ISSN:0975-9646

To Enhance Type 4 Clone Detection in Clone Testing

Swati Sharma[#1], Priyanka Mehta[#2]
[1]*M.Tech Scholar,*
[2]*Head of Department,*
Department of Computer Science & Engineering,
Universal Institute of Engineering & Technology,
Lalru, Punjab, India

***Abstract*— The means of software reuse is copying and modifying block of code that detect cloning. As a survey, it is observed that 20-30% of module in system may be cloned. So it is mandatory to detect clones in system to reduce replication and improve reusability.**

Code clone is similar or duplicate code in source code that is created either by replication or some modifications. Clone is a persistent form of Software Reuse that effect on maintenance of large software. In previous research, the researcher emphasis on detect type 1, type 2, and type 3 of type of clones. The existing code clone detection tools are used to detect clone in source code. In this research, the enhancement in code clone detection algorithm will be proposed which detect type 4. In this work, firstly, use an existing algorithm to detect clone. Secondly, we put some intensification in that algorithm to detect clone. Thirdly, we combine algorithm with type 4 to detect a clone in particular function.

By using type 4, the efficiency of clone detection is increased. Clone is detected in particular function, which is more accurate and more efficient in manner.

***Keywords*— Software clone, clone detection, clone testing, code clone, algorithm, effectiveness of software.**

I. INTRODUCTION

Software engineering means building, evolving and maintaining software systems. Software engineering means set of problem solving skills, techniques, technology and methods applied upon a variety of domains to evolve and create useful systems that solve many problems like practical problems. Software engineer handles software engineering projects that discover, create, build software and tells its behaviour [15].Software means a non-tangible device like documentation and computer programs and it is different from tangible hardware device. Software Engineering is the branch of computer science which applies engineering principles to create, operate, modify and maintain of software components [21]. Software engineers adopt an organized and systematically approach regarding their work using some techniques and tools depending upon the resources available and problem to be solved. System engineering is different from software engineering. System engineering means deployment, architectural design and integration where as software engineering is concern with development, quality and testing and control of the system.

The main goals of software engineering are as follows.

- To produce software of high quality having less cost.
- To achieve Correctability.
- To gain reliability.
- To improve efficiency.
- To produce the system under budget and on schedule.

A. Basic Activities of Software Engineering

The basic activities which are necessary to follow in software engineering are as follow:-

- Design of the product
- Implementation of the product
- Testing of the product.
- Integrates sub parts and test them as a whole.
- Maintenance of the system.

B. Software Cloning

Software clones are the regions of source code which are highly similar; these regions of similarity are called clones, clone classes, or clone pairs. While there are several reasons why two regions of code may be similar, the majority of the clone analysis literature attributes cloning activity to the intentional copying and duplication of code by programmers; clones may also be attributable to automatically generated code, or the constraints imposed by the use of a particular framework or library. In addition to these, some other issues, including programmer's behaviour such as laziness and the tendency to repeat common solutions, technology limitations, code understand ability and external business forces have influences on code cloning. Cloning is the unnecessary duplication of data whether it is at design level or at coding level.

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 deceivingly raise the number of lines of code without expected reduction in maintenance costs associated with other forms of reuse. So, clones, is a promising way to reduce the maintenance cost in future.

Clone detection techniques play an important role in software evolution research where attributes of the same code entity are observed over multiple versions.
The reasons why programmers duplicate code are:

- Making a copy of a code fragment is simpler and faster than writing the code from scratch. In addition, the fragment may already be tested so introduction of a bug seems less likely.
- Evaluating the performance of a programmer by the amount of code he or she produces gives a natural incentive for copying code.
- Efficiency considerations may make the cost of a procedure call or method invocation seems too high a price. In industrial software development contexts, time pressure together with first and second points lead to plenty of opportunities for code duplication.

According to the definition of cloning, there can be different notions of similarity. They can be based on lexical or syntactic structure or can be semantics, or functionality. They can even be similar if follow the same patterns, that is, the same building plan. Semantic similarity relates to the observable behavior. A piece of code, A, is similar to another piece of code, B, if B subsumes the functionality of A.

