Detection of gapped code clones using Solid SDD tool

Prabhjot Kaur[1], Rupinder kaur[2]

[1] *Assistant Professor, GZS, Punjab technical University Bathinda, Punjab, India*

[2] *Assistant Professor, YCOE, Punjabi University campus Talwandi sabo, Punjab, India*

ABSTRACT

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. Several studies show that about 5% to 20% of a software systems can contain duplicated code, which is basically the results of copying existing code fragments and using then by pasting with or without minor modifications.. Since developers usually modify the copied-and-pasted code portions, there are some gaps between the original code portion and it. Here, we call such code portions include some gaps Gapped code clone. Up to the present, several code clone detection methods, which give consideration to such gap, have been proposed. However, it needs a lot of cost to detect all the gapped code clones. This paper proposes a new method to visualize the gapped code clones just as if they were actually detected, based on the detection results of conventional code clone using Solid SX and Solid SDD tool.

Keywords: *Software Maintenance, Gapped Code Clone,Solid SX, solid SDD tool.*

1 Introduction

CLONE a general term for the research activity that creates a identical copy of some entity.

Software that copies the entire hard disk to another hard disk or to another computer via direct cable or the network. Cloning is used to back up the computer for a future system restore and to provide interim storage for replacing the hard disk. It is also used to make a master copy of a computer configuration for quick installations on new machines. The cloned disk is copied verbatim (cloned again) to the new computer much quicker than installing the OS and applications individually.

You've volunteered to set up a 20-machine computer lab at your local youth center, but don't have the time to install the operating system and drivers, download software patches, and perform all sorts of additional configurations on every computer. One solution? Cloning.

Computer cloning, or ghosting as it is sometimes called, is a process that involves setting up the operating system, drivers, software, and patches on a single computer, then automatically replicating this same setup on other computers using specialized software.

Copying code fragments and then reuse by pasting with or without minor modifications or adaptations are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment (with or without modifications) is called a clone of the original.

Such activities are very easy and can significantly reduce programming effort and time as they reuse an existing fragment of code rather than rewriting similar code from scratch. This practice is common, especially in device drivers of operating systems where the algorithms are similar. . Depending on the type of software you use, cloning can be done using a disk or other media, or over a network. For example, Acronis True Image and Symantec Ghost are popular commercial products to clone PCs.

2. Why Do Clones Occur

Code clones do not occur in software systems by themselves. There are several factors that might force or influence the developers and/or maintenance engineers in

making cloned code in the system. Clones can also be introduced by accidents.

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

2.1.1 Reuse Approach

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

Simple reuse by Copy/Paste: It is a fast way of reusing reliable semantic and syntactic constructs[1].

Forking:For example, when creating a driver for a hardware family, a similar hardware family may already have a driver, and thus can be reused with slight modifications. Similarly, clones can be introduced when porting software to new platforms and so on[2].

Design, functionalities and logic reuse: Functionalities and logic can be reused if there is already similar solution available. For example, there is a high similarity between the ports of a subsystem (especially, for OS's subsystems). 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.

2.1.2 Programming Approach

Clones can be introduced by the way a system is developed. Example :

Merging of two similar systems: Sometimes two software systems of similar functionalities are merged to produce a new one.

Delay in restructuring: It is also a common practice to the developers that they often delay in restructuring their developed code which may ultimately introduce clones.

2.2 Maintenance Benefits

Clones are also introduced in the systems to obtain several maintenance benefits. Examples are:

Risk in developing new code: Clones do frequently occur in financial software as there are frequent updates/enhancements of the existing system to support similar kinds of new functionalities. The developer reuse the existing code by copying and adapting to the new product requirements because of the high risk of software errors in new fragments and because existing code is already well tested [3].

Speed up maintenance: As two cloned code fragments are independent of each other both syntactically and semantically, they can evolve at different paces in isolation without affecting the other and testing is only required to the modified fragment. [4].

High cost of function calls in real time programs: If the compiler does not offer to inline the code

2.1 Development Strategy

automatically, this will have to be done by hand and consequently there will be clones.

