Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 132 (2018) 718–727

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2018)

Code Clones: Detection and Management

Neha Saini[a,*], Sukhdip Singh[b], Suman[c]

[a,b,c] *Deenbandhu ChhotuRam University of Science and Technology, Murthal, India*

Abstract

In a software system, similar or identical fragments of code are known as code clones. Instead of implementing a new code from scratch, most of the developers prefer copy–paste programming in which they use existing code fragments. So, the primary reason behind code cloning isboth developers and programming languages used by them. Reusing existing software for increasing software productivity is a key element of object oriented programming which makes clone detection and management a primary concern for current industry. As a software system grows, it becomes more complex which leads to difficulty in maintaining it. The main reason behind difficulty in software maintenance is code clones which do not lead to conclusion that code clones are only harmful for software development. Code clones can be both advantageous and disastrous for software development. Therefore, clones should be analysed before refactoring or removing them. For analysing the clones properly, there is a need to study all the clone detection techniques, various types of clones and techniques to manage them.The main purpose of this paper is to gain insight in to the research available in the area of clone detection and management and identify the research gaps to work upon. It will help the researchers to get started easily with clones as they can study the basic concepts, techniques, general steps for code clone detection and management and research gaps together at one place. Also, it will help in the selection of appropriate techniques for detecting and managing clones as comparative analysis of different techniques on the basis of various parameters is also given in the paper.

Keywords: Clone Lifecycle, Clone Detection, Clone Management, Software Maintenance, Clone Refactoring, Bug Detection, Bad Smell.

1. Introduction

In a software program, different types of replication and redundancy can be found which is called cloning. There are various definitions given by different researchers. Fowler defined clone as a duplicate code which is an example of bad smell [1]. In software development, it is very frequent activity to copy or use fragments of existing code either with little modification or without any modification into other sections of code. The code that has been copied from

Corresponding author email: neha3998akalacademy@gmail.com

10.1016/j.procs.2018.05.080

existing code is called a software clone and the process is known as software cloning. Therefore, if a bug is detected in any section of code, correction is required in all replicated code fragments. This creates a need to find fragments in the source code which are related to each other. Basic lifecycle of a clone is shown in Fig.1. Basically clones can be created intentionally or unintentionally. Both intentional and unintentional clones are considered to be baseline clones. Clones can be refectory or removed if they are harmful for software maintenance. In the field of software clones, Roy and Cordy[2] and Koshchke[3] has done two landmark literature surveys. Volume of research in this area is increasing continuously. Software clone has emerged as an active area of research in which clone detection and management is one of the main aspect to focus upon.

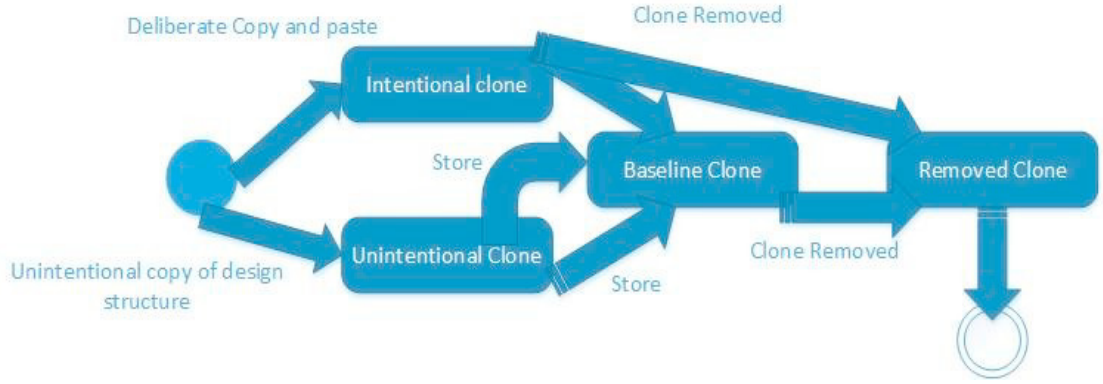


Fig.1.Lifecycle of a Clone

During code implementation and maintenance, code duplication is identified to be one of the main source of defects. The main issue with code clones is similar pieces of code which are indirectly related to each other and cannot be identified easily. If there is any change in some fragment of code, it needs to be propagated to other similar fragments of code. Otherwise, there will be inconsistency and unreliability in the developed system. The modification of software system has become more expensive and error prone due to declination in quality of code. Various studies have revealed that percentage of duplication in the software system is generally 20-30 percent [4]. If anexception is detected in the original code, we need to check for it in other related fragments as well in order to solve the main issue. Dhavleesh and Rajesh has given a detailed literature survey on clones and their detection [5]. Requirement of cloning practice in software development is very well explained in [6] [7] [8].

