Review of Literature for Code Duplication and Refactoring of Code Clones

Dr.G.Anil Kumar
Sr. Assistant Professor CSE MGIT Hyderabad T.S. India

Abstract
Code duplication popularly known as code cloning is an act of reusing the code by cut, copy and paste. Code cloning research was carried out by many researchers throughout the world. This paper discusses some of the research contributions made by different scholars in this area to understand the direction of research work.

I. CODE CLONE LITERATURE

Research in Code cloning is growing steadily from 1990s onwards. So, for the past two decades lot of research has been done in the area of code cloning. In this paper we are presenting some of the back ground work and literature that was encouraged us to take up this work as our research area.

A. Code Clone Terminology

Usually there are only four types of clones. However, people use different terms when they refer to the clone relation for their experiments. The term exact clone is used when they refer to the identical code fragments. The term near-miss clone is used when they refer to code fragments which are identical with statements added, deleted or modified.

a. Exact Clones

If two or more code fragments are similar to each other except with some differences in white spaces, comments or layout, they are called exact clones. Editing can be done in the copied fragment. There are many editing activities such as changing the comments (i.e. changing the line, position etc of the comments) restructuring in layout like changing the position of begin & end brackets, removing or adding tabs, or changing language constructs like new lines, blanks etc., Sometimes the usage of Line-based methods may not enable the detection of the exact clones that are edited through addition or removal of new lines. This happens in changing the position of language elements. The exact clones are generally called as Type I Clones.

b. Renamed Clones

The term Renamed Clones generally used by people when comments, whitespace, literal values or identifier names changed in the coped code fragments. That is the reason why a renamed clone is necessarily as Type II Clone which is a parameterized clone. All parameterized clones are renamed clones but not all renamed clones are parameterized clones. Consistent renaming which is not essentially required in renamed clones case is a necessity in the parameterized clones.

c. Parameterized clones

A renamed clone with systematic renaming is called as a parameterized clone or P-match clone. The clone detector identifies consistent name matching instead of normalizing all identifiers or literals to some special symbols. These clones are part of or a subset of Type II Clones.

d. Near-Miss Clones

In clones if the copied fragments are very much similar to the original code fragment, such clones are called near-miss clones. In near-miss clones editing is applied. This editing include activities like changes in comments, layouts, changes in the position of the source code elements by inserting or removing blanks and new lines, changes in the identifiers, literals, macros etc. All these editing activities indicate that all renamed and parameterized clones are near-miss clones. In near-miss clones, a copied fragment is not an exact copy of the original because of slight changes. However, the syntactical structure still remains the same as its original. All the near-miss clones are Type II Clones. Most of the authors also assume that a minor modification in a statement or even addition or deletion of statement in the copied fragment will not bring any difference from the original. Therefore, all such clones are near-miss clones. This proves that Type III Clones can also be called as near-miss clones.

e. Functional Clones

The clones which are restricted to refer to complete functions or procedures are known as function clones. A subset of structural clones is function clones. Depending on the similarity level, function clones can be of the four types of clones, as is the case with structural clones. Generally, functional clones fall under the category of Type IV clones.

f. Clone Pair

When two or more code fragments are similar to the maximum possible extent they are known as clone pair (CP). Hence, pair of code portions or fragments that are similar and identical to each other. These clone pairs are critical to assess the efficiency of a clone detection tool through precision and recall values.

g. Clone Cluster

All the clone pairs identified in a system have code portions. The union of all the code portions common in those clone pairs is known as clone cluster (CC). These clone clusters are used for analysis and refactoring process in the later phases of clone detection process.

B. Clone detection and its application

Apart from the immediate applications of the clone detection techniques to code clone refactoring process, clone avoidance and management, there are various other domains where clone detection techniques are going to help. There are other areas which are related to clone detection through which clone detection techniques can be benefitted from [17].

To find malicious software, clone detection is useful. It is easy to find the matched parts of one software system with another one, by making comparison between malicious software with other similar kind of software.

Some of the applications of clone detection are as follows:

a. Plagiarism detection in projects

In clone detection one of the closely related areas is plagiarism detection [47]. These detection techniques could be utilized in the domain of detecting plagiarism. Token –based CCFinder[48] which is a clone detection tool is most commonly used in the detection of plagiarism.

b. Copyright infringement

The detection of source code copy right infringement is a problem. This is also viewed as a code similarity measuring problem between software systems. Clone detection tools might be applied or can easily be executed in detecting copyright infringement [49]. It is viewed as a serious problem in this competitive world, where cyber laws are becoming much stronger to protect copyright laws. This may happen with many reasons like by accident, or by using same logic by different programmers etc.

c. Clone detection in models

Clone detection can also be used for models [50]. In every similar way, phenomenon occurs in models. However, this is not restricted to code. Just as model clones are detrimental to code quality. For data flow model, general model, UML domain model clone detection can be utilized.

C. Different Clone detection techniques

Various clone detection tools are available in the clone literature. All these clones can be categorized depending upon the method they used to detect the clones presented in the code. Following sections explain these categories.

a. Text based tools

In the clone detection process, before the actual comparison is made, textual approaches use very littleor no normalization/transformation on the source code. The raw source code is utilized directly in most of the cases. The person who pioneered text–based clone detection is Johnson. His approach utilizes “finger prints” on the substrings of the source code statements [18].