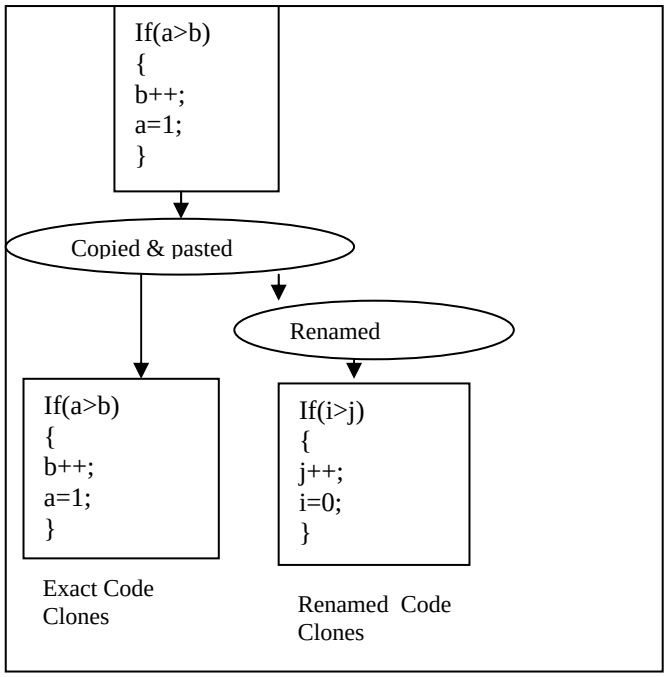

Fig. 1 Source code with its clones

C. Background of Software Cloning

Basic clone detection terms:

1) Code Fragment: A code fragment a piece of code including function definition, begin-end block, or sequence of statements. We use file name and begin-end line numbers in the original code base to identify code fragment.

2) Code Clone: A code clone is a similar or duplicate code fragment in a source code or created either by replication or some modifications.

3) Clone Pair: if there is an equivalence relation between two code segments, then they form a clone pair.

4) Clone Class: It is defined as collection of similar code segments. Each code segment in a clone class form a clone pair with other code segment of that class.

5) 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.

6) Renamed Clones: People use the term renamed clones when identifier names, literals values, comments or whitespace changes in the copied fragments. Thus, a renamed clone is essentially a Type II clone.

7) Near-Miss Clones: These are those clones where the copied fragments are very similar to the original. Editing activities such as changing in comments, layouts, changing the position of the source code elements through blanks and new lines, changing the identifiers and literals.

8) Semantic Clones: Semantic Clones are defined as functionally identical code fragments.

D. Types of Code Clones

There are basically four types of code clones. They are explained following:-

1) Type 1(Exact clones): In Type I clones, a copied code fragment is the same as the original. These code clones are identical code clones with some modification in white space and comments. Type I is widely known as Exact clones.

Let us consider the following code fragment:

```
if (a >= b) {
c = d + b; // Comment1
d = d + 1;}
else
c = d - a; //Comment2
```

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

```
if (a>=b) {
// Comment1'
c=d+b;
d=d+1;}
else // Comment2'
c=d-a;
```

We see that these two fragments are textually similar (even line-by-line) after removing the whitespace and comments.
A typical line-by-line technique may fail to detect such clones that vary in layout.

2) Type 2(Renamed/parameterized clones): A Type II clone is a code fragment that is the same as the original except for some possible variations about the corresponding names of user-defined identifiers (name of variables, constants, class, methods and so on), types, layout and comments.

Let us consider the following code fragment.

```
if (a >= b) {
```

```
c = d + b; // Comment1
d = d + 1 ;}
else
c = d - a; //Comment2
```

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

```
if (m >= n)
{ // Comment1'
y = x + n;
x = x + 5; //Comment3
}
else
y = x - m; //Comment2'
```

We see that the two code segments change a lot in their shape, variable names and value assignments. However, the syntactic structure is still similar in both segments.

3) Type 3(Near Miss Clones): These code clones are copied fragments by changing, adding or removing statements.

Consider the original code segment:

```
if (a >= b) {
c = d + b; // Comment1
d = d + 1;}
else
c = d - a; //Comment2
```

If we now extend this code segment by adding a statement e = 1 then we can get,

```
if (a >= b) {
c = d + b; // Comment1
e = 1; // This statement is added
d = d + 1; }
else
c = d - a; //Comment2
```

This copied fragments with one statement inserted is called Type III code clone of the original with a gap of one statement inserted.

4) Type 4(Semantic Clones): These code clones are based on function similarity but they are different in syntax. These clones are termed as Type IV semantic clones In this type of clones, the cloned fragment is not necessarily copied from the original. 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. Let us consider the following code fragment 1, where the final value of 'j' is the factorial value of the variable VALUE.

Fragment 1:

```
int i, j=1;
for (i=1; i<=VALUE; i++)
j=j*i;
```

Now consider the following code fragment 2, which is actually a recursive function that calculates the factorial of its argument n.