Due to advancement in technology, it has become easy to write the source code using code reuse practice. To adapt a software product in a new environment, we need to modify it which requires proper knowledge of the code structure and different dependencies between various fragments of code. To accomplish this, one need to study software detection and management in detail. Maintenance consumes around 90% of the total software development cost which shows a need for methods, techniques and tools that can keep track of software evolution[9] and helps in its maintenance. For this, proper study of clones is required.

2. Software Clones

The Merriam-Webster dictionary defines a clone as one that appears to be a copy of an original form, thus being synonymous to a duplicate. If an original code is copied and pasted with or without some modifications in the same software system or different software systems, then that copied fragment of code is known as clone. Till date, there is no suitable definition of code clones in software engineering field. Ira Baxters has defined clones as code segments that are similar according to some pre-defined definition of similarity [10]. Clone duplication or what we call cloning is one of the software reuse form as defined by Roy and Cordy. Baker has tested a sample program and concluded that based on exact matches and parameterized matches code shrinks by 14% and 61% respectively [4]. In case of large software systems almost 20-30% of program consists of cloned code [4]. The maintenance cost gets

increased by code duplication as stated by Fowler [1]. The practice of code reuse is increasing day by day due to increase in use of open source software's and variants. The software clone detection has become an active research to fulfil the urgent need to detect different clones present in software.

2.1. Clone terminology

The clone detection is done using clone detection tools that process the input source code and gives output in the form of clone pairs, clone class or clone family. If two fragments of code are similar according to some predefined measures, then there exists relationship between them [11]. Various terminologies related to code clones are discussed below in this section.

- **Code Fragment:** A sequence of code lines combined to make code fragment. It can be represented by a tuple (f,s,l) where f stands for file name, s is start line number and 1 defines total number of lines. End line numberof code fragment can be calculated by s+l-1. The tuple (f, s, e) can also be used to identify a fragment of code where e stands for end line number and e –s +1 defines the length of code fragment. The file name, start and end token numbers are used to identify a fragment of code when intermediate representation of the source code is employed.
- **Code Clone/Clone Pair:** Two similar fragments of code are defined as clones on the basis of some similarity definition. For e.g. in Fig.2. {c1, c2} and {c3, c4} are clones or clone pairs in software system 1. Code clones can also be found between two different systems. For e.g. in Fig.2. {c1, c5}, {c2, c5}, {c3,c6} and {c4,c6} are clone pairs between software system 1 and software system 2.

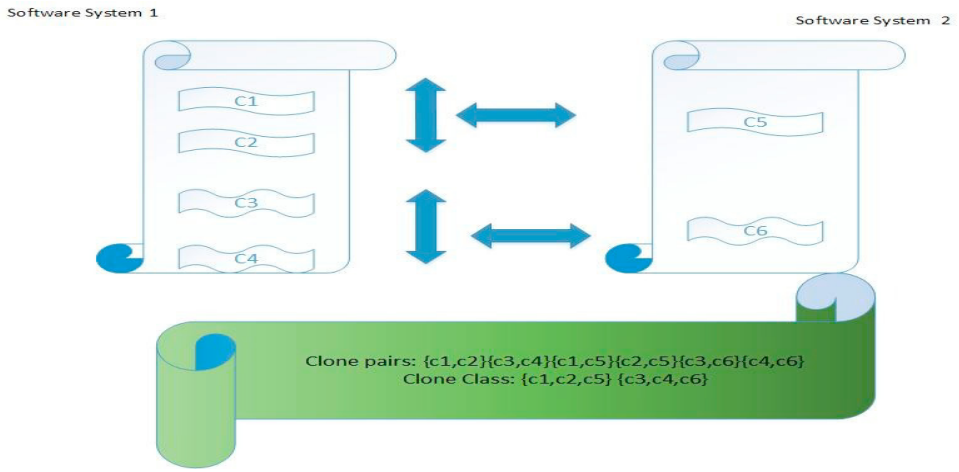


Fig. 2. An Example of Code Clone

- **Clone Class:**It is the collection of all the clone pairs which are related to each other by same equivalence relation. For e.g. in Fig.2. there are two clone classes between software system 1 and 2 i.e. {c1, c2, c5} and {c3, c4, c6}.
- **Clone Granularity:** It defines the syntactic boundary at which clones will be detected. There can be different levels of granularity like blocks,functions etc. It's completely the choice of programmer to fix the granularity or not. Clones with defined syntactic boundary are known as fixed granularity clones whereas clones without any predefined boundary are known as free granularity clones.
- **Clone Refactoring:** It is process of altering the program structure without any impact on the semantics of program. This process has so many advantages like improvement in code quality, readability and maintainability.
- **Language Paradigm:** It tells about the language paradigm targeted upon by the clone detection method being used. Language paradigm can be procedural or object oriented or both.