First step is a fixed number of lines of code called window is hashed. A sliding window technique along with an incremental hash function is used. This is done to identify the sequences of lines which have the same hash value as clones. The sliding window technique will be applied repeatedly with different lengths in order to find clones of different lengths. Manber also used fingerprints depending on the subsequences identified by leading keywords [62]. This is done to trace out files which are similar.

Ducasse et al.[51] method is considered to be one of the recent text–based clone detection approaches. This approach is based on dot plots. The dot plot is also called as scatter plot. This is a two dimensional chart where source entities are listed by both X-axis and Y-axis. The comparison entities are the lines of a program in the approach proposed by Ducasse. If x and y coordinate values re equal, there is a dot at the co-ordinate (x, y). There must be the same hash value for two lines if they have to be considered equal to visualize clone information dot plots can be utilized.

Here, the clones are recognized as diagonals in dot plots. Normalization is done to ignore white space and comments. The identification of clones in dot plots can also be automated. Ducasse et al. used string based dynamic pattern matching which applies on dot plots to compute total lines. The gaps with diagonals indicate possible type 3 clones.

Wettel and Marinescu [52] used another approach to locate near miss clones using the dot plots method. This approach is an extension of the Ducasse et al. approach. Beginning with the lines which have the same hash value, the algorithm chains neighboring lines together are used to identify various kinds of type 3 clones. Another approach which is similar and which applies an n-neighbor approach in locating near miss clones is SDD [63].

NICAD is another approach which is also text-based [55]. This is necessarily a hybrid technique. However, this approach exploits the uses of tree based structural analysis which is based on a light weight parsing technique in order to implement flexible pretty printing normalization of code, source code transformation and code filtering.

Marcus and Maletic [53] have applied Latent Semantic Indexing (LSI) to source text to locate high level conceptual clones like Abstract Data Types (ADT) in the source code. This information retrieval approach restricts its comparison to identifiers and comments, two code fragments will be returned as potential clones or cluster of some potential clones, if there is high degree of similarity between the sets of comments and identifiers.

b. Token based tools

By using compiler style lexical analysis, lexical approaches start to transform the source code into a series of lexical tokens [48]. The corresponding original source code is returned as clones after the sequence is scanned for subsequences of tokens which are duplicated. The lexical approaches are usually robust compare to minor code changes like spacing, formatting and renaming than other textual techniques.

Brenda Baker [54] is the researcher who pioneered efficient token based clone detection. The lines of source files are primarily divided into tokens by a lexical analyzer in Baker's tool Dup. The tokens are divided into parameter tokens and non-parameter tokens. The non-parameter tokens are encoded using a hashing function whereas the parameter-tokens are encoded utilizing a position index for the tokens occurrence in the line. These encoding abstracts which are totally away from concrete names and parameter values, but not in the order, which allows continuously the parameters substituted parameterized clones which are also known as Type 2 clones to be found.

The prefixes of the discovered sequence of symbols are represented by a suffix tree, where the same set of edges has a common prefix. If there is a common prefix for two suffixes, the prefix occurs clearly more than once and this can be regarded as a clone. This technique permits one to trace Type 1 and Type 2 clones, and Type 3 clones can be traced by concatenating Type 1 and Type 2 clones if the clones are lexically not differ than the threshold defined by user, away from each other. These can be better exploited using dynamic programming technique.

This technique is later extended by Kamiya et al.[48] utilizing additional source normalizations to erase superficial differences. His technique named as CCFinder. These changes can be statement bracketing (e.g., if(x) y=10; vs if(x) ky=10; g). The other basis for this technique is Gemini, which shows near miss clones utilizing scatter plot or Dot plot technique. RTF permits the uses to tailor the tokenization for improved a clone detection using a more memory efficient suffix-array instead of suffix trees.

c. Syntactic approaches

Parsing is the technique which is used to convert the source code programs into the required parse trees or Abstract Syntax Trees (ASTs) in syntactic approaches. These parse trees or abstract syntax trees can be processed utilizing either structural metrics or tree matching to find the clones present in the source code.

Finding similarities between sub trees is the basic idea behind tree-based techniques or tree-matching approaches. The literal values, names of the variables and other leaves or tokens in the source code may be abstracted in the basic construction or representation of the tree. This allows more sophisticated clone detection process.

One of the existing tree-matching clone detection approaches is Baxter et al.'s CloneDr [9]. In his approach he used a compiler generator to generate a constructor for the annotated parse trees. Hashing is then used to cluster the sub trees into buckets. Sub trees are then compared to each other within the same bucket by a tolerant tree matching algorithm. Though the use of hashing is optional but it reduces the total number of tree comparisons drastically.

The same approach has been adapted for Abstract Syntax Tree –based clone detection by Bauhaus [56] as CCDIML. The basic difference between CloneDr and CCDIML is CCDIML's modeling of sequences, which makes easy to search in groups of sub trees that form clones together and its accuracy in matching of trees.

Yang [57] proposed a dynamic approaching of programming for discovering the syntactic differences in similar sub trees comparison. His approach CDIFF is not known for clone detection but the technique can be used for clone detection.

