ISSN: 0976-3104
SPECIAL ISSUE: COMPUTER SCIENCE

ARTICLE

A DETAILED STUDY OF SOFTWARE CODE CLONING

Annu Vashisht[1], Akanksha Sukhija[2], Arpita Verma[3], Prateek Jain[4]*

[1,2,3] *Department of Computer Science and Engineering, Manav Rachna International Institute of Research and Studies, Faridabad, INDIA*

[4]*Accendere KMS Services Pvt. Ltd, New Delhi, INDIA*

ABSTRACT

Background: *Code cloning is one of latest area of research in software systems. Copying and pasting the code with or without modification is termed as code cloning. Code clone detection techniques which are concerned to find the code fragment that produce the same result. The issue of finding the duplicate code leads to different tools that detect the copied code fragments. In this paper we have discussed about the detailed study of the code cloning along with its types, benefits, advantages, drawbacks, clone detection process as well its techniques, tools for its detection. Further this paper also shows a typical comparison between the various techniques of the code clone detection.*

KEY WORDS
Software clone, Clone Detection, Semantic clones, Model based clones

Received: 5 Jan 2018
Accepted: 31 Jan 2018
Published: 24 Feb 2018

***Corresponding Author**
Email:
prateek.jain@accendere.co.in

INTRODUCTION

Software engineering (SE) is the application of engineering for development of software in a systematic method. Research, design, develop, and test operating systems-level software, compilers, and network distribution software for medical, industrial, military, communications, aerospace, business, scientific, and general computing applications. SDLC (Software Development Life Cycle) is sometimes also referred to as Application Development Life Cycle. It is basically a term used in system engineering, to copy the code and reused the code by doing some modifications or without doing some modification in the exiting code are common activities in software development is known as code cloning.

Developers are asked to reuse the existing code because of high risk in developing the new code. One of the major cause of code duplication is the time limit assigned to developers. In the software system copied code fragments and code clones are considered as bad smell of the software. It is observed that code clone has bad effect on the maintenance of the software system. To remove the clones from the software systems is quite beneficial. These clones are syntactically or semantically similar. It is very difficult to identify which code is copied code or which code is original. Several studies show that it is difficult to maintain software system which contains the code clones as compared to others which does not contain the clone.

Code Fragment: A code fragment (CF) is any sequence of code lines (with or without comments) and of any granularity, e.g., function definition, begin–end block, or sequence of statements.
Clone pair: If there is any clone relation exist in the pair of code fragments then it is called a clone pair or clone pair is a pair of code fragment having some similarity between them.
Clone set: A set of all the identical or similar fragments.
Clone class: A set of sall the clone pairs in which the existing clone pairs having some clone relationship between them is known as clone class.
Clone class family: The group of all the clone classes that have the same domain is termed as clone class family [4].

Cloning may increase the bug probability if some bug is found in the source code and that code is reused by copying and pasting then that bug is also found in that pasted code fragment. For fixing the, these code fragment should be detected. It is being shown in [Fig. 1].

Normal Reasons of code cloning:

There are various reasons for code duplication.

Reuse of code, logic and design is the main reason of code duplication: Sometimes there is a need to merge two similar system having similar functionalities to develop a new one which result duplication of code even both the system is developed by different teams

Time Limitations: Developers are asked to reuse the existing code because of high risk in developing the new code. One of the major cause of code duplication is the time limit assigned to developers. To complete a project some time limit is assigned to developers. Developers find the easy solutions of the problem due to time limit. They find the similar code related to their project. They just copy and paste the existing code

Development Strategy: Clone can be introduced in different systems other than software system due to the different reuse.

IIOAB JOURNAL
ISSN: 0976-3104

Reuse Approach: Reusability of code, logic, design and/or an entire system are the major reasons of code clone occurrence. Reusing code, logic, design and/or an entire system are the prime reasons of code duplication. Reusing existing code by copying and pasting is the simplest form of reuse mechanism in the development process. It is a rapid way of reusing reliable semantic and syntactic constructs.

Programmers limitations and time constraints: The software is written seldomly in an ideal condition. Limitations of the programmer's skills and the hard time constraints inhibit proper evolution of the software. Hence the copy pasting is the only solution left with the programmers [7].

Complexity of the system: The difficulty in understanding large systems is the utmost reason for copying the existing functionality and logic [7].

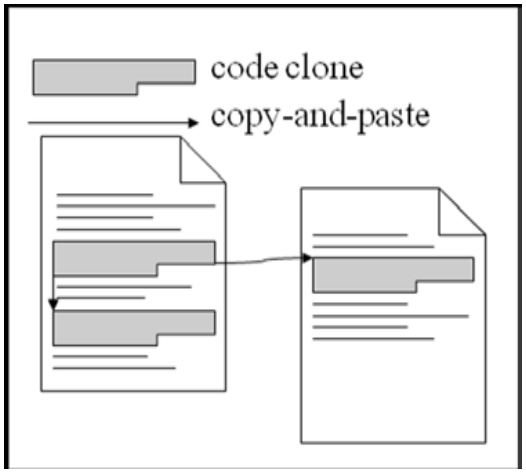

Fig .1: Code Clone [22]

Language Limitations: Kim et al. [7] conducted an ethnographic study on why programmers copy and paste code. Sometimes programmers are forced to copy and paste code due to limitations in programming languages. Many languages lack inherent support for code reuse, leading to duplication.