2.3 Overcoming Underlying Limitations

Clones can be introduced due to the underlying limitations concerning the programming languages of interest and the developers.

2.3.1 Language Limitations

Clones can be introduced due to the limitations of the language, especially when the language in question does not have sufficient abstraction mechanisms. Examples are:

Lack of reuse mechanism of programming languages: Sometimes programming languages do not have sufficient abstraction mechanisms, e.g., inheritance, generic types(called templates in C++) or parameter passing (missing from, e.g., assembly language and COBOL) and consequently, the developers are required to repeatedly implement these as idioms. Such repeating activities may create possibly small and potentially frequent clones

Significant efforts in writing reusable code: It is hard and time consuming to write reusable code. Perhaps, it is easier to maintain two cloned fragments than the efforts to produce a general but probably more complicated solution.

Writing Reusable code is error-prone: Writing reusable code might be error-prone, especially for a critical piece of code. It is therefore preferred to copy the existing code and then reuse it by pasting with or without modification rather than making reusable code.

.2.3.2 Programmer's Limitations

There are also several limitations associated with the programmers for which clones are introduced in the system. Examples are:

Difficulty in understanding large system: It is generally difficult to understand a large software system. This forces the developers to use the example-oriented programming by adapting existing code developed already.

Time limit assigned to developers: One of the major causes of code cloning in the system is the time frame allowed to its developers. In many cases, the developers are assigned a specific time limit to finish a certain project or part of it. Due to this time limit, developers look for an easy way of solving the problems at hand and consequently look for similar existing solutions. They just

copy and paste the existing one and adapt to their current needs.

Wrong method of measuring developer's productivity: Sometimes the productivity of a developer is measured by the number of lines he/she produces per hour. In such circumstances, the developer's focus is to increase the number of lines of the system and hence tries to reuse the same code again and again by copying and pasting with to make a new solution even after finding a similar existing solution and thus, reusing the existing one gets higher priority than making a new one.

Lack of ownership of the code to be reused: Code may be borrowed or shared from another subsystem which may not be modified because it may belong to a different department or even may not be modifiable

2.4 Cloning By Accident

Clones may be introduced by accidents. Examples are: Protocols to Interact with APIs and Libraries: The use of a particular API normally needs a series of function calls and/or other ordered sequences of commands. For example, when creating a button using the Java SWING API, a series of commands is to create the button, add it to a container, and assign the action listeners. Similar orderings are common with libraries as well [2]. Thus, the uses of similar APIs or libraries may introduce clones.

Coincidentally implementing the same logic by different developers: It may happen that two developers were involved in implementing the same kind of logic and eventually, come up with similar procedures independently, thus leading to look-alikes more than clones.

Side effect of developers' memories: Programmers may unintentionally repeat a common solution for similar kind of problems using the common solution pattern of his/her memory to such similar problems. Therefore, several clones may unknowingly be created to the software systems.

3 Drawbacks of Code Duplication

The factors behind cloning described in Section 2 are reasonable and consequently, clones do occur in large software systems. While it is beneficial to practise cloning, code clones can have severe impacts on the quality, reusability and maintainability of a software system. In the following we list some of the drawbacks of having cloned code in a system.

adaptations instead of following a proper development strategy.

Developer's Lack of knowledge in a problem domain: Sometimes the developer is not familiar to the problem domain at hand and hence looks for existing solutions of similar problems. Once such a solution is found, the developer just adapts the existing solution to his/her needs.

Increased probability of bug propagation: If a code segment contains a bug and that segment is reused by coping and pasting without or with minor adaptations, the bug of the original segment may remain in all the pasted segments in the system and therefore, the probability of bug propagation may increase significantly in the system .

Increased probability of introducing a new bug: In many cases, only the structure of the duplicated fragment is reused with the developer's responsibility of adapting the code to the current need. This process can be error prone and may introduce new bugs in the system .