Whaler et al. [58] find parameterized clones along with exact clones by converting AST to XML at a higher level of abstraction using a data mining technique to detect code clones. Evans et al. [59] proposed a structural abstraction, that allows the variation in arbitrary sub trees than just leaves or

tokens for handling near –miss clones along with exact clones using gaps.
To reduce the complexity of complete sub tree comparison, some of the recent researchers used an alternative tree representation. In this approach, according to Koschke et al. [37] used AST sub trees which are serialized as sequences of AST nodes for which suffix trees are constructed after words. This approach makes to discover the syntactic clones at the pace of token based techniques. Another approach proposed by Tairas and Gray which based on Microsoft Phoenix framework [60] detects function level clones using suffix trees.

d. Metric based approaches
The collection of number of metrics for given code fragments and comparing their metric vectors instead of comparing code or Abstract syntax trees directly is known as metric based technology. A method which is a well-known method in the literature of code clone uses fingerprinting functions and metrics calculated for their syntactic units. These units consist of class, method, function and statements which yield values that can be compared to detect clones.
Generally, parsing of source code to an Abstract syntax tree or Control Flow Graph (CFG) is happening first, and then calculation of the metrics is taking place. Mayrand et al.[61] use many different metrics to find functions as clones which has similar metric values. These metrics are calculated using layout, names, control flow and expressions of the functions. The functional clone can be discovered as a pair of total function bodies along with similar metric values.
Davey et al.[64] proposed a clone detection approach which detects near-miss clone, parameterized and exact clones. In this method first computation of certain features of blocks of code and then process some training to neural networks to identify similar blocks depending on features. To find duplicate web pages and clones in the content of the web documents, the metric based clone detection approaches can be applied.

D. Clone detection techniques and tools
Clone detection techniques try to find duplicated code. These might undergo some minute changes later. To make out copy –past –adapt code and to substitute it by a single procedure is the typical motivation for clone detection. The code in large software system which is modified and replicated by hand is found by clone detection [29].
People usually copy the code which is conceptually identifiable blocks and make very few changes. This shows that the same syntax is noticeably repeated. Clone detection can identify all such things. The presence of a useful problem domain concept is an indication of each identified clone. Simultaneously this provides an example for implementation. The parameters or points of variation can be identified by the differences between the copies. The product line development of clones can be enhanced in several ways.
Some of them are removal or redundant code, reducing maintenance costs, identifying domain concepts for using them in the present system or the next and identifying parameterized reusable implementations [30]. Detection of code clones helps in software development and maintenance of tasks including identification of refactoring candidates, location of potential bugs and finally understanding software evolution. Many clone detectors are based on the similarity of text [31].
The productivity of software maintenance is hampered by cloning in classical code –based development environments. This is because reforms to cloned code are error –prone as they should be carried out multiple times for all instances of a clone. Therefore, the software engineering community promoted a multitude of approaches and strong tools for detecting code clones [32]. The various sections of code which occur in multiple locations of a program are code clones. The purpose of using clone detection tools is to search for clones automatically and to report any identified clones back to the user.
Apart from the source file names and starting and ending of line locations of a single clone instance, various clones are listed together in the textual representation of the result. Clone detection results can be represented graphically. A popular way of graphical representation of clone detection results is scatter plot. In a scatter plot, the duplicate sections of code are identified as a sequence of dots connected in a graph [33]. Some Type 2 clones and most of Type 3 clones are not identified by fast algorithms. Those that targeted to find Type 3 clones using dependence –based algorithms might locate Type 3 clones but at a very high computational cost. Therefore, the present state of the art shows the software engineers status with a classic ‘speed –quality’ trade off.
Code clones are essential to be tracked, managed and if possible it should be removed through refactoring depending on feasibility. The IDEs should unite to lend support for all such activities for clubbing clone management, with actual development effort. However, most of the clone detectors are promoted as separate tools. There are only few tools that are integrated along with IDEs. These tools are all the time focused in the detection of Type 1and Type 2 clones. They are also expected to offer enough support for flexible code clone management and

refactoring[34]. In this point of view, clone detection techniques are outstanding for two reasons. One is occurrence of code cloning within scattered cross – cutting concern implementation.
In the first instance, scattered code is not properly modularized. There are many reasons for that. One reason for this lack of modularity is missing features of the language that has to be implemented. Exception handling is an example for this. Other reason for that is the way the system was designed. In either of the cases developers are not able to reuse the concerned implementations using the language module mechanisms. Hence, the developers are compelled to write the same code over and over again. This results in the practice of copying existing code fragment and modifying it slightly to their needs [35]. The improvement of quality of source code by refactoring cloned code fragments is a major application of clone detection [36].

a. Taxonomy of detection techniques
There are several properties or dimensions for each of the clone detection techniques using these properties, a particular technique can be classified. For instance, how it does, what it does etc., some of the properties are discussed below.
Source transformation/Normalization:Before applying the actual comparison, instead of directly working in the raw source code, each approach uses a kind of transformation or filtering or normalization. Some approaches utilize comprehensive transformation where as some just remove white spaces or comments. This is done to evolve an alternative form of code representation suitable for tracing target clone types for the reengineering purpose and the comparison algorithm. Hence, source transformation / normalization is used for a specific method with all the above said properties.
Source representation:By utilizing various types of transformations /normalizations or filtering an apt code representation is procured to meet the requirements of the target algorithm which is used for comparison of the source code. This means that code representation is used for comparison phase.
Comparison granularity:Various algorithms work on diverse code representations on different levels of granularity. The granularity of one source code line is taken up by some algorithm and AST /PDG nodes are taken up by others. With the use of this property, the type of code clone granularities used for a specific technique in the comparison phase is revealed.
Comparison algorithm:In the detection of clones of various types, one of the major concerns is choice of the algorithm. All the algorithms from diverse areas are taken into consideration for clone detection. For instance, the sequence matching algorithm is used by some approaches. This sequence matching algorithm is usually applied in the biological science for DNA sequencing. There are other algorithms like data mining /information retrieval algorithms which are used for other applications. With the use of this property, the kind of comparison algorithm used for a particular method is revealed.
Computational complexity:The major concern is computational complexity of a clone detection technique should scale up to identify clones in large software where millions of lines of code exist. The kind of transformations and the comparison algorithm used ultimately decide the complexity of an approach. With the use of this property, the overall computational complexity required for a particular method is revealed

