ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Implementing Lexical Approach of Clone Detection and Removal for Type-3

Navdeep Singh, Er.Harpal Singh

PG student, Dept. of CSE, Guru Kashi University, Talwandi Sabo, Bhatinda, India

Assistant Professor, Dept. of CSE, Guru Kashi University, Talwandi Sabo, Bhatinda, India

ABSTRACT: Software Systems are evolving by adding new functions and modifying existing functions over time. Through the evolution process, copy paste programming and other processes leads to duplication of data resulting in model clones or code clones. Since clones are believed to reduce the maintainability of software. Several code clone detection techniques and tools have been proposed. The process is automated by developing a tool that requires no parsing yet is able to detect a significant amount of code duplication. The existence of code clone will make system difficult to debug. In result will increase the cost of the product development and debug. We propose the use of roles to remove such clones since roles provide a finer degree of composition. We sketch four role refactoring to remove code clones and apply them in a case study using the outdraw framework. Results show that roles have a positive impact in clone reduction as they were able to remove almost all clones traditional refactoring could not.

KEYWORDS: Code segments; Code clone; Hybrid approach.

I.INTRODUCTION

Software Clones affects software maintenance and other engineering activities. Hence clone detection and removal has grown as an active area in software engineering research community yielding numerous techniques, various tools and other methods for clone detection and removal. From requirement analysis till software maintenance, it is important to consider various factors that affect its quality, reuse and maintenance. In this respect, software cloning plays an important role. Detecting and removing clones and redundant data will improve the overall efficiency of the software and specifically will ease the maintenance and reuse of the components from the repositories. Cloning works at the cost of increasing lines of code without adding to overall productivity. Same software bugs and defects are replicated that reoccurs throughout the software at its evolving as well its maintenance phase. It results to excessive maintenance costs as well. So cut paste programming form of software reuse raise the number of lines of code without expected reduction in maintenance costs associated with other forms of reuse. So, to eliminate code clones, is a promising way to reduce the maintenance cost in future. In this paper three different aspects of software engineering are considered and are integrated to detect and possibly remove the clones. First, Model based Visual Analysis. Model clones are segments of models that are similar according to some definition of similarity. During software development complete system is modeled by UML diagrams. The model clones can be detected within UML diagrams at the initial phase of development. The removal of model clones will prevent further penetration of model clones as code clones. Second aspect is pattern based semantic analysis. Refactoring patterns are used to find the cloned codes. Refactoring pattern improves the code by detecting clones andthen removing them. Here four cloning patterns are discussed namely extract, pull up, template and strategy pattern. If the same code structure occurs in more than one place, it's sure that code will become better if it is unified.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

II.THEORETICAL FRAMEWORKS

[i]Metrics-Based Data Mining Approach for Software Clone Detection[2012 IEEE 36th International Conference on Computer Software and Applications]

The detection of function clones in software systems is valuable for the code adaptation and error checking maintenance activities. This paper presents an efficient metrics-based data mining clone detection approach. First, metrics are collected for all functions in the software system. A data mining algorithm, fractal clustering, is then utilized to partition the software system into a relatively small number of clusters. For large software systems, the approach is very space efficient and linear in the size of the data set. Evaluation is performed using medium and large open source software systems. In this evaluation, the effect of the chosen metrics on the detection precision is investigated.

[ii]A Survey on Software Clone Detection Research[Chanchal Kumar Roy and James R. Cordy September 26, 2007]
Code duplication or copying a code fragment and then reuse by pasting with or without any modifications is a well-known code smell in software maintenance. Refactoring of the duplicated code is another prime issue in software maintenance although several studies claim that refactoring of certain clones are not desirable and there is a risk of removing them. However, it is also widely agreed that clones should at least be detected.

[iii]Implementing a 3-Way Approach of Clone Detection and Removal using PC Detector Tool[GinikaMahajan, MeenaBharti 2014 IEEE]
This three way technique is based on three steps like first is model based visual analysis second is pattern based semantic analysis third is syntactical code analysis. This approach heslp in identifying code clones which will reduce the debugging cost.