Increased probability of bad design: Cloning may also introduce bad design, lack of good inheritance structure or abstraction. Consequently, it becomes difficult to reuse part of the implementation in future projects. It also badly impacts on the maintainability of the software.

Increased difficulty in system improvement/modification: Because of duplicated code in the system, one needs additional time and attention to understand the existing cloned implementation and concerns to be adapted, and therefore, it becomes difficult to add new functionalities in the system, or even to change existing ones

Increased maintenance cost: If a cloned code segment is found to be contained a bug, all of its similar counterparts should be investigated for correcting the bug in question as there is no guarantee that this bug has been already eliminated from other similar parts at the time of reusing or during maintenance. Moreover, When maintaining or enhancing a piece of code, duplication multiplies the work to be done .

Increased resource requirements: Code duplication introduces higher growth rate of the system size. While system size may not be a big problem for some domains, others (e.g., telecommunication switch or compact devices) may require costly hardware upgrade with a software upgrade. Compilation times will increase if more code has to be translated which has a detrimental effect on the edit-compile-test cycle [5] .

4. Clone Relation Terminologies

Clone detection tools report clones in the form of *Clone Pairs* (CP) or *Clone Classes* (CC) or both. These two terms speak about the similarity relation between two or more cloned fragments. The similarity relation between the cloned fragments is an equivalence relation(i.e., a reflexive, transitive, and symmetric relation) A clone-relation holds between two code portions if (and only if) they are the same sequences. Sequences are sometimes original character strings, strings without whitespace, sequences of token type, transformed token sequences and so on. In the following we define *clone pair* and *clone class* in terms of the clone relation:

Clone Pair[6]: A pair of code portions/fragments is called a clone pair if there exists a clone relation between them, i.e., a clone pair is a pair of code portions/fragments which are identical or similar to each other. For the three code fragments, Fragment 1 (F1), Fragment2 (F2) and Fragment 3 (F3) of Figure , we can get five clone pairs, <F1(a), F2(a)>, <F1(b), F2(b)>, <F2(b), F3(a)>, <F2(c), F3(b)> and <F1(b), F3(a)>. By considering the maximum possible extent of cloned segments, we get basically four clone pairs, <F1(a + b), F2(a + b)>, <F2(b + c), F3(a + b)> and <F1(b), F3(a)>.

Clone Class[6]: A clone class is the maximal set of code portions/fragments in which any two of the code portions/fragments hold a clone-relation, i.e., form a clone pair. For the three code fragments of Figure , we get a clone class of <F1(b), F2(b), F3(a)> where the three code portions F1(b), F2(b) and F3(a) form clone pairs with each other, i.e., there are three clone pairs, <F1(b), F2(b)>, <F2(b), F3(a)> and <F1(b), F3(a)> do exist too. A clone class is therefore, the union of all clone pairs which have code portions in common. Clone classes are also called clone communities .

Clone Class Family[7]: The group of all clone classes that have the same domain is called a clone class family. Such a clone class family is also termed *super clone* . In their context: *Multiple clone classes between the same source entities are aggregated into one large super clone*.

5. Code Clone Types

There are basically two kinds of similarities between two code fragments. Two code fragments can be similar based on the similarity of their program text or they can be similar in their functionalities without being textually similar. The first kind of clones are often the result of copying a code fragment and then pasting to another location.

Textual Similarity: Based on the textual similarity we distinguish the following types of clones :

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

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

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.

Functional Similarity: If the functionalities of the two code fragments are identical or similar i.e., they have similar pre and post conditions, we call them semantic clones and referred as *Type IV* clones.

Type IV: Two or more code fragments that perform the same computation but implemented through different syntactic variants. This increasing level of analytical complexity from *Type I* through *Type IV* does not vary whether the process is automatic or not.

Type I Clones

In *Type I* clones, a copied code fragment is the same as the original. However, there might be some variations in whitespace (blanks, new line(s), tabs etc.), comments and/or layouts.