E. Related works
A lot of research has been done related to the detection of clones. Utilization of artificial intelligence techniques like Abstract Syntax Trees, Kclone, Frequent Itemset, Substring Matching techniques has attracted the attention of researchers. A review of related literature is discussed below:

According to Rainer Koschke et al. [37] reuse of software through copying and pasting was a constant plague in software development in spite of the fact that it leads to serious maintenance problems. Different techniques are proposed to locate the duplicated redundant code. This is also known as software clone. Rainer compared all the techniques and proved that token –based clone detection on suffix trees is tremendously quick. However, it gives clone candidates which are usually non syntactic units. The current techniques which are based on abstract syntax trees locate syntactic clones. These syntactic clones are less efficient. It enables how to make use of suffix trees in order to locate clones in abstract syntax trees. This new method could find syntactic clones in linear time and space. Many large case studies results are reported. The new technique is used to compare using the Bellon bench mark for clone detectors.

StephaneDucasse et al. [38] put forward the proposal that there can be diverse problems for the maintenance of software if duplicated code is used. It is really difficult to make out in large software systems various techniques are developed to locate software clones. Some of them are not only highly sophisticated but also very expensive to execute and adapt. There are some techniques which are very easy to implement. They are light weight techniques, which are based on simple string matching

algorithms. However, these techniques may not be effective. A simple string –based approach is used to a number of languages such as JAVA, PYTHON, COBOL, C, C++, SMALL TALK, PASCAL AND PDP-II ASSEMBLER.
The time taken to adapt the approach to a new programming language was less than 45 minutes in each case. In the experiment, assessment of the quality of clone detection for various case studies was done. A variety of simple variants of string –based clone detection techniques are used to bring normalization to differences which occur because of common editing operations. The results showed that a clone detection technique which is not expensive usually receives high recall value and acceptable precision value. An unacceptable number of false positives may come up if the code is excess normalized before comparison.

According to Chanchal Kumar Roy et al. [39] various techniques and tools for detecting software clones have been proposed. In his research in order to organize a large amount of information into a framework which is conceptual and coherent a qualitative comparison and assessment of the current state of the art is provided in clone discovering techniques and tools.
The back ground concept, a generic clone detection process and taxonomy of present techniques and tools are discussed. There are two dimensions in which classification, comparison and evaluation of the techniques and tools can be done. First the approaches are classified and compared depending on the number of facets that has a group of attributes. Secondly, the classified techniques and tools are evaluated qualitatively. This is done with respect to classification of editing scenarios designed for model the creation of all four types of clones (i.e. type 1, type 2, type 3, and type 4). The research has also shown how the results can be used to choose the appropriate clone detection technique or tool of a particular goals and constraints. There are two major contributions for this paper. They are i. classification of clone detection techniques and tools using a schema and classification of present clone detectors based on this schema. ii. A classification of editing scenarios which produced various types of clones and evaluation which is qualitative for the present clone detectors depending on the taxonomy.

According to YueJia et al. [40] for all the applications of detection of clones, it is necessary to identify algorithms which are efficient and precise. His work specifies a new algorithm, Kclone for detection of clones which includes a new combination of lexical analysis and local dependence analysis to get exact result by achieving high precision without losing speed. The initial results of case study implementing Kclone and its experiments have been dealt in detail. The results show that Kclone can be able to detect type 1, type 2 and type 3 clones when compared to PDG –based and token –based techniques. The results show the ability of the performance of an initial empirical study of the Kclone when compared to CCFinderX.

In the research work carried out by R.R.Brooks et al. [41] presented that, the cloning activity in adverse captures a sensor node, modifies its programs, creates number of clones of these and inserts these clones on to the network. The network sensor processing is subverted from within by the cloned nodes. In another paper, they showed the detection and removal of clones from sensor networks by using security methods which have random key pre distribution measures of security.
Clone codes consisting of keys are detected by utilizing authentication statistics based on the usage frequency of the keys. The random key pre distribution literature for its consistency using random key pre distribution technique and ease of explanation, the network used in that publication is Erdos-Renyetopology[65]. In this topology if two nodes need to be connected, the probability for this connection in the network is uniform. But the sensor nodes communication ranges are very limited. For this reason the topology has failed. In Brook's article, the application of clone detection technique was more realistic in network topologies. Adhoc and grid topologies shows node connectivity patterns of the networks of different nodes with range limits. This approach provides analytical methods for selecting detection threshold which accurately detects code clones. But they used only simulators to verify the method. In addition the other limitations of this approach are a number of nodes which can be inserted without properly being detected.