Fragment 2:

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

II. CLONE TESTING

A code clone means similar or duplicate code in a source code or code that is created either by replication or some modifications .These cloned code needs high maintenance cost of software and also cause the code bloating. This is because when changes are performed on one clone, then the same action is performed on respected clone, this will increase the maintenance. These clones can also increase risk of faults in system. Past research conclude that around 7% to 23% of the source code in a software system contains code clone. There are number of techniques and tools to detect the code clones, but it is not effective to remove the clones. Because code clones are needed for software to function properly. So we can apply the principal of refactoring or modularity to improve the reusability and maintainability of software from clone code.

A. Clone detection techniques

There are basically 4 types, that are:- textual, lexical, syntactic and semantic.

1) Textual approach: In Textual approaches there is little need of normalization or transformation of code. In this, basically line to line comparison is done, which basically based on two types, one is simple line matching and other one is parameterized line matching. This technique is basically string based.

2) Lexical approach: In lexical technique we convert source code into tokens using lexical rules. These tokens are then compared.

3) Syntactic approach: In syntactic technique an abstract tree is generated. Using parser source code is converted into parse tree. Abstract tree is then processed either using tree matching or metric to find the clones.

4) Semantic approach: In this approach, a source code is represented as program dependency graph. Nodes represent the statements and expressions and edges represent control and data dependencies.

III. PROPOSED METHODOLOGY

This algorithm safely detects the function clones in the source code. In this algorithm ant colony technique is used to detect the function clones.

This algorithm is basically an enhanced version of the proposed algorithm.

```
If(file is distinct)
Begin
Initialization: i:=1, clone=0
while(i<=n)
if( Functions are distinct)
if(clone is distinct from any previously processed clone)
```

```
then
if(startNo_Clone has method parameters)
then
compose advice specification with parameter binding
Initialization: line:= startNo_Clone+ 1
while(line<= endNo_Clone)
copy line to Aspect text area
line++
end while
else
compose advice specification without parameter binding
Initialization: line:= startNo_Clone + 1
while(line<= endNo_Clone)
copy line to Aspect text area
line++
end while
end if
end if
i++
end while
Initialization: i=1
while(i<=n)
comment out clone from array full_File[ ][ ]
i++
end while
end begin
```

This algorithm is invoked after clones are detected by ant colony optimization technique. The input of this algorithm is file name, function names. In line 1, the algorithm is firstly checking the file name. In line 3, the algorithm begins by initializing the variables. In line 5 and 6 function name will checked and then different types of clones will be checked. If the function name is distinct then it will also check the internal part of the function. From line 8-15, when clone is detected in the line, then line number will be incremented by 1 and control will go to the next line. This procedure will continue till the end of the function. From line 17-26, it starts to find the function clone without parameter binding. From line 27-32, it starts to save the information about the file in the form of array.

IV. RESULTS

As illustrated in the Fig 2 interface is developed for clone testing. The tool will test the efficiency of code cloned with existing and with new algorithm for different lines of first and second codes. Here we will firstly choose the lines from first and second code from which we want to compare and find the clones and then we can choose old as well as new algorithm. The efficiency of clone testing with new algorithm is always greater than the efficiency of clone testing with old algorithm. The tool also provides the option to reset the chosen lines. The tool also provides options that draw bar graphs of corresponding efficiencies of clone testing with new and old algorithms. We can also show the graphical comparison of the efficiencies of code cloned with both the algorithms with Graph option. The exit option is used to exits the tool.