Type I is widely know 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. However, even after removing the comments and whitespace, the following code fragment is not similar to the previous two on a line-by-line basis as the positions of the \f" and "*g*" are changed in the code. Nevertheless, this fragment is also *Type I* exact copy clone of the other two.

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

A typical line-by-line technique may fail to detect such clones that vary in layout.

Type II 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. The reserved words and the sentence structures are essentially the same as the original one. Let us consider the following code fragment.

```
if (a >= b) {
c = d + b; // Comment1
15
d = d + 1;}
```

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'
```

Type III Clones

In *Type III* clones, the copied fragment is further modified with statement(s) changed, added and/or deleted. 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.

Type IV Clones

Type IV clones are the results of semantic similarity between two or more code fragments. 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, i.e., captures similar functional properties and similarity assessment methods rely on matching of pre/post-conditions. Let us consider 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) ;
}
```

7. Advantages and Applications of Detecting Code Clones

In addition to the direct benefit of knowing how to improve the quality of the source code by refactoring the cloned code, there are several other benefits and applications of detecting clones. We list some of those as follows:

Detects library candidates: a code fragment that has been copied and reused multiple times in the system apparently proves its usability. As a result, this fragment can be incorporated in a library, to announce its reuse potential officially.

Helps in program understanding: If the functionality of a cloned fragment is comprehended, it is possible to have an overall idea on the other files containing other similar copies of that fragment. For example, when we have a piece of code managing memory we know that all files which contain a copy must implement a data structure with dynamically allocated space

Helps aspect mining research: Detecting similar code is also required in aspect mining for detecting cross-cutting concerns. The code of cross-cutting concerns is typically duplicated over the entire application and could be identified with clone detection tools.

Finds usage patterns: If all the cloned fragments of a same source fragment can be detected, the functional usage patterns of that fragment can be discovered .

Detects malicious software: Clone detection techniques can be used in finding malicious software. By comparing one malicious software family to another, it is possible to find the evidence where parts of one software system match parts of another .

Detects plagiarism and copyright infringement: Finding similar code may also useful in detecting plagiarism and copyright infringement .

Helps software evolution research: Clone detection techniques are successfully used in software evolution analysis by looking at the dynamic nature of different clones in different versions of a system.

Helps in code compacting: Clone detection techniques can be used for compact device by reducing the source code size .

8. Clone detection tool: solidSX and solidSDD

SolidSX and SolidSDD are provided as self-contained installers, freely available for research [4] on Windows XP and later editions. The solidSX tool supports the analysis of software structure, dependencies, and metrics. It gives basic metrics like code and comment size, complexity, fan-in, fan-out, and symbol source code location. Clone detection uses the

same algorithm as CCfinder [1], configurable by clone length (in statements), identifier renaming (allowed or not), size of gaps (inserted or deleted code fragments in a clone), and whitespace and comment filtering. Nodes and edges have metrics, e.g. percentage of cloned code, number of distinct clones, and whether a clone Includes identifier renaming or not. Figure shows the text view of SolidSDD. The table shows all files with percentage of cloned code, number of clones, and presence of identifier renaming. Sorting this table allows e.g. finding files with the most clones or highest cloned code percentage.

8.1 Study of SolidSDD

The tool extracts clones from java files. It takes as the input path of source files and displays the result by highlighting the same lines.