- **Precision and Recall:** Recall is the percentage of total clones detected in a software system including false positives whereas precision is the percentage of true clones detected in the system.

3. Need of Cloning and its Effects

In the last section a brief description about clones and different terminologies used in clones has been given that will help in better understanding of clones. In this section different reasons behind cloning, its advantages, disadvantages and process of clone detection is discussed.

3.1. Need and advantages of cloning

Reusing a code fragment for similar requirements by cloning is an efficient method in software development that helps in reducing cost and time. The use of templates has been encouraged by some of the paradigms of programming [9]. Sometimes, a developer writes a piece of code which accidentally matches with some existing code. This leads to accidental cloning. Developers fear to write the code from scratch due to its large size. Sometimes, people copy a section of code due to difficulty in understanding it.

There are a number of advantages of cloning as it proves to be very helpful when there is lack of knowledge about language on which programmer is working. It also helps in meeting all the deadlines of task assigned to each developer that are involved in some particular work. The research in this area has proved that it will save time and money as it makes software system development faster [12]. The software evolution and analysis becomes easier using it. It also helps in detecting different bugs existing in the code fragment so that those bugs will not affect the other program where that fragment will be used. The size of program gets reduced and it enhances the understand ability of a program. To protect copyright content from being copied a plagiarism detection tool plays an important role which makes use of the concept of cloning. Various applications of code clone detection and management are given in Fig.3

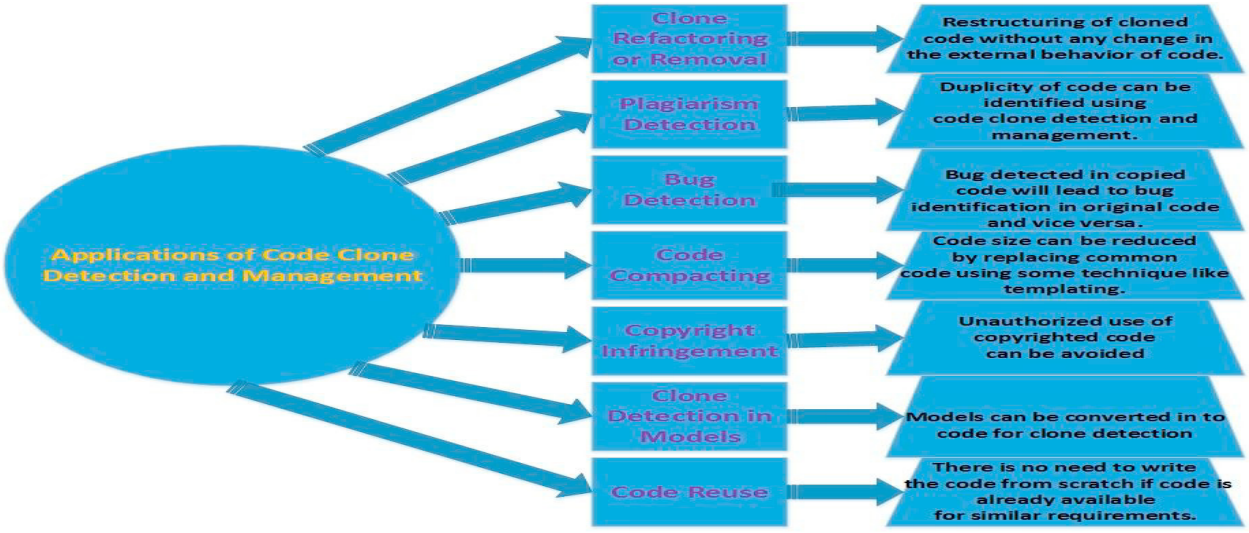


Fig.3.Applications of Code Clone Detection and Management

3.2. Disadvantages of cloning

- Increase in maintenance effort is an adverse effect of cloning as we need to keep track of all the copied sections. If any change is made in one copied section, it must be reflected in all the copied sections to remove inconsistency [13].
- The bugs get propagated from original code to the copied one which is known as bug propagation.

- There is bad impact on software system design as it becomes very difficult to understand it[14].
- Originality of the code gets lost as most of the time every part of the program is copied from different sources.