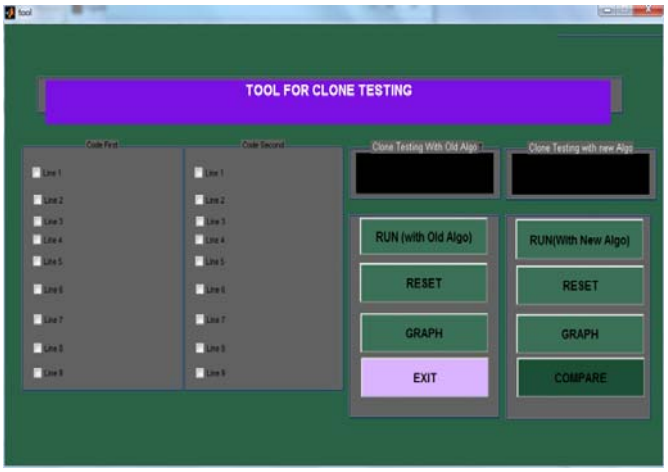

Fig. 2 Tool for clone testing

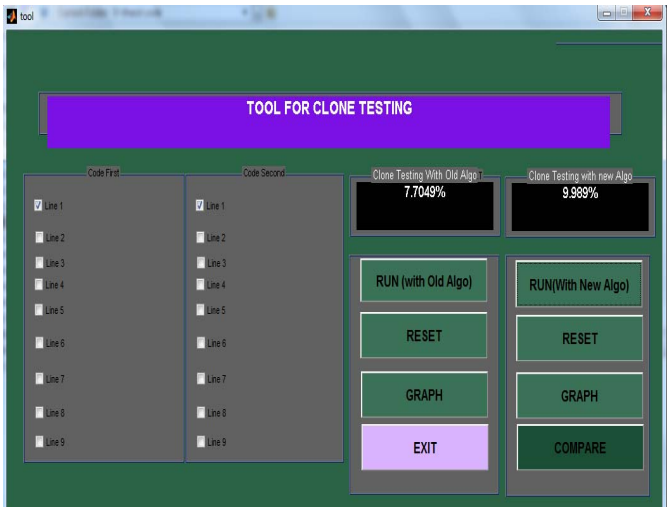

Fig. 3 Efficiency comparison between new algorithm and old algorithm

As illustrated in the Fig. 3, when Line 1 of first code and the Line 1 second code are checked, the tool calculated that 7.7049% of the code is cloned with the old algorithm where as 9.989% of the code is cloned with the new algorithm.

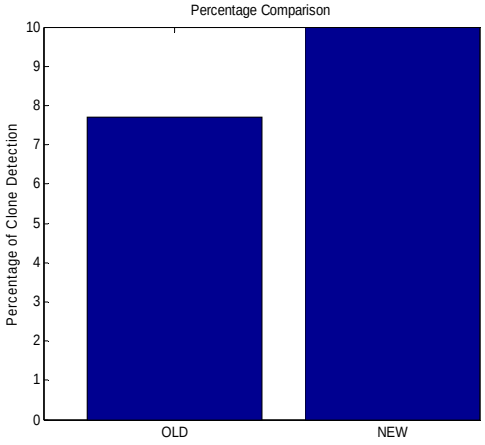

Fig. 4 Graphical Comparison of efficiency with old and new algorithm

As illustrated in Fig. 4, the bar graph shows the comparison of efficiencies of code cloned with the old algorithm as well as with the new algorithm when the Line 1 of first code and Line 1 of the second code are checked and tool calculated that 7.7049% of the code is cloned with the old algorithm and 9.989% of the code is cloned with the new algorithm.

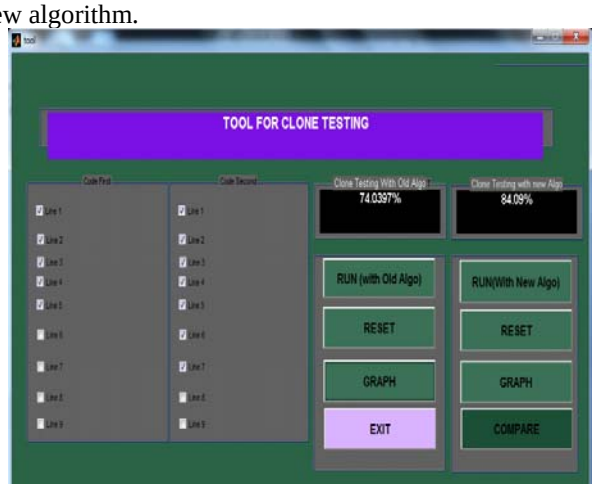

Fig. 5 Efficiency comparison between new algorithm and old algorithm

As illustrated in the Fig. 5, when Line 1 to Line 5 of first code and the Line 1 to Line 7 of second code are checked, the tool calculated that 74.0379% of the code is cloned with the old algorithm where as 84.09% of the code is cloned with the new algorithm.