Step1: Giving input file

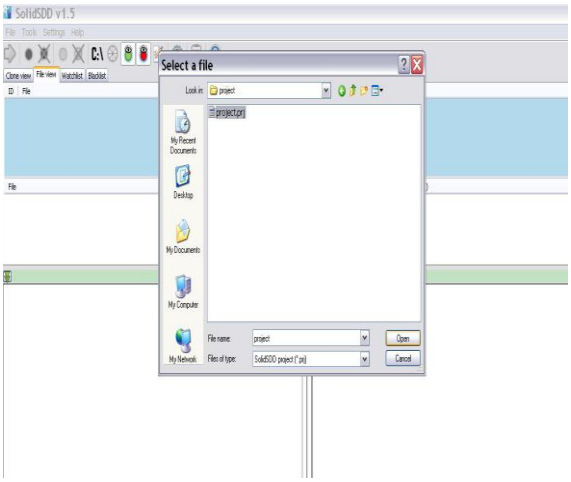

Figure 1: shows opening a already build project

Step 2: Finding duplicates in file

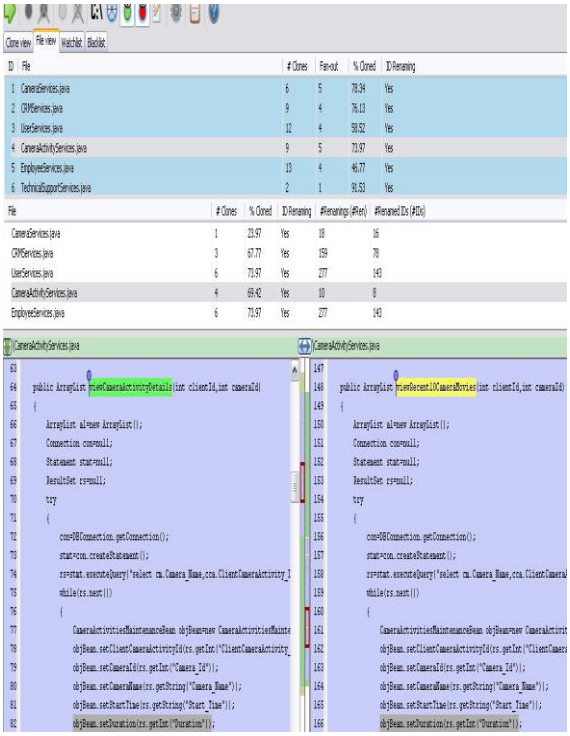

Figure2: File view of the project

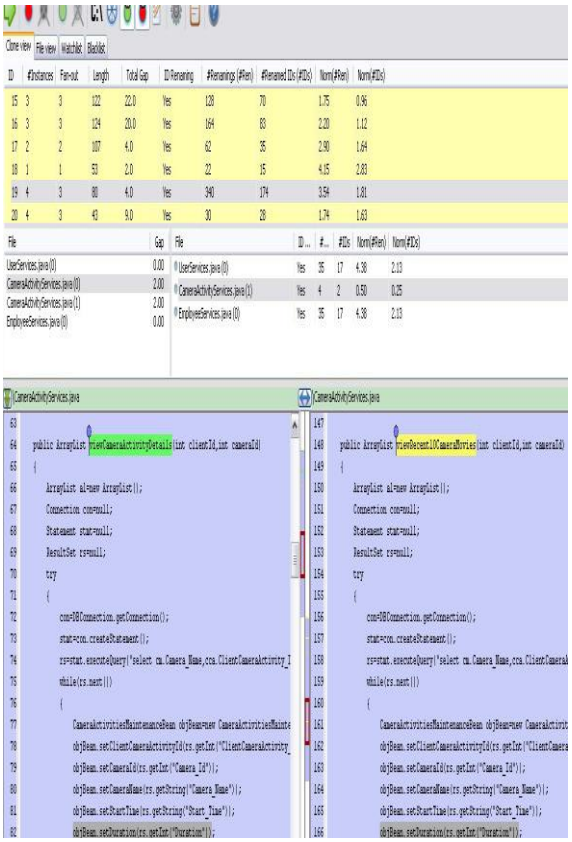

Figure3: Clone view of the project

Step3: Duplication summary

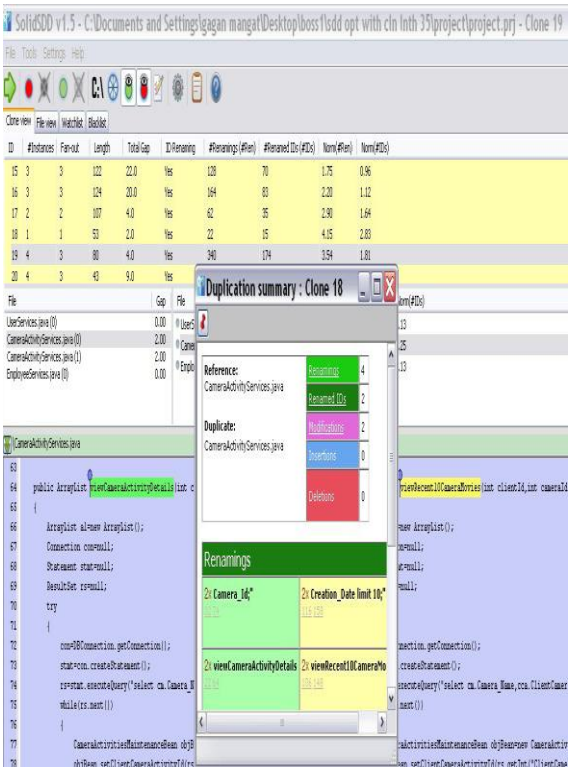

Figure 4: shows duplication summary of the project

Figure shows the clone duplication summary of specific file of project, the modifications like insertions, deletion etc. It also shows any renaming of identifiers.

9. Code clone metrics used for analysis of tool.

The clone set metrics are:

a) Population of clone class (POP) [4,5]: Population of a Clone Class; POP is the number of elements of a given clone class C. A clone class with a large POP means that similar code portions appear in many places. Higher POP(S) values mean that code clones in a clone set appear more frequently in the system. Contrary to this, lower POP(S) values mean that code clones in a clone set appear in fewer places is the system.

b) LEN [4,5] : LEN(S) is the average length of token sequences of code clones in clone set S. Higher LEN(S) values mean that each code clone in a clone set S consists of more token sequences. Contrary to this, lower LEN(S) values mean that each code clone in a clone set S consists of less token sequences and the size of code fragments are smaller.

***c) Execution Time*:** The time it takes the tool to execute the code and find clones.

d) #clones*:* The number of clones a tool finds in a file.

e) Gaps*:* If there are any insertions or deletions in a code then it is called as gaps. Code example is as follows:

```
If (b < = c) {
d = e + c; //comment 1
e = e + 1;
else
d = e- b; //comment 2
```

code of this segment after adding one statement could be as follows:

```
if (b < = c) {
d = e + c; //comment 1
f = 2;
e = e + 1;
```

