i<5;i++) // for loop
           {
            cout<<i;
           }// end of for
               Fragment 1

           for(float j=0;j<5.5;j++) // for loop
       {
                    cout<<j;
             }// end of for
                    Fragment 2
```

Example-2
Type-II clone :Change in type of variable name

```
 void sum()
{
   int c= a+b;
   cout<<c;
}
                Fragment 1
int sub()
{
   int c= a-b;
   return(c);
}
              Fragment 2
```

Example-3
Type-II clone :Change in type of method and its name

```
void sum()
{
   int c= a+b;
   cout<<c;
}
                Fragment 1
```

```
void sub()
{
   int c= a-b;
   cout<<c;
}
```

Fragment 2

Example-4
Not a Type-II clone since return type of function is changed

Following are the clone terms which include Type-II clones:

1. Renamed Clones

These include the clones in which there is change in the identifier's name . It can be well understood by example-1,2,3 and 4. in a way Type-II clones are renamed clones.

2. Parameterized Clones

They are also known as p-match clones. They include consistent change in the identifier's name.[12]

```
e.g  for(int i=0,j=0;i<5,j<4;i++,j++)
       {
           cout<<i;
       }
```

Fragment 1

```
          for(int x=0,y=0;x<5,y<4;x++,y++)
       {
           cout<<x;
       }
```

Fragment 2

```
      for(int y=0,x=0;y<5,x<4;y++,x++)
       {
           cout<<y;
       }
```

Fragment 3

In the above e.g i and j has been changed to x and y so fragment 1 is p-match of fragment of 2 but fragment 2 is not p-match of fragment 3 since the variables are not changed consistently but fragment 2 is a renamed clone of fragment 3. So we can say that all parameterized clones are also renamed clones but not vice-versa.

3. Reordered clones

In this type of clones the change in identifier's name is acceptable along with it the sequence of statements can also be changed but that should not alter the data or control dependencies.[20]

```
e.g    for(int i=0,j=0;i<5,j<5;i++,j++)
       {
                  if(i%2==0)
                    i++;
        if(j%2==0)
                    j++;
              i=i+j;
                  cout<<i;
       }
```

Fragment-1

```
       for(int i=0,j=0;i<5,j<5;i++,j++)
       {
                  if(j%2==0)
                    j++;
        if(i%2==0)
                    i++;
             i=i+j;
                  cout<<i;
       }
```

Fragment-2

```
       for(int i=0,j=0;i<5,j<5;i++,j++)
       {
                  i=i+j;
                  cout<<i;
if(j%2==0)
                    j++;
        if(i%2==0)
                    i++;
       }
```

Fragment-3

In the above e.g fragment 1 and 2 they are reordered clones because they are not altering the data, but they are not clone with fragment 3 since there is change in value of i.
Type-III : This includes the features of Type-II along with this it can also add or remove the statements. The clone terms used with Type-III clones are as follows:

Near-miss clones

In this type of clone the syntactic structure of the code is same but slight modifications has been done by changing the name of the identifiers, comments layouts etc. So they include all the types of Type-II clones but in recent researches near miss clones also include addition and deletion of statements so they include Type-III clones.

```
e.g  for(int i=0;i<5;i++) //for loop
     {
                  cout<<i; //end of for
          }
```

Fragment-1

```
    for(int j=0;j<5;j++) //for loop
     {
                  if(j%2==0)
    cout<<j;
          }
```

Fragment-2

As shown in the above fragment 2 is a clone of fragment 1 in which the name of identifier is changed as well the statement is inserted, such type of clones are classified as Type-III clones.
Gapped Clones: In this type of clones either the statements are inserted or deleted but no change takes place in the name of the identifiers. [16, 21] So they follow some features of type-III clones.

Gapped clones are also known as Non-Contiguous Clones.

```
e.g  for(int i=0;i<5;i++)
     {
          if(i%2==0)
            cout<<i;
     }
Fragment-1

for(int i=0;i<5;i++)
     {
          i=i*2;  //inserted line
                if(i%2==0)
             cout<<i;
     }
Fragment-2

for(int i=0;i<5;i++)
     {
          cout<<i;
     }
Fragment-3
```

As can be seen from above e.g fragment 1, 2 and 3 are clones of each other in which a line is inserted in fragment 2 and a line is deleted in fragment 3.

Semantic Similarity

It includes the clones which have same pre and post conditions, i.e, the code performs the same computation but through different syntax.[10-12] The complexity of identifying such clones is highest of all the clones known so far. This is also known as Type-IV clones. For e.g to find a prime number whatever different logics are applied but still the conditions will be same i.e the logic will be different but computations will be similar.

The clone term associated with this type of clone is Intertwined Clone in which the code clones are combined to form a single code .