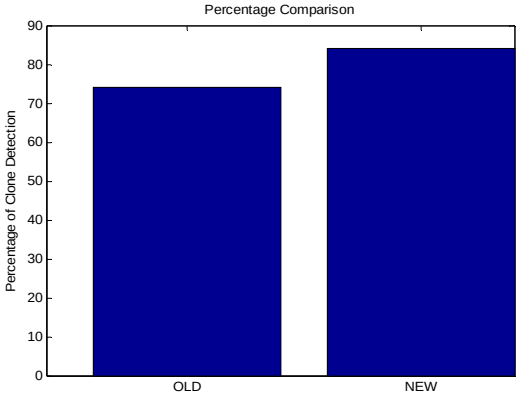

Fig. 6: Graphical Comparison of efficiency with old and new algorithm

As illustrated in Fig. 6, the bar graph shows the comparison of efficiencies of code cloned with the old algorithm as well as with the new algorithm when Line 1 to Line 5 of first code and the Line 1 to Line 7 of second code are checked and tool calculated that 74.0379% of the code is cloned with the old algorithm where as 84.09% of the code is cloned with the new algorithm.

V. Conclusion

In this paper, we have presented an algorithm approach that is used to detect the function clones in software. Function clones are harmful to software system because they increase the maintenance cost.

Our approach in this paper is to handle the type 4 clone. Because only type 4 can detect the function clones in source code. The results also show the comparison of new algorithm with old algorithm. By comparison, we analyze that enhancement of old algorithm increase efficiency, decrease maintenance cost and find out both code clone and function clones.

VI. Future Scope

In this work, the enhanced algorithm has proposed to detect function clones in source code. The further enhancement technique will detect clone line by line and will tell in which line, clone exist. It will also assign severity to the detected clones. The severity of the clones will provide better analysis in terms of code clone detection.

References

[1] C. K. Roy, "*Detection and Analysis of Near-Miss Software Clones*", Ph.D. Thesis, Queen's School of Computing, Queens University, 2009-08-31, 14:05:30.233.

[2] J.H. Johnson, "*Identifying redundancy in source codeusing fingerprints*", Proceedings of the 1993 Conferenceof the Centre for Advanced Studies on Collaborative.

[3] B.S. Baker, "*On finding duplication and nearduplicationin large software systems*", Proceedings ofthe 2nd Working Conference on Reverse Engineering.

[4] T. Kamiya, S. Kusumoto, K. Inoue, "*CCFinder: AMultilinguistic Token-Based Code Clone Detection System for Large Scale Source Code*", IEEETransactions on Software Engineering.

[5] Kodhai.E, Perumal.A, Kanmani.S, "*clone detection using textual and metrics analysis to figure out all types of clones*" in ITC, 2010.

[6] Rubala Sivakumar, Kodhai.E "*code clone detection in website using hybrid approach*", in IJCA(0975-888) volume 48-No.13,june 2012.

[7] Jean Mayrand , Claude Leblanc , Ettore M. Merlo "*Experiment on automatic detection of function clone in a software system using metrics*". In proc of the Int'l Conf. on software maintenance, page 244,1996.

[8] Robin Sharma , Dr. Sushil Garg "Hybrid *approach for efficient software clone detection*" ISSN:2250-3498 April 2013.

[9] Salwa K.abd-El-Hafiz "*A Metrix Based Data Mining Approach For Software Clone Detection*", proc. IEEE 36th intenational conference on computer software and application,2012.

[10] Amandeep Kaur, Balraj Singh, "*Study on Metrix Based Approach for Detecting Software Code Clones*" ISSN:2277 128X 2014.

[11] Rainer Koschke, Raimar Falke, Pierre Frenzel " *Clone Detection using Abstract Syntax Suffix Trees*"-working conference on Reverse Engineering-2006.

[12] Peter Bulychev and Marius Minea, "*Duplicate Code Detection using Anti-Unification*", A Survey on Software Clone *Detection* Research, 2007.

[13] Jens krinke, "*Identifying Similar Code with Program Dependency Graph*", proc.eighth working Conf. Reverse Eng,2001, pp 301-309.

[14] Raghavan komondoor and Susan Horwitz "*Using Slicing to Identify Duplication in Source Code*", proc. Int. Symp. Static Analysis,2001.

[15] Swarnendu Biswas and Rajiv Mall , " *A approach to software engineering*", 2009.

[16] http://www.slideshare.net/engineerrd/software-requirement

[17] Myers, Glenford. (1979). "*The Art of Software Testing*".

[18] Åshäll, F. N. "*Testing The Product Propagation*" ,2011.

[19] http://guru99.199tech.com/software-testing-introduction-portance.html

[20] http://arxiv.org/abs/1205.5615

[21] Angad Singh Gakhar, "*Converting Code Clones To Aspects Using Algorithm Approaches*",2009.

[22] Jovanovic Irena, "*Software testing methods and techniques*", 2002.

[23] Salwa K.Abd-El-Hafiz, "*Code Cloning: The Analysis, Detection and Removal*", International Journal of Computer Applications,April 2013.