In his research Shinji Kawaguchi et al. [42] proposed that the maintainability and reliability is decreased for the software programs because of code clones. This is a major factor for development and cost of the maintenance phase. A new code clone detection /modification tool SHINOBI was introduced. This tool was designed to help in the recognition and highlighting of code clones for software maintenance tasks. SHINOBI had implemented as an add–in for Microsoft Visual Studio which reports modified snippets of clones in real time.

According to Nam H.Pham et al. [43] the important development framework for the maintenance of large scale software is Model Driven Engineering (MDE). Earlier researches have reported that cloning occurs in Model Driven Engineering just as it happens in traditional code based development. Not enough work has been done on clone detection with specifications on the precision value of the clone detection and completeness. This paper came up with a new tool for detection of clones i.e. ModelCD for Matlab /Simulink models. This tool enables the detection of not only exactly matched clones but also approximate model clones in an efficient and accurate method. The ModelCd consists of two new graph based algorithms discover clones systematically and incrementally with a maximum extent of accuracy, completeness and scalability. An evaluation is done empirically with diverse experimental studies on different real world systems. This was done not only to show the usefulness of the approach but also to compare the performance of ModelCD with the current existing tools.

Kodahai.E et al. [44] proposed clone detection has been in practice for the past ten years. This paved way for better results but at the same time increased complexity. Majority of the approaches confined to find program fragments which are same in the syntax or semantics. There were similarities between the candidate that were actually clones and the fraction of actual clones. In his paper, a new approach is suggested and this approach is metric based approach. This enables the textual comparison of the source code which helps in the detection of type IV functional clones in a procedure oriented programming language source code like C has been proposed. Different metrics which are suitable to discover the degree of clones in the programming language were formulated and their metric values were used during the process of clone detection. When it compared with other approaches, this approach is considered to be very less complex and it provides more efficient and accurate results in the way of clone detection. The results of this method were compared with the two existing methods for a open source project wetlab.

ArmijnHemel et al. [45] proposed that the software which is in binary form uses third–party packages without taking into consideration of their licensing terms. For example, many firmware consumer devices developed using Linux Kernel. But, most of these are not followed the General public license requirements of the GNU organization. The license violations are usually accidental. For instance, when the binary code from suppliers received by the vendors , there is no hint about its provenance. In order to trace out such violations, the Binary Analysis Tool (BAT) is developed. This system helps in detection clones in binaries. The firmware image attempts in detection cloning of code from the repositories of packages in binary form and source code. There were three clone detection techniques proposed by BATs clone detection. The effectiveness of these clone detection techniques have been evaluated and compared. They are scanning for string literals, detection of similarities and data compression using computing binary deltas.

According to Kodhi E et al. [46] a clone detection approach enables to reuse the code fragment in order to maintain the application. The clone detection techniques have identifies different kinds of clones. Clone detection enables better results and focused on complexity reduction of the work. The detection process becomes easy and efficient when a different clone detection tool is used. In the existing system, the focus is on line by live detection. Sometimes to find out the clone in the system, token based detection is used. The system will take long time to do all this.
When the fragment of code doesn't match with the exact code, however the functionalities bring out the similarities. The present system doesn't trace out the clones in it. This paper proposed an analysis in combination to detect all kinds of clone in a set of fragment of a source code. This enables better detection of all types of clone. During the detection process different semantics had been formulated and various values were used. There is less complexity in tracing out the clones and giving correct results.

F. Clone Detection Evaluation

There are many clone detection techniques existing and their corresponding tools. A comparison of these techniques or tools is worthy in order to choose right technique for a particular purpose. There are different parameters with which these tools can be compared. These parameters are usually called as clone detection challenges. Following are some of the parameters with which we can compare the tools.
*Portability:*Any tool should be portable in terms of different programming languages. Having hundreds of programming languages in use with several differences (dialects) among them, a clone detection tool required to be portable and easily configured for several languages and dialects showing syntactic variations of those languages.

*Precision:*Any tool must be sound enough that it can detect less number of false positives. Meaning, the tool should find code clones with higher precision. Most of the times this precision value along with recall value will be used as a measurement to assess the efficiency of a clone detection tool.

*Recall:*Any tool should be capable enough to find most of the clones in a system. Duplicated code fragments may not textually similar, but editing activities on the copied code fragments may mislead us to compare the similarity with the original code fragment, but it should consider as a clone. A clone detection technique that is said to be good must be robust enough to identify such clone relationships which are hidden.

*Scalability:*Any tool must be capable enough of finding code clones in a large code bases as the duplication of code is the most problematic in large and complex software system. A tool should handle complex and large systems with efficiency in terms of memory. Computational time is another concern of efficiency of the tool. In this research work main focus is on efficiency of the tool with different sizes of the code.

*Robustness:*As we discussed earlier any tool should be robust enough in terms of various editing activities that might apply on a copied fragment. i.e., it must detect different types of clones or all four types of clones with high precision and recall values.