```
e.g  if(x>y)
        cout<<x;
Fragment-1

       if(y>x)
         cout<<y;
Fragment-2


        if(x>y)
         cout<<x;
       else if(y>x)
         cout<<y;
Fragment-3
```

So fragment 1 and 2 are combined into a single code in fragment 3.

3. Structural Similarity

This type of similarity is based on the software specification structures made. So, the software which have the same specifications will also have same design structures[1,17]. This is also known as Design-level similarity. Function clones are also subset of structural similarity. They include all the types of textual similarity under it.

Example includes structure of compound statements, structure of methods, range of global declarations, range of declaration in blocks etc.

III. Conclusion

In this paper, a review about the clone has been shown. This covers clones, its types, its advantages in coding as well as its drawbacks.

References

[1] W. Yang,"Identifying Syntactic Differences Between Two Programs", Soft.-Prac. and Exper., 21(7). pp. 739-755, 1991.

[2] J. Johnson,"Visualizing Textual Redundancy in Legacy Source", In CASCON, pp. 171-183, 1994.

[3] J. Johnson,"Visualizing Textual Redundancy in Legacy Source", In CASCON, pp. 171-183, 1994.

[4] B. Baker,"On Finding Duplication and Near-Duplication in Large Software Systems", In WCRE, pp. 86-95, 1995.

[5] J. Mayrand, C. Leblanc, E. Merlo,"Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", In ICSM, pp. 244-253, 1996.

[6] J. Mayrand, C. Leblanc, E. Merlo,"Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", In ICSM, pp. 244-253, 1996.

[7] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, L. Bier, "Clone Detection Using Abstract Syntax Trees", Proceedings IEEE International Conference on Software Maintenance, 1998, pp. 368-377.

[8] I. Baxter, A. Yahin, L. Moura, M. Anna,"Clone Detection Using Abstract Syntax Trees. In ICSM, pp. 368-377, 1998.

[9] S. Ducasse, M. Rieger, S. Demeyer,"A Language Independent Approach for Detecting Duplicated Code", In ICSM, pp. 109-118, 1999.

[10] R. Komondoor, S. Horwitz,"Using Slicing to Identify Duplication in Source Code", In SAS, pp. 40-56, 2001.

[11] J. Krinke,"Identifying Similar Code with Program Dependence Graphs", In WCRE, pp. 301-309, 2001

[12] G. Casazza, G. Antoniol, U. Villano, E. Merlo, M. Di Penta,"Identifying clones in the Linux Kernel", In Workshop on Source Code Analysis and Manipulation, pp. 90–97, 2001.

[13] V. Wahler, D. Seipel, J. Gudenberg, G. Fischer,"Clone Detection in Source Code by Frequent Itemset Techniques", In SCAM, pp. 128-135, 2004.

[14] C. Kapser, M. W. Godfrey,"Aiding comprehension of cloning through categorization", In 7th International Workshop on Principles of Software Evolution (IWPSE 2004), 6-7 September 2004, Kyoto, Japan, pp. 85–94, 2004.

[15] Z. Li, S. Lu, S. Myagmar, Y. Zhou,"CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code", IEEE TSE, 32(3), pp. 176-192, 2006.

[16] C. Liu, C. Chen, J. Han, P. Yu,"GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis", In KDD, pp. 872-881, 2006.

[17] R. Koschke, R. Falke, P. Frenzel,"Clone Detection Using Abstract Syntax Suffix Trees", In WCRE, pp. 253-262, 2006.

[18] Z. Li, S. Lu, S. Myagmar, Y. Zhou,"CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code", IEEE TSE, 32(3), pp. 176-192, 2006.
[19] C. Kapser, M. Godfrey,"Supporting the Analysis of Clones in Software Systems: A Case Study", JSME: Research and Practice, 18(2), pp. 61-82, 2006
[20] R. Geiger, B. Fluri, H. C. Gall, M. Pinzger,"Relation of code clones and change couplings", In Proceedings of the 9th International Conference of Funtamental Approaches to Software Engineering (FASE), number 3922 in Lecture Notes in Computer Science, pp. 411–425, Vienna, Austria, March 2006. Springer.
[21] Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inonue,"On Detection of Gapped Code Clones using Gap Locations".
[22] Foyzur Rahman, Christian Bird, Premkumar Devanbu "Clones: What is that Smell?".

Ms. Sonam Gupta has received her BE degree from Mody Institute of Scinece & Technology, Lakshmangarh, Rajasthan (formely affiliated to Rajasthan University and now a deemed university), Mtech degree from SRM University, Chennai. Currently she is pursuing her PhD from Gyan Vihar University, Jaipur and working as Assistant Professor(CSE) in Ajay Kumar Garg Engineering College, Ghaziabad. Her research interests include software maintenance and evolution.

Dr. P. C. Gupta has received his bachelor degree from Awadhesh Pratap Singh University, Rewa, the master degree from Rani Durga Vati University Jabalpur and the Ph.D. degree from Bundelkhand University, Jhansi. Currently he is heading the Dept. of Computer Sc. & Engg. His area of interest is in pattern matching and classification.