Metrics / Tool	Clones	Gaps	POP	LEN	Execution time
Solid SDD	9	2	386	80	1second

Table 1: Clone set metrics

Table 1, represents clone metrics measured by solidSDD tool showing the number of clones detected by tool in the project coding. Also type 3 clones i.e gapped clones detected are 2. Execution time taken by solid SDD tool is very less i.e. only 2 seconds which increases its efficiency by giving clone length 80.

Conclusion

Accurate clone detection results are important for both research and practice. we have evaluated and analysed clone detection tool i.e. solidSDD on the basis of various clone set metrics. As solidSDD take less time to find the clones, and show gaps also i.e. type 3 clones.A single tool is not fully efficient to detect code clones, like in my thesis SolidSDD is best

suited and find more number of clone . Therefore, the tools available basically depends upon the application, project and language.

Future scope

Further investigation can be done how we can make the tool more efficient to detect type-4 clones.

References

[1]. 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.

[2]. 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.

[3]. J.R. Cordy Comprehending reality: Practical challenges to software maintenance automation. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03)*, pp. 196206, Portland, Oregon, USA, May 2003.

[4]. Matthias Rieger. *E®ective Clone Detection Without Language Barriers*. Ph.D. Thesis, University of Bern, Switzerland, June 2005.

[5]. 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.

[6]. Jean Mayrand, Claude Leblanc, EttoreMerlo, “Experiment on the Automatic Detection of Function Clones Software System Using Metrics”, In Proceedings of Sannella, M. J. 1994 Constraint Satisfaction and Debugging for Interactive User Interfaces. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95 09398., University of Washington.

[7]. 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).