Specific reasons for cloning

Code clones don't occur itself in the software systems. There are various factors that influence the developers/ engineers in cloning the specific code in any system. Cones can also be accidently introduced in a system [36, 30, 37, 26, 33, 28, 38, 39]. The various factors have been shown in [Fig .1].

Development Strategy
Reusability and the programming approaches are also a vital reason for the occurrence of the code cloning.

Re-usability based approach
Reusability of code, logic, design and/or an entire system are the major reasons of code clone occurrence.

- **Simple Re-usability by copy/pasting:** The re-usability of the existing code by copying and pasting it with or without any modifications is the simplest means in the development cycle which is responsible for code cloning/duplication. It's an easiest way to re-use the semantic and syntactic constructs. The cross cutting concerns can also be introduced using this strategy [39].

- **Forking:** It means reusing the similar solutions with the hope that it will diverged significantly with the system's evolution. This term was used by Kasper & Godfrey [40]. For e.g. while creating a driver for the hardware family, a same hardware family may have a driver already and thus the same can be re-used after the slight changes into it. Similar to this the clones can be brought in during software porting to a new platform.

- **Design functionalities & logic re-usability:** The logics and other functionalities can also be re-used if a same sort of solution already exists for the same. There is high sort of similarity among the various ports/versions of a sub-system. Like that of the OS's subsystem, it is similar in the structure and functionality too with little been change/addition of features and functionalities. We can also say that the Linux kernel device drivers are also bound with more cloning/duplication [41] as all the drivers have the similar interface with mostly the simple logic. Moreover, the design of such systems does not allow for more sophisticated forms of reuse.

IIOAB JOURNAL
ISSN: 0976-3104

Programming approach:

The way the system is developed also plays a crucial role in introducing the errors. Few of them are as follows:

- **Merging of two systems based on similarity**: Sometimes the two software's of same functionalities and merged together so as to produce a new one. Although two different teams are involved in the development of these 2 systems, but it can lead to occurrence of clones in merged systems due to the implementation of similar functionalities in both the systems.

- **System development using generative approach in programming**: The generation of the code by the tool based on generative programming approach can also be responsible for producing the clones in good amount as these tools use the same template for the generation of similar logic.

- **Delay in code re-structuring**: The delay in the re-structuring of the code being developed by the developers is also responsible for introducing the clones in the code.

Maintenance benefits

Clone are also introduced in many systems to obtain several and important maintenance benefits. Examples are:

- **New code development risk**: Cordy [42] has reported that the reason for frequent occurrence of clones in the financial software is the updation/enhancements in the existing system for supporting the similar sorts of new functionalities. There are not much of the changes being observed in the financial products specifically within the same financial institution. The programmer is supposed to reuse the existing code by copying so as to adapt to the new requirements of the product. This is done so as to reduce the high risk of errors in new fragments and other major reason is that the existing code is already tested properly.

- **Clean and understandable software architecture:** The software clones are sometimes introduced intentionally so as to keep the software architecture clean and understandable [40].
- **Speed up maintenance**: As two cloned fragments are independent of each other in terms of syntax and semantics, hence it is possible to implement them at various paces without any effect on other clone. In this case testing needs to be finally done in the altered/modified fragment only.
- **Ensuring robustness in life-critical systems:** The clones/redundancy are intentionally introduced during the design of life-critical systems. More often the similar set of functionalities are developed by different teams so as to reduce the probability of implementation failure under various same circumstances.
- **High cost of function calls in real time programs**: Function calls may be deemed very costly in real time-based programs. The code is to be made inline manually if not made automatically by the compiler otherwise this may also lead to clones in the code.

Overcoming underlying limitations

Clones occur in the code due to the following limitations which consist of language limitation and programming limitations of the developers.

Language limitations

Clones may also occur due to the language drawbacks especially when the language doesn't have efficient abstraction mechanism. E.g.

- **Lack of reusability mechanism of programming languages:** Some programming languages lacks sufficient mechanism of abstraction, inheritance, generic types in C++ hence the code needs to reused from an existing one. This leads to the introduction of clones too.
- **Significant efforts in writing reusable code**: The writing of the code based on re-usability is a complex and time-consuming task. Perhaps it is much better to maintain 2 different fragments of code by cloning rather than producing a general code.
- **Reusable code writeup is error prone:**
  Code based on reusability mechanism might consist of errors. Hence it is preferred to copy paste the existing code after reusing the code with or without much changes/alteration.

Programmer's limitations

One more reason for the cloning is the drawbacks of the programmer's ability to write the code. Some of the examples are:

- **Difficulty in understanding large system**: Understanding of the code of the larger systems seems to be a cumbersome task for the programmers. Hence, they are forced to use the example-oriented programming by the adaption of already developed code.

(

IIOAB JOURNAL
ISSN: 0976-3104

- **Time limit assigned to developers**: Time frame is also a major cause of cloning for the programmers. In certain situations, developers are bound to complete the project in specific time frame hence they search for solving the same by an easier way. This easier way is none other than using the existing code.

•**Wrong method of measuring developer's productivity**: The productivity of the developer is sometimes predicted by the number of lines of code being produced by him. Hence the focus of the developer is to increase the number of lines in the code and reuse the existing code again and again by copy pasting. This is not done with the proper development strategy rather being done only for increasing the number of lines in the coding.

•**Lack of ownership of the code to be reused:** One main reason of code cloning is that the code is being borrowed from some other department and hence the same can't be modified by the developer due to not having its ownership. In these situations, copy pasting is the only thing being left to be done by the developer.