3.3. Steps involved in Clone Detection

In this section, theprocess of clone detection has been given in brief. On the basis of similarity in a program, clone pairs are detected using the process in Fig.4 (a). There are so many existing tools and techniques to detect clones and to do so different steps needs to be followed which are discussed below:

Step. 1: Pre-processing: This is the first step in clone detection which is further divided into different steps as given below:

- **Removal of irrelevant code**: All irrelevant source code like whitespace and comments will be discarded.
- **Generation of source units:**The remaining code will be divided into different set of disjoint fragments known as source units.
- These source units are further divided into even smaller units based on different comparison techniques used by the tool.

Step. 2: Transform: In this step, source units obtained in the previous step are converted in to some intermediate form which can be supplied as an input to comparison algorithm.There is a need of this step in all the techniques except text based technique. It can be achieved either using normalization or extraction. The extraction is further subdivided into three subcategories given below:

- **Tokenization:** Gathered source units from previous step are converted into tokens using some procedures or lexical protocols after removing comments, back spaces etc. Tokens are further arranged in to sequences.
- **Parsing:** Abstract Syntax Tree (AST) is generated by analysing the entire source code. AST is further divided in to subtrees [15]. Those subtrees are compared for clone detection.
- **Control and data flow analysis:**Program Dependency Graph (PDG) graph is created through some tools in which control and data dependency is represented by edges and statements by nodes. The PDG sub graphs are compared for clone detection.

Step. 3: Match Detection: In this step, a proper match is found by comparing the output of transformation phase i.e. transformed units using comparison algorithm. The clones are represented in the form of clone pairs, family and classes. The suffix trees and hashing are some of the comparison approaches that can be used.

Step. 4: Formatting: This phase is quite different from previous phase as in this phase, clone pairs of transformed code are mapped to original source code using file location.

Step. 5: Post processing: In this step, automatic heuristic or manual analysis is used to rank and filter out theclones. Human experts are used to filter out false positives by doing manual analysis. Heuristics based on diversity, length, frequency and other characteristics of clones are used to automatically filter out or rank clone candidates.

Step. 6: Aggregation: It is the last step of clone detection process which includes proper data contraction and analysis. The clone family and classes are formed by combining detected clone pairs.

4. Types of Clones and Clone Detection Approaches

In this section, a brief review of different clone types and various existing approaches to detect them has been given.

4.1. Types of clone

There is a need to study about different kinds of clones to detect them efficiently [16]. Mainly the clones are divided in to Exact, Renamed or Parameterized, Near-miss and Semantic clones. The other name given to these clones are Type 1, 2, 3 and 4 respectively. They are divided according to their speciality which is given in Table 1.

Table 1. Types of clones

S.No.	Clone type	Description
1.	Type 1 or Exact clone	These type of clones look like original code with variation of comments and blank spaces. They can be detected by use of text based, token based or any simple clone detection technique. Tools based on algorithms like KMP, SDD algorithm can detect these clones by string matching
2.	Type 2 or Renamed/Parameterized clone	Variations in name of literals, variables, keywords creates renamed or type 2 clones. The use of token or metric based techniques will be able to detect these kinds of clones in a code. CLAN, Columbus, MCD-finder are some of the tools used by developers to detect renamed clones.
3.	Type 3 or Near miss clone	They are created from base code by statement modification, addition or deletion [17]. Tools like ClonedDr, Clone Digger based on tree technique can be used to detect near miss clones. The AST sub tress is compared with each other in tree based techniques.
4.	Type 4 or Semantic clone	The program coding or syntax is different but the behaviour or function of the clone remains same in semantic clones. Tools like Duplex, Scorpio based on graph technique are proved to be fruitful in detecting these types of clones [18].

4.2 Clone Detection Techniques

In order to detect code clones, different techniques and tools have been developed over time by taking maintenance effort of clones in to account.The techniques are categorized according to input, representation and algorithms used by them. Broadly these techniques are classified as text based, token based, metric based, Abstract Syntax Tree (AST) based, Program Dependency Graph (PDG) based and hybrid as shown in Fig.4.(b). The detailed description of different techniques on the basis of various parameters along with their comparison is given in Table 2.Related research in the area of clone detection and management is summarized in Table 3.

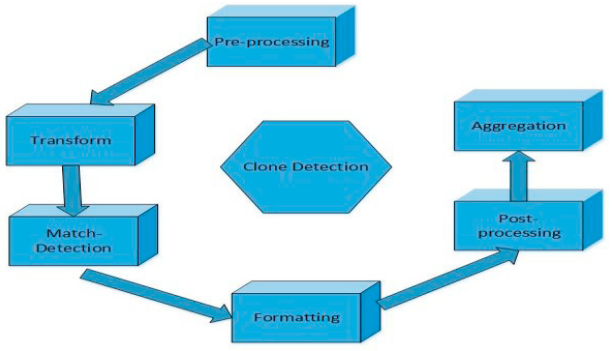