G. Refactoring

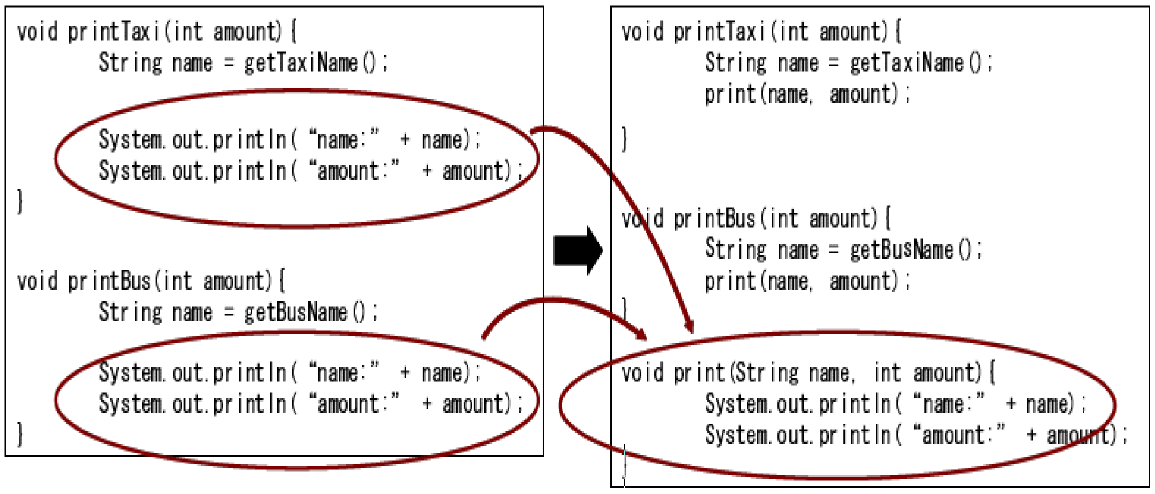
Refactoring is the process of rectifying the negative effect of code duplication. In software code clone literature various types of refactoring approaches have been mentioned. In this section we are discussing approaches which we have used in our method.

a. Types of Refactoring

In order to maintain speed in software development a good design is necessary. The process of refactoring enables the programmer to build software more quickly, for refactoring prevents the design from decaying.

Fowler[11] analyzed in his book about the common mistakes in coding. These mistakes are mentioned as bad smells of coding patterns.The book clearly mentioned about 22 bad smells (code patterns that need to be refactored) and 72 refactoring patterns (modification techniques to erase bad smells) are used to remove these bad smells. As per this book, the top position of the bad smells is nothing else than duplicated code. There are different patterns that are used to remove code clones. The following sections explain these patterns.

Extract method: To enhance the readability, understandability and maintainability of a code clone, extract method is used. This method is applied to a lengthy method or sometimes to a complex function. Even to merge the code clones, this method can be used. Figure 2.1 illustrates an example of extract method. There are duplicated instructions for the two methods before the refactoring is done. Once the refactoring is done, the duplicated parts are used as a new method. A caller statement of the new method replaces the duplicated code.



```
void printTaxi(int amount){
        String name = getTaxiName();

        System.out.println( "name:"  + name);
        System.out.println( "amount:"  + amount);
}

void printBus(int amount){
        String name = getBusName();

        System.out.println( "name:"  + name);
        System.out.println( "amount:"  + amount);
}
```

```
void printTaxi(int amount){
        String name = getTaxiName();
        print(name, amount);

}

void printBus(int amount){
        String name = getBusName();
        print(name, amount);
}

void print(String name, int amount){
        System.out.println( "name:"  + name);
        System.out.println( "amount:"  + amount);
}
```

Figure 2.1 Example of extract method

Pull up method: A method in which a class is moved to its parent class is known as pull up method. When there are similar methods for many child classes, an effective refactoring way is to change them to the common parent class. When the similar methods have the same body, which suggests that there is a copy and paste, the easiest way to identify is to use the pull up method. Figure 2.2 shows an example of pull up method. In the figure, before refactoring is done, two classes have identical methods (Salesman and Engineer classes). The duplicated code could be eliminated in these classes by pulling them up to the common parent class (Employee class).

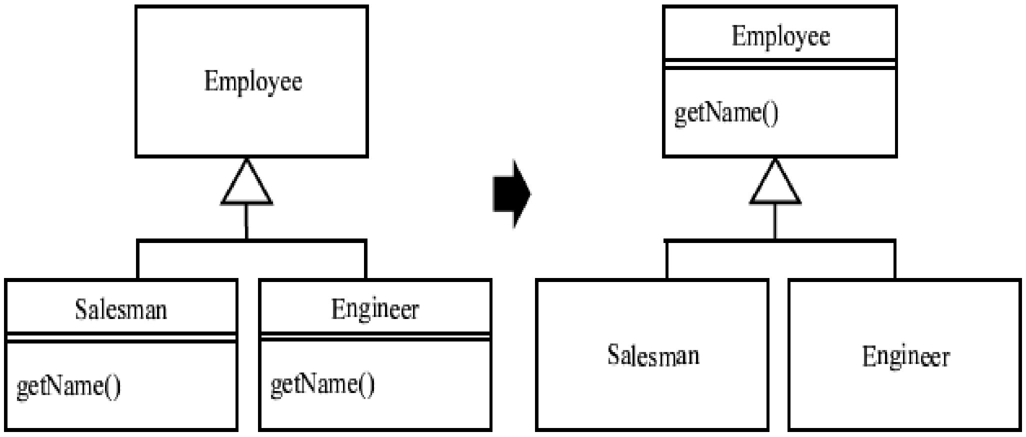


Figure 2.2 Example of pull up method

Move method: Move method is almost the same as pull up method. The only difference is that duplicated methods are changed to a new class, but not the parent class, which is an extended version by the current class. Pull up method follows the principle of inheritance but not move method. Sometimes programming environment forces us to use move method rather than pull up method.