Advantages of code cloning

Various advantages of code cloning are as follows:

1. **Detects library candidates**- If a code fragments has been reused and copied various times shows its usability in system.
   Therefore, the fragment of code must have been integrated in the library for showing its potential officially.

2. **Helps in Understanding Program**-
   To have an overall understanding of other files containing other same content of that fragment, it is quite possible only if the functionality of the cloned fragment is being comprehended.

3. **Helps aspect mining search**- Code detection is a necessary aspect of mining to detect cross-cutting concerns. The code of cross-cutting concerns is typically cloned over whole application that could be detected with the help of code cloning detection tools.
4. **Finding patterns of usage**- If all the cloned fragments of the same source fragments are detected, then the functional usage patterns of cloned fragments can be discovered.
5. **Malicious software detection** - It is possible to find the evidence where a part of one software system can match pats of others, by comparing one malicious software to another. Clone detection can play a vital role in detection of malicious software's.
6. **Helps in code compacting**- By reducing source code size clone detection techniques can be used for compact devices.
7. **Plagiarism and copyright infringement detection**– In detection of plagiarism and copyright infringement finding same sort of code may also be useful.

1.4 Drawbacks of Code Cloning

Code clones have bad impact on the maintainability, reusability and quality of the software. If there is any code segment present in the software which having a bug and the code segment is copied and pasted anywhere in the system then the bug is remains in all the pasted code segment which is difficult to maintain. When duplicated code used in the system it may lead to bad design which increase the cost of the system. If in the software system there is duplicated code, to understand the system additional time needed. It becomes difficult to upgrade the system or even to change the existing one.

1. **Increase probability of bug propagation**_– If a segment of code contains a bug so that segment can be reused by copy and pasting with or without minor alterations. The bug of an original segment may be present in all entire pasted segments in a system. Hence, the bug propagation probability may rise up significantly in the system.

2. **Increased graph of bad design** - Code Cloning also tends to make the design bad, lack of good inheritance structure or abstraction. Therefore, it becomes very difficult to reuse implementation part for the future tasks. It also has a very negative impact on the software maintenance activity.

3. **Increased cost of maintenance** - If the clone consists of any bug, so all of its same counterparts need to be properly checked for the correction of bug as cloning don't guarantee removal of bug in other codes during reusability or maintenance.

ISSN: 0976-3104

Fig .1: Reasons for code cloning [2]

EARLIER WORK DONE

According to the state of the art in clone detection research [10] several tools are available to detect Type 1 and Type 2 clones but more empirical study is required to derive the classification for Type3 and Type 4 clones. This is the base paper of our research work and we have considered here only the research study done on code clones of Type 3 and Type4. For Type 3 clone detection, Tiarks et.al [11] have proposed a study of the current state of the art. In his study, he use the Levenshtein distance to compute a clones that falls in two subcategories: structure substituted clone and modified clone. Yue Jia et al. [12] has developed a detection tool named as KClone, which is based on the implementation mechanism with the hybrid approach (specifically the token & PDG based techniques). For example, the algorithm which quickly detect Type-3 clones are normally only detected by means of slow, semantic, clone detection techniques etc. This tool developed in C programming and is useful in finding clones in C, C++, and Java programs. KClone pre-helps in processing all the files prior to running firstly the syntactic and then analysis based on dependence. They showed that KClone can detect clones more rapidly with respect to CCFinderX and Duplix. Yang Yuan et al [13] introduce Count Matrix based techniques to detect in clones in program codes. This technique works well/hard for detecting the clones. It is language independent as it is based on the variable counts. The source code is splitted into the classes during the time of the processing. For each and every method it is possible to obtain the matrix based on counts. We can also construct a bipartite graph for two methods, and the same matching on the graph. The similarity among the two methods are closely related to the size of the matching. The same method can also be used to find the similarity among the classes. A false positive elimination step is perform after matching to eliminate some obvious false positive cases based on heuristics. The authors have taken into consideration the all code clones of Type I, Type II, Type III and Type IV. This algorithm is responsible in successful detection of all

IIOAB JOURNAL
ISSN: 0976-3104

types of clones except the type IV clones which is difficult to detect as its approach doesn't consider the control loops. Yoshika Higo et.al [14] has proposed a code clone detection technique based on PDG incremental approach for detecting the code which is non-contiguous. The methodology as proposed is developed as a tool of prototype which is useful in efficiently obtaining the code clones with shorter time spam. The speed of detection is better than that of the KClone. E.Kodhai et. al. [15] proposes a hybrid approach of textual and metric analysis to detect all types of clones in java source code only. The process of clone detection has divided into different phases. First the input is selected and the parsing is applied to detect Type I clones. Secondly template conversion is done to detect type II clones. Thirdly the metrics method is applied to detect all types of clones. D. E. Krutz et.al [16] has proposed a new code clone detection technique which is based on concolic analysis, which uses a mixture of concrete and symbolic values for traversal of large and diverse portion of the source code. By this analysis on the target source code and by the examination of the holistic output for similarities, code clone candidates can be consistently identified. The author has founded that this technique was able to accurately and reliably discover all categories of code clones. This technique is based on small C & JAVA programs. It is one from few known processes which is able to detect Type IV clones. Ripon K. Saha et. al. [17] have discussed about the concrete data on the Type 3 clones evaluation in very different settings than the previous studies and have drawn various broad conclusions about their changing patterns, frequency, type conversions, and lifetime. As per the findings it is very important to manage the Type 3 clones very intelligently due to their more inconsistent nature. They have discussed about the Type 1,2,3 clones and have analysed their evolution independently by using the different clone detection tool and gCad extractor. Based on the study results, the researcher suggests several approaches for dealing with Type III clones which would be helpful for designing a robust clone management system. H. Kim et.al [18] proposed an abstract memory-based code clone detection technique, with its implementation as a tool MeCC, and discussed its applications. Their experimental study shows that MeCC can accurately detect all four types of code clones. The only limitation is that MeCC detects only procedure level clones.