III.PROBLEM FORMULATION

PC Detector detects Type 1 and Type 2 clones. This filtered token based clone detection method, is very efficient and scalable to large software systems. This can further be extended to detect Type3 and Type4 clones.
According to definition, Type 2 clones are syntactically identical fragments, except for variations in identifiers, literals, types, whitespace, layout and comments. Two code fragments having same data structure and same
number of literals with different name can be considered as clones of each other because they can replace each other as their outputs are same. But if two code fragments have renamed literals and different data structures, e.g. one code fragment has integer data type and second one has double data type, these two code fragments cannot be considered as clones as they can never generate same output.
We will identify Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments as type-3 solution.Alsosimalteously identify type-1 and type-2 type clones.
Research Gap:

I. According to existing work if two code fragments have renamed literals and different data structures, e.g. one code fragment has integer data type and second one has double data type, these two code fragments cannot be considered as clones as they can never generate same output. This problem has to be addressed.
II. If Type-2 clone has been detected,later on Copied fragments with further modifications such as changed, added or removed statements should also be identified as type-3 clone.

Objectives and goals:

I. To enhancing the process of type 2.
II. To Identify type-3 clones

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

III. To Implementing using java coding for c program clone detection.

IV.RESULTS AND DISCUSSION

In our implementation we have started from our base paper of 3-way technique for clone detection. We are implementing the clone detection for type-3 type of clones. In this implementation we have used java as programming language. Two source programs written in c-language are to be taken as input. These two programs are being identified as type-3 type of clones.

Methodology

Steps followed

I. In first step two programs written in c are being taken as input.

II. In java two programs as source programs being normalized with type-1 and type2 type of clones.

   I. According to type-1 type of clone all header files,mainstatement,comment statements are being removed.

   II. According to type-2 type of clones textual comparison is being performed. Because in type-2 type of clones, two codes are exactly equal. ie there is copy paste situation according to theoretical framework and according to the base paper.

   III. In our approach we have taken Lexical as two codes comparision technique. In this we have used what is the external dependency and what is the internal dependency. As in type-3 type of clones some lines has been added and some lines of code has been removed. We have to identify that there is equal dependency between two codes.

For example.

Code 1 File:

```
Int x=10,y=20,z;
Printf("X=%dy=%d",x,y);
Printf("enter the value of x and y");
Scanf("%d%d",&x,&y);
Z=x+y;
Printf("Sum=%d",z);
```

Code 2 File:

```
Int x=10,y=20,z;
Z=x+y;
Printf("sum=%d",z);
```

According to our approach for calculating the value of z it is dependent on x and y. In both the codes code dependency is similar. So this may be called as type-3 type of clones. This type of dependency may be called as internal dependency.

Example 2

Code 1

```
Float x,y,z;
x=10.20;
y=sqrt(x);
printf("%f",y);
```

code 2

```
floatx,y,z;
x=34.21;
y=20.2;
```

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

z=pow(x,y);
printf("%f",z);
again according to our approach we have identified that there are similiar no. of external dependencies. So it may again be called as type-3 type of clone.

For representing it in actual we have taken two source files as:

Source file 1

```
#include<stdio.h>
void main()
{
int a;
intc,b;
a=10;
b=20;
float e=0.0;
floatq,x,z;
z=q+x;
if
int d=a+b+c;
else
{
}

else
{}
while
{}
while
{}
do
{}
do
for
for
intx,y,z;
printf("sum");
scanf("%d",d);
printf("sum");
scanf("%d",z );
}
```