Extract superclass: There cannot be a common parent class when two or more classes have similar functionalities. In such instances extract super class can eliminate the duplicated code. The programmer first can create a class which will be a parent class to both the classes and then use pull up method to each duplicated method.

However we are using only Precision and Recall to prove the efficiency of the model that we proposed. This research work is limited to compare these two values with the existing models.

II. Conclusion

This paper presents the review of literature to encourage the research in the area of code cloning. It is also mentioned the scope of solutions to reduce the maintenance complexity in the form of refactorization of discovered clones

References

[1] IEEE. *Standard for Software Maintenance*.IEEE Standard 1219, 1998.

[2] ISO/IEC. *Software Engineering - Software Maintenance*. ISO/IEC 14764,1999.

[3] L. Arthur. *Software Evolution: The Software Maintenance Challenge*. Wiley,1988.

[4] S. W. L. Yip and T. Lam.A software maintenance survey. In *Proc. of the 1stAsia-Pacific Software Engineering Conference*, pages 70–79, Dec 1994.

[5] S. Chidamber and C. Kemerer.A metric suite for object-oriented design. *IEEE Transactions on Software Engineering*, 25(5):476–493, Jun 1994.

[6] ClearCase. http://www-306.ibm.com/software/awdtools/clearcase/.

[7] Robert Tairas, "Clone detection and refactoring", Proceeding of OOPSLA '06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 780-781, New York, USA, 2006

[8] Chanchal K. Roy, James R. Cordya and Rainer Koschkeb, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal Science of Computer Programming, Vol. 74, No.7, pp. 470-495, May 2009.

[9] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna and Lorraine Bier, "Clone Detection Using Abstract Syntax Trees", Proceedings of the International Conference on Software Maintenance, pp. 368, Washington DC, USA 1998

[10] G.Anil Kumar, Dr.CRK.Reddy, Dr.A.Govardhan "Code duplication in Software Systems: A survey", International Journal of Software Engineering Research & Practices Vol.2, Issue 1, Jan 2012

[11] M. Fowlor. *Refactoring: improving the design of existing code*. Addison Wesley, 1999.

[12] R. H. Page. http://www.refactoring.com/.

[13] MagielBruntink, Arie van Deursen,Remco van Engelen, and Tom Tourwe, "On the Use of Clone Detection for Identifying Crosscutting Concern Code", Ieee Transactions On Software Engineering, Vol. 31, No. 10,pp. 804-818, October 2005

[14] Abouelhoda M.I., Kurtz S.andOhlebusch E, "The enhanced suffix array and its applications to genome analysis", In Proc. Workshop on Algorithms in Bioinformatics, vol. 2452,pp. 449–463, Berlin, 2002

[15] Hamid Abdul Basit and Stan Jarzabek, "Detecting Higher-level Similarity Patterns in Programs", European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 1-10 Lisbon, Sept. 2005

[16] Lingxiao Jiang, Zhendong Su and Edwin Chiu, "Context-based detection of clone-related bugs", Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 55 – 64, New York, USA, 2007.

[17] Chanchal Kumar Roy and James R Cordy, "A Survey on Software Clone Detection Research", Computer and Information Science, Vol. 115, No. 541, pp. 115, 2007

[18] J Howard Johnson. Identifying Redundancy in Source Code Using Fingerprints. In *Proceeding of the 1993 Conference of the Centre for Advanced Studies Conference (CASCON'93)*, pp. 171-183, Toronto, Canada, October 1993.

[19] Zhenmin Li, Shan Lu, SuvdaMyagmar, and Yuanyuan Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. In *IEEE Transactions on Software Engineering*, Vol. 32(3): 176-192, March 2006.

[20] G.Anil Kumar, Dr.CRK.Reddy, Dr.A.Govardhan "An Efficient Method-Level Code Clone Detection Scheme Through Textual Analysis Using Metrics", International Journalof Computer Engineering & Technology, pp 273-288, Chennai, India.

[21] Elizabeth Burd and Malcolm Munro.Investigating the maintenance implications of the replication of code.In *Proceedings of the 13th International Conference on Software Maintenance (ICSM'97)*, Bari, Italy, September 1997.

[22]G.Anil Kumar, Dr.CRK.Reddy, Dr.A.Govardhan "code clone detection with refactoring support through textual analysis", International Journal of Computer Trends and Technology- Volume2 Issue2- 2011

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

[24] MagielBruntink. Aspect Mining using Clone Class Metrics.In *Proceedings of the 1stWorkshop on Aspect Reverse Engineering*, 2004.

[25] MagielBruntink, Arie van Deursen, Remco van Engelen, Tom Tourwe. On theUse of Clone Detection for Identifying Crosscutting Concern Code.*Transactions onSoftware Engineering*, Volume 31(10):804-818, October 2005.

[26] Andrew Walenstein and ArunLakhotia. The Software Similarity Problem in Mal-ware Analysis. In *Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, andSimilarity in Software*, 10 pp., Dagstuhl, Germany, July 2006.