D. E. Krutz et.al [19] presented CCCD, a tool which uses concolic analysis to discover code clones. It is very effective in discovering clones of all four types. The disadvantage is that Only C programs are compatible with CCCD since CREST is only capable of analyzing C code. S. Bazrafshan [20] studied to analyses the evolution of two clone classes throughout three versions of a software system. According to the researchers findings near miss clones require more concentration during maintenance. The researcher proposed two different approaches. First the author analyzed the evolution of type II and type III clones together as near miss clones which is hard to understand the unique behavior of type III clones. Secondly, researchers compare the results to existing studies of the late propagation patrons with type I and type II clones and assumed to be their result valid and draw conclusions regarding differences of late propagation in identical and near miss clones.

CODE CLONING TYPES

There are two types of code cloning which are basically based on similarities that are "Textual similarity" and "Functional similarity" which are further categorized in four types which are Type I, Type II, Type III which are textual similarities whereas Type IV is functional similarity.

3.1 Textual Similarity - The textual or program based similarity means that the code fragment are similar to each other on the basis program text. These are further classified in three types-

- Type I (exact clones)- In this similarity, the code fragments similar to each other but there is a variation in blank spaces, comments or layouts. For further explanations let us consider an example –

  Original fragment –

```
int n;
cout<<" ENTER THE NUMBER";
cin<<n;
```

  Copy fragment –

```
int n;
cout<<" Enter the name";
cin>>n;
```

The original and copied fragments are similar if we remove spaces.

- Type II(renamed/parameterized clones) – In this similarity, the code fragments which are copied from original fragments are similar. However there can be a difference in variations in the literals, variables, constants, class, types, layout and comments. The syntactic structures of both the code segments are same. Let us consider am example-

  Original fragment-

```
int  number;
cout<<"Enter the number";
cin>>number;
```

  Copy Clone-

ISSN: 0976-3104

```
int n;
cout<<"Enter the number";
cin>>n;
```

Both original fragment and copy clone would be similar if we name both the variable number.

- Type III(near miss clones)- In this type, by adding or changing some statements the copied code fragment can be modified. For further details let us take an example-

```
Original fragment –
Total= phy+chem.+math;
per=Total/3;
if(per>=70)
{
cout<<"First division";
}
else
cout<<"Second division";
Copy clone –
total=phy+chem+math;
per=total/3;
if(per>=70)
{
cout<<"first division";
cout<<"Excellent";
}
Else
{
cout<<"Second division";
}
```

The original fragment and copy clone are similar only a statement is added in the copy clone to modify it.

Functional Similarity-Functional clones are also known as semantic clones it means that the codes fragments are functionally similar. Type IV are semantic clones.

- Type IV (semantic clones) – In this type of cloning, it is not necessary that the codes are textually similar but there functionality is similar and it is not that the codes are copied from each other. Two code fragments may be developed by the different teams but they perform same computation. Code fragments are similar in their functionality because different teams implement the same logic. Let us consider the following code fragment 1 and code fragment 2 where the swapping of two variables done.

Fragment 1:

```
int a=5, b=10 , temp ;
 temp=a;
a=b;
b=temp;
Fragment 2:
int a=5,b=10;
a=a+b;
```

In fragment 1 swapping is done using three variables whereas in fragment 2 swapping is done using two variables. Here both the fragments are similar from semantic or functional point of view.

Code clone description

Type	Description
Type 1	Exact clones where a copied code fragment is similar to the original code fragment except whitespaces and comments
Type 2	Renamed clones where copied code fragment is identical to the original code fragment but modifications in names of variables, functions
Type 3	A copied code fragment is modified by changing the structure of the original code fragment, i.e. .by adding or removing some statements.
Type 4	Clones have semantic similarity between code fragments i.e. they implemented by different logic but are similar in their functionalities

IIOAB JOURNAL
ISSN: 0976-3104

PROCESS OF CLONE DETECTION

The role of the clone detector is to find the pieces of the code which is having the high amount of same source text in the system. The main issue is that we don't know in advance about the code fragments which can be found out more than once in a specific code. Hence the detector needs the comparison among the various fragments of code. This way is very expensive in terms of the cost of the computation. Thus, various alternatives are being adopted for the reduction of the domain of comparison before the actual comparison. Once we find out the potential fragments of code, further there is a requirement of analysis and/or tool support for detection of actual clones. We are trying to present the typical process of the clone detection. It consists of various phases which are discussed as below:

Pre-processing phase
While starting the clone detection process the first thing is to partition the targeted source code & determining the comparison domain. This phase constitutes to 3 main tasks:

- **Removal of un-interesting parts**: All the source codes which are un-interesting to the comparison phase is altered first. E.g., we apply the concept of partitioning to the embedded code which includes SQL embedment in Java code or Assembler in C code. This task is being conducted for the separation of the different languages specifically if the method is language dependent.