Source file 2

```
#include<stdio.h>
void main()
{

inte,r;
e=1o;
r=20;
```

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

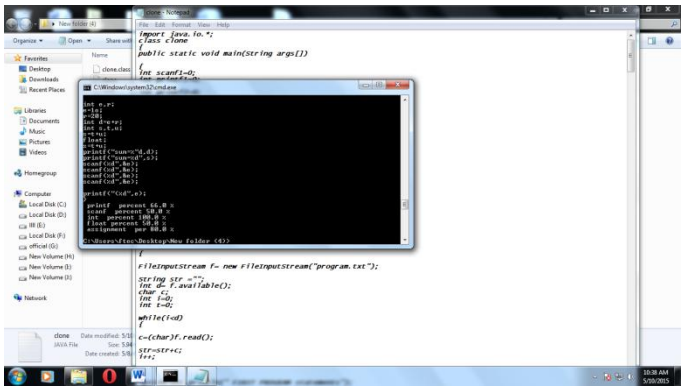
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

```
int d=e+r;
ints,t,u;
s=t+u;
float;
s=t+u;
printf("sum=%"d,d);
printf("sum=%d",s);
scanf(%d",&e);
scanf(%d",&e);
scanf(%d",&e);
scanf(%d",&e);
printf("(%d",e);
}
```

Result Screen:



Screen shot showing the percentage similarity amongst various types of statements

V.CONCLUSION AND FUTURE WORK

Conclusion

As in our implementation we have identified that our type-3 clone identification based on Lexical based technique gives better result for two source codes. In our implementing tool we have undergone identification of both internal and external dependencies. Such that beyond average 60% dependency the two codes are declared to be clones. It will helps the tester to reduce there debugging time. Because tester will try to identify such type of clones and rather than recoding they will build a common module.so that if error exists he need to remove from single place itself.

Future Work: we have done till identifying type-3 and improvement of type-2 clones. In our research we have not taken type-4 type of clones. In future type-4 can also be taken care. Such that all types of clones can be identified with single tool.

REFERENCES

1. Mondal M., Rahman M., Saha R., Roy C., Krinke J. and Schneider K., "An Empirical Study of the Impacts of Clones in Software Maintenance", in *Proceedings of IEEE International Conference on ProgramComprehension*, pp. 242-245, 2011.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

2. Rahman F., Bird C., Devanbu P., "Clones: What is that smell?", accepted to Empirical Software Engineering, an International Journal 2011 Springer-Verlag.
3. Choi E., Yoshida N., Ishio T., Inoue K. and Sano T., "Finding Code Clones for Refactoring with Clone Metrics: A Case Study of Open Source Software", in *Proceedings of The Inst. Of Electronics, Information andCommunication Engineers (IEICE)*, pp. 53-57, July 2011.
4. Mondal M., Rahman M., Saha R., Roy C., Krinke J. and Schneider K., "An Empirical Study of the Impacts of Clones in Software Maintenance", in *Proceedings of IEEE International Conference on ProgramComprehension*, pp. 242-245, 2011.
5. Störrle H., "Towards Clone Detection in UML Domain Models", in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pp. 285-293, 2010.
6. Mahajan G., and Ashima., "Software Cloning in Extreme Programming Environment", *International Journal of Research in IT and Management*, vol. 2, no. 2, pp. 1906-1919, February 2012.
7. Lakhotia A., Li J., Walenstein A. and Yang Y., "Towards a Clone Detection Benchmark Suite and Results Archive", in *Proceedings of the11th IEEE International Workshop on Program Comprehension*, pp. 285-286, 2003.
8. Kamiya T., Kusumoto S., and Inoue K., "CCFinder: A MultiLinguistic Token-Based Code Clone Detection System for Large Scale Source Code", *Journal IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654-670, July 2002.
9. Kapser C. and Godfrey M., "Cloning Considered Harmful", in *Proceedings of the 13th Working Conference on Reverse Engineering*, pp.19-28, 2006.
10. Rysselberghe F. V., Demeyer S., "Evaluating Clone Detection Techniques" in *Proceedings of 19th IEEE International Conference onAutomated Software Engineering (ASE)*, pp. 336-339, September 2004.
11. Fowler M. and Beck K., "Refactoring: Improving the Design of Existing Code", The Addison Wesley Object Technology Series, Addison Wesley, Boston, 2000.
12. Kelter U., Wehren J., and Niere J., " A Generic Difference Algorithm for UML Models", in *Proceedings of Natl. Germ. Conf. Software-Engineering* 2005 (SE'05),pp. 105-116, 2005.
13. Roy C. K., Cordy J. R. and Koschke R., "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach",
14. *Science of Computer Programming*, vol. 74, no. 7, pp. 470-495, May2009.