[27] GiulianoAntoniol, Gerardo Casazza, Massimiliano Di Penta, Ettore Merlo. ModelingClones Evolution through Time Series. In *Proceedings of the 17th IEEE InternationalConference on Software Maintenance (ICSM'01)*, pp. 273-280, Florence, Italy, November 2001.

[28] W-K. Chen, B. Li, and R. Gupta. Code Compaction of Matching Single-Entry MultipleExit Regions. In *Proceedings of the 10th Annual International Static Analysis Symposium (SAS'03)*, pp. 401-417, San Diego, CA, USA, June 2003.

[29] MagielBruntink, Arie van Deursen,TomTourwe and Remco van Engele, "An Evaluation of Clone Detection Techniques for Identifying Crosscutting Concerns", Proceedings of the 20th IEEE International Conference on Software Maintenance, pp. 200- 209,Washington DC, USA 2004

[30] Ira D. Baxter and Dale Churchett, "Using Clone Detection to Manage a Product Line", Clone detection using abstract syntax trees, pp. 1-3,1998

[31] HeejungKimy, YungbumJungy, SunghunKimx and Kwangkeun Yi, "MeCC: Memory Comparison-based Clone Detector", 33rd international conference on software engineering, Waikiki,Honolulu, Hawaii, May 21-28,2011

[32] Florian Deissenboeck, Benjamin Hummel, ElmarJurgens, Bernhard Schatz, Stefan Wagner, Jean-François Girard and Stefan Teucher, "Clone detection in automotive model-based development", Proceedings of the 30th international conference on Software engineering, pp. 613-622,New York, NY, USA,2008

[33] Robert Tairas, Jeff Gray and Ira Baxter, "Visualization of clone detection results",Proceedings of the 2006 OOPSLA workshop on eclipse technology exchange ACM, pp 50-54, New York,USA,2006

[34] Minhaz F. Zibran and Chanchal K. Roy, "Towards Flexible Code Clone Detection, Management, and Refactoring in IDE", Fifth International Workshop on Software Clones,Waikiki, Hawaii, USA,May 23,2011

[35] M. Kim, L. Bergman, T.A. Lau, and D. Notkin, "An Ethnographic Study of Copy and Paste Programming Practices in OOPL," Proc. Int'l Symp. Empirical Software Eng. (ISESE '04), pp. 83-92, Aug. 2004

[36] M. Rieger, S. Ducasse, and G. Golomingi, "Tool Support for Refactoring Duplicated OO Code," Proc. European Conf. Object- Oriented Programming (ECOOP '99), pp. 177-178, June 1999.

[37] Rainer Koschke, RaimarFalke and Pierre Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," In Proc. of the 13th Working Conference on Reverse Engineering, Benevento, pp. 253 - 262, Oct 2006.

[38] StephaneDucasse, Oscar Nierstrasz and Matthias Rieger, "Research On the effectiveness of clone detection by string matching," Journal of Software Maintenance and Evolution: Research and Practice, Vol. 18, No. 1, pp. 37-58, 2006.

[39] Chanchal K. Roy, James R. Cordy and Rainer Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Science of Computer Programming, Vol. 74, No. 7, Feb 2009.

[40] YueJia, David Binkley, Mark Harman, Jens Krinke and Makoto Matsushita, "KClone: A Proposed Approach to Fast Precise Code Clone Detection," In Proc. of the Third International Workshop on Detection of Software Clones (IWSC 2009), pp. 12-16, 2009.

[41] R. R. Brooks, P. Y. Govindaraju, M. Pirretti, N. Vijaykrishnan and M. Kandemir, "Clone Detection in Sensor Networks with Ad Hoc and Grid Topologies," International Journal of Distributed Sensor Networks, Vol. 5, pp. 209–223, 2009.

[42] Shinji Kawaguchi, TakanobuYamashinay, HidetakeUwanoz, KyhoheiFushida, Yasutaka Kamei, MasatakaNagura and Hajimu Iida, "SHINOBI: A Tool for Automatic Code Clone Detection in the IDE," In Proc. 16th Working Conference on Reverse Engineering, pp. 313 - 314, Oct 2009.

[43] Nam H. Pham, HoanAnh Nguyen, Tung Thanh Nguyen, Jafar M. Al-Kofahi and Tien N. Nguyen, "Complete and Accurate Clone Detection in Graph-based Models," In Proc. of the 31st International Conference on Software Engineering, Washington, DC, 2009.

[44] Kodhai. E, Kanmani. S, Kamatchi. A, Radhika.R and VijayaSaranya. B, "Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics," In Proc. of the 2010 International Conference on Recent Trends in

Information, Telecommunication and Computing, Washington, DC, pp. 241-243, 2010.

[45] ArmijnHemel, Karl TrygveKalleberg, Rob Vermaas, and EelcoDolstrac, "Finding Software License Violations Through Binary Code Clone Detection," In Proc. of the 8th working conference on Mining software repositories, New York, NY, May 2011.

[46] Kodhai.E, Perumal.A, and Kanmani.S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones," In Proc. of the International Joint Journal Conference on Engineering and Technology (IJJCET 2010), pp. 99 - 103, 2010.

[47] F. Calefato, F. Lanubile and T. Mallardo, “Function Clone Detection in WebApplications: A Semiautomated Approach,” in *Journal of Web Engineering*, vol.3, no. 1, pp 3–21, 2004.