- **Determining Source Units**: After the removal of the un-interesting code, the left code is being distributed/partitioned among various set of disjoint fragments which are also involved in the direct clone relations among each other. These units are not responsible for any sort of order maintenance in the source code and hence, the units which are similar to it can't be aggregated beyond the border of those source units.

- **Determining comparison unit/granularity**: After the second phase the source code has to be further partitioned among various smaller units which is dependent on the comparison function of a method/function. E.g. the source units can also be sub-divided into the lives or even the tokens fin order to do the comparison. The same can also be derived using the syntactic structure of the source unit. The comparison units are being ordered within among the corresponding source units. This sort of ordering is very crucial for the comparison function.

Transformation *phase*: The units for comparing the source code are trans-structured into other intermediate internal form of representation in order to compare and extract the comparable properties/features. This sort of transforming into other may also vary from simple to very complex where simple one includes only removing the white spaces and comments while the complex one includes only generation of PDG representation [23,26]. The methods based on the metrics contributes to an attribute vector for every unit comparison for intermediate representing the same. We have discussed about the various approaches of the transformation as one or more techniques may be used for an algorithm based on comparison.

- **Pretty source code printing**: The re-organization of the source code into a standard form can be performed using this approach of transformation. It tends to transform the source code of different layout into a common standardized format. It's being commonly used by text based clone detection approach. It helps in the avoidance of the false positives which can occur as a result of different layouts with the same code segments.

- **Removal of comments**: There are various approaches which either ignore or remove the source code comments prior to actual comparison [27,28].

- **Removal of whitespace**: Every technique discards the whitespace except for the line-based approaches. Other approaches use pattern based on indentation based on pretty printed source text as being the attribute vector feature [29]. While other approaches may also work on the basis of the layout metrics including the quantity/number of the lines which are blank [28].

- **Tokenization**: In tokenization based techniques, every line of source is being divided into the tokens corresponding to a rule based on the lexical analysis of programming language with interest. The tokens obtained corresponding to all the lines are then used for the token sequence formation. Each and every whitespace which includes the line breaks, tabs, comments among the tokens are being removed from the sequencing of the tokens. The same can obtained the CCFinder and Dup tools.

- **Parsing**: In case of the approaches based on the parse tree, the entire source code is being parsed for building of the parse tree in an automated way or an AST. In such representation mechanism the source as well as the units for comparison are being represented as being the parse tree or the AST subtrees [30,31,32]. The comparison algorithm then uses these subtree for finding the clones. The approaches based on the metrics uses these code representations of code for the calculation of the subtrees and clone finding on the basis of the metrics values [33, 28].

IIOAB JOURNAL
ISSN: 0976-3104

- **Generating PDG**: The techniques based on semantics aware generates the program dependence graph (PDGs) by the help of the source code. Source or the comparison units tends to be the subgraphs for these PDGs. For finding of the clones the detection algorithm further looks for the isomorphic [24,25]. Some approaches based on the metrics also uses various sub-graphs for forming the data metrics and the control flow metrics which can further be utilized for searching of the clones [28,33].

- **Normalizing identifiers**: Many approaches apply the normalization of the identifier before going through with the comparison phase. All the source code identifiers are being replaced with a single token in such normalization mechanisms.

  - **Transformation of program elements**: In addition to the normalization of an identifier, there are various other transformation rules which may be applied on various elements of the source code as per the needs and the requirements.

  - **Calculate metrics values**: This applies with calculating the value of the metrics on the basis of the outcomes of the preceding phases.

Match Detection: The code once transformed using the earlier phases is then inputted to an appropriate comparison-based algorithm which includes the comparison of the transformed unit among each other in order to find the matching among the clones. By the help of order of the unit's comparison, the similar adjacent units are being merged together so as to form a much larger unit. In case of the clones with fixed granularity, every unit of comparison belonging to a source unit are then aggregated. It is then continued for the clones based on the free granularity. While the aggregation is continued until the aggregated summation is more than the given threshold for the number of aggregated units of comparison. This assures the performing of the aggregation till the largest possible groups of the comparison of units are found therein. The output of the phase is the list of matches w.r.t the transformation of the code. All of the similarity earlier exist in the clone pair candidates or the same need to be aggregated so as to form the same. Each of this pair of clone is being represented using the information about the location of source code portitions being similar in the transformed code [34,35].

Formatting: The list of clone pair as obtained in the last phase w.r.t the transformation of the code is now converted into the list of clone pair w.r.t the original code base. In normal, each pair of location of clone being obtained in the earlier phase is now being converted into the line numbers on the basis of the original source code files. It is done using the common format for the representation of the pair of the clone which includes the nested-tuple.

Post-processing: This phase is concerned with filtering the false positive clones with the help of manual analysis and/or a visualization tool. It consists of 2 parts:

- **Manual Analysis:** This phase constitutes to filter out the false positive clones.

- **Visualization**: The clone pair list as obtained previously is used for the visualization of the clones by the help of this tool. It can also help in speeding up the process of manual in order to remove the false positives and/or other associated analysis.

Aggregation: For the reduction of the quantity of data or performing the various analysis, the clone pairs are being aggregated with the clusters, classes, cliques of clones or in the group of clones respectively. The all above phases as discussed for the clone detection process are very general and can thereby overlooked in a given detection process.

Clone Detection Techniques

Text-based Techniques*:* In the text-based technique the source code fragment is assumed as sequence of line. After removing the various comments, whitespace by applying the various transformations the code fragment is compared with each other. Once the two code fragments are found to similar to each other to some extent they are known as clone pair or clone pairs form the clone class. Sometimes in the clone detection process the source code is directly used. Text based technique is efficient technique but it can detect only Type I clones. Text based approach cannot detect the structural type of clone having the same logic but different coding.

Token-based Techniques*:* In the token-based technique, first sequence of tokens is generated from the source code. For converting the source code into tokens, it requires a lexer. Lexer convert the source code into tokens then the various transformation is performed by adding, changing or deleting some tokens. For finding the duplicated code or duplicated subsequence of token the sequence is scanned and the code portions representing the duplicated code returned as clones. Token based technique can detect Type I, Type II clone.

IIOAB JOURNAL
ISSN: 0976-3104

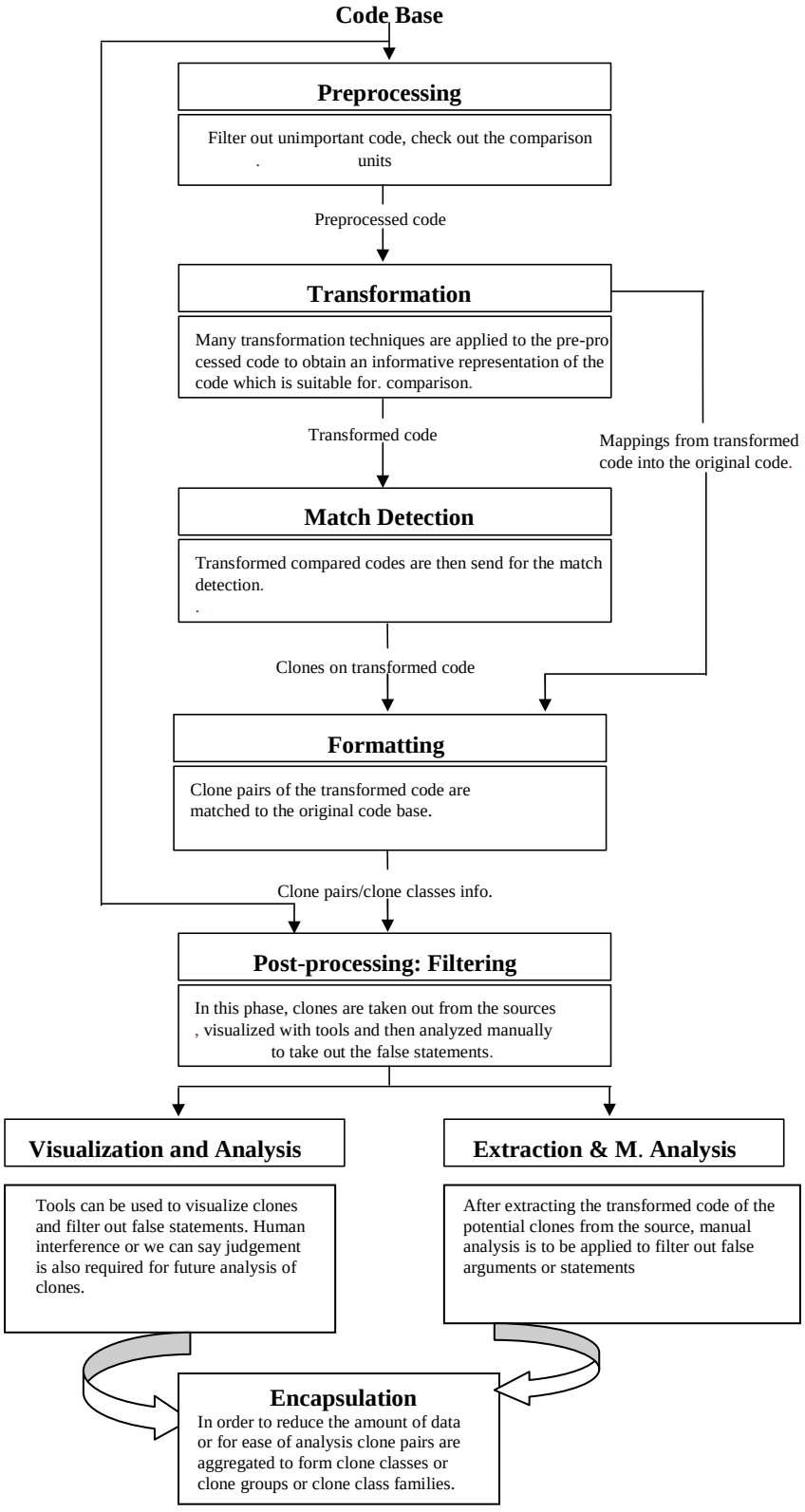

Fig. 2: Clone Detection Process.

Tree-based Techniques: This technique creates sub trees rather than creating tokens from each statement. The code then said to be code clone if the sub trees match. With the help of parser of a language similar sub trees are searched in the tree using tree matching algorithm or structural metrics then the code of similar sub trees is returned as clone pairs. Abstract syntax tree has the complete information about the code. The result obtained from this technique is quite efficient but to create an abstract syntax tree is difficult for a large software and the scalability is also not good.

IIOAB JOURNAL
ISSN: 0976-3104

PDG-based Techniques: Program Dependency Graph (PDG) technique is more efficient then tree based technique. Program dependency graph show data flow and control flow information. First the program dependency graph is obtained from the source code then to find the similar sub graphs or clones several types of sub graph matching algorithm are applied and returned as clones. This technique can detect both semantic and syntactic clones but in case of large software to obtain the program dependency graph is very difficult.

Metrics-based Techniques: In Metrics based Technique first different types of metrics of the code like number of lines and number of functions are calculated and compare these metrics to find the clones. Metrics based technique does not compare code directly. To find the code clones several type of software metrics are used by clone detection techniques. Most of the time, for calculating the various type of metrics the source code is converted into abstract syntax tree or program data graph. Metrics are calculated from the name, layout, control flow and expression of the functions.

CODE CLONE DETECTION TOOLS

Baker's *Dup* represents the source code as sequence of lines and detects the clones in the code fragment line-by-line. Baker's uses a line-based string matching algorithm or lexer on the individual lines. First Dup tool removes comments and white space from the source code and then it replaces various identifiers, variables and types with a special parameter so that if the name of the two variables is different clone can be identified. Baker's Dup tool cannot detect the clones if the source code is written in different style.

. *CCFinderx* is one of the tool of the token-based techniques. CCFinderx find the clones both within the files or from various files from programs and find the location of the clones in the program. First, tokens are generated from the source code and then the single token sequence is formed by concatenating all the tokens. Various transformations are applied on the token sequences based on the transformation rules.

One of the important tools of metrics-based techniques is *Covet/CLA* to detect the clones using metrics. Mayrand et al. calculate various type of metrics for each function unit of a program like number of CFG edges, lines of source code, number of function calls etc. Code fragments which have similar metrics values are known as code clones. Covet/CLA does not detect the partly similar codes.

One of the important program dependency graph based clone detection approach is that of *Komondoor* and Horwitz's PDG-DUP which identify isomorphic program dependency sub graphs using program slicing.

CloneDR is one of the tools of the abstract syntax tree based clone techniques. Compiler is used to generate AST or abstract syntax tree and the complier compares the sub trees, the sub trees which are similar are returned as clones.

There are various code clone detection techniques which are Text based, Token based, Tree based, Metrics based and PDG based.

- Text based clone detection techniques transform code by removing whitespaces and comments. Its complexity depends on Algorithm. It is only good for Similar for exact matches. It compares the two fragments by tokens of line. It only detects Type I clones.
- Token based clone detection techniques transform code by generating a token from original fragment. Its complexity is linear. It needs some post processing for clone detection. It compares the fragments on basis of tokens. It detects both Type I and Type II clone.
- Tree based clone detection techniques transform the fragments by generating ASR from source code. Its complexity is quadratic. It is able to find syntactic clones. It compares codes by using nodes of tree. It detects all the types of Text or syntax based clones.
- Metrics based clone detection transforms code fragments by generating AST from source code to find metrics. Its complexity is quadratic. It also Detects all syntax based clones.
- PDG based code clone techniques transform code fragments by generating PDG (Program Dependency Graph). Its complexity is Quadratic. In this some manual instructions are also required. It detects Types IV clones.

Text based clone detection techniques are easily adaptable. In token based clone detection techniques lexer in needed. In PDG detection techniques syntactic knowledge of edge and PDG is required, whereas in both Tree based and Metrics based detection techniques Parser is required.

CONCLUSION

Code cloning is considered as bad practise for copying the existing code for the formation of a new code. It involves various types of the code cloning such as Type 1, type 2, Type 3 and Type 4. Type 1,2,3 code cloning mechanism works with the textual content. Type 4 code cloning works with the functional format and some part of type 3 is also being considered in the functional category. There are various code clone detection techniques at present which are text based, tree based, token based, PDG based And Metrics based clone detection techniques. The most used Code clone detection tools are Baker's Dup and

IIOAB JOURNAL
ISSN: 0976-3104

CloneDR. Text based Clone detection technique only detect type I clones. Token based detection techniques detect Type I and Type II clones. Tree based and Metrics based detection techniques detect all text or syntax based clones. Whereas PDG detection techniques detect type IV clones. The existence of code clones in a program enhancement is conservation cost as their existence makes the execution program complex and generates the issue of redundancy. The study of prior research work suggests the major focus of their research work on implementation approaches for detection of identified clones.

Table 1. Comparison between various Code clone detection techniques

Properties	Text Based	Token Based			
Transformation	Removes whitespace and comments	Token is generated from the source code	ASR is generated from the source code	PDG is generated from the source code	To find metrics values AST is generated from the source code
Representation	Normalized source code	In the form of tokens	Represent in the form of abstract syntax tree	Set of programs of dependency graph	Set of metrics values
Comparison Based	Tokens of line	Token	Node of tree	Node of program dependency graph	Metrics values
Computational Complexity	Depends on algorithm	Linear	Quadratic	Quadratic	Linear
Refactoring Opportunities	Good for exact matches	Some post processing needed	It is good for refactoring because to find syntactic clones	Good for refreshing	Manual inspection is required
Language in dependency	Easily adaptable	It needs a lexer but there is no syntactic knowledge required	Parser is required	Syntactic knowledge of edge and PDG is required	Parser is required

CONFLICT OF INTEREST
There is no conflict of interest.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to Dr. Prateek Jain, Accendere Knowledge Management Services Pvt. Ltd., for his valuable comments that led to substantial improvements on an earlier version of this manuscript

FINANCIAL DISCLOSURE
None

REFERENCES

[1] Chanchal K. Roya, James R, Cordya Rainer Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach", Science of Computer Programming 74 (2009) :470–449

[2] Roy, Chanchal Kumar, and James R. Cordy. "A survey on software clone detection research." Queen's School of Computing TR 541.115 (2007): 64-68.

[3] Kuldeep Kaur, Dr. Raman Maini," A Comprehensive Review of Code Clone Detection Techniques", IJLTEMAS, volume iv, Issue XII, December 2015

[4] Zhen Ming Jiang, Ahmed E. Hassan, and Richard C. Holt. Visualizing Clone Cohesion and Coupling. In Proceedings of the 13th Asia Pacific Software Engineering Conference (APSEC'06), pp. 467-476, Bangalore, India, December 2006.

[5] Md. Monzur Morshed, Md. Arifur Rahman, Salah Uddin Ahmed, "A Literature Review of Code Clone Analysis to Improve Software Maintenance Process"

[6] M. Fowler, "Refactoring: improving the design of existing code, Addison Wesley", 1999.

[7] M. Kim, L. Bergman, T. Lau, D. 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), Redondo Beach, CA, USA, 2004, pp. 83–92.

[8] L. Jiang, Z. Su, E. Chiu, Context-based detection of clone-related bugs, in: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'07), Dubrovnik, Croatia, 2007, pp. 55–64.

[9] C. Domann, E. Juergens, J. Streit, The curse of copy & paste-cloning in requirements specifications, in: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, Florida, USA, 2009, pp. 443–446.

[10] C.K Roy, J.R Cordy, Rainer Koschke, Comparison and evaluation of code clone detection techniques and tools: A qualitative approach Science of Computer Programming, pp. 470-495, 2009.

[11] Tiarks et.al. An Assessment of Type 3 clones as detected by State-of-the Art Tools. In proceedings of the ninth International Working Conference on Source Code Analysis and Manipulation. IEEE Computer Society 2009. pp 67-76

[12] Yue Jia, David Binkley et.al. KClone: A Proposed Approach to Fast Precise Code Clone Detection, Proc. of Third International Conference on Software Clones, Number 740, March 2009.

[13] Y. Yuan and Y. Guo., CMCD: Count Matrix based Code Clone Detection, APSEC, IEEE Computer Society, pp.250-257, 2011.

[14] Higo. Yoshiki, et al. Incremental code clone detection: A PDG-based approach. Reverse Engineering (WCRE), 2011 18th Working Conference on. IEEE, 2011.

[15] Kodhai. E, Perumal. A and Kanmani. S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones," in IJCCIS, Vol2 (1). ISSN: 0976–1349 July – Dec 2010.

[16] D. E. Krutz et.al, Examining the Effectiveness of Using Concolic Analysis to Detect Code Clones, Proceedings of the 30th Annual ACM Symposium on Applied Computing, New York, pp.1610-1615, 2015.

[17] Ripon K.Saha et.al, Understanding the Evolution of Type-3 Clones: An Exploratory Study, In Proceedings of the !0th Working Conference on Mining Software Repositories, San Francisco, California, USA,IEEE, pp.139-148, May 2013.

[18] H. Kim et.al, MeCC: Memory comparison-based clone detector. In Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, New York, pp. 301–310, 2011.

[19] D. E. Krutz et.al, CCCD: Concolic code clone detection, 20th Working Conference on Reverse Engineering (WCRE), IEEE Computer Society, pp. 489-490, 2013.

[20] S. Bazrafshan, Evolution of Near-Miss Clones, In Proceedings of IEEE 12th International Working Conference on Source Code Analysis and Manipulation, pp74-83, 2012.

[21] H.Murakami et.al, Gapped code clone detection with lightweight source code analysis, ICPC, IEEE Computer Society, page 93-102, 2013.

[22] Katsuro Inoue, "Code Clone Analysis and Its Application", Software Engineering Lab, Osaka University.

[23] Brenda S. Baker. A Program for Identifying Duplicated Code. In Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.

[24] Raghavan Komondoor and Susan Horwitz. Tool demonstration: Finding duplicated code using program dependences. In Proceedings of the European Symposium on programming (ESOP'01), Vol. LNCS 2028, pp. 383386, Genova, Italy, April 2001.

[25] 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.

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

[27] Andrian Marcus and Jonathan I. Maletic. Identification of high-level concept clones in source code.In Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01), pp. 107-114, San Diego, CA, USA, November 2001.

[28] 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.

[29] 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

[30] 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.

[31] V. Wahler, D. Seipel, Jurgen Wol von Gudenberg, and G. Fischer. Clone detection in source code by frequent itemset techniques. In Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM'04), pp. 128135, Chicago, IL, USA, September 2004.

[32] Wuu Yang. Identifying syntactic differences between two programs. In Software Practice and Experience, 21(7):739755, July 1991.

[33] K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M. Bernstein. Pattern Matching for Clone and Concept Detection. In Automated Software Engineering, Vol. 3(1-2):77-108, June 1996.

[34] S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees. In In Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS95), pp. 631638, October 1995.

[35] E. Mc Creight. A space-economical suffix tree construction algorithm. In Journal of the ACM, 32(2):262272, April 1976.

[36] 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.

[37] 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.

[38] Matthias Rieger. Effective Clone Detection Without Language Barriers. Ph.D. Thesis, University of Bern, Switzerland, June 2005.

[39] 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.

[40] 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.

[41] M.W. Godfrey, D. Svetinovic, and Q. Tu. Evolution, growth, and cloning in Linux: A case study. In CASCON workshop on Detecting duplicated and near duplicated structures in large software systems: Methods and applications, October 2000.

[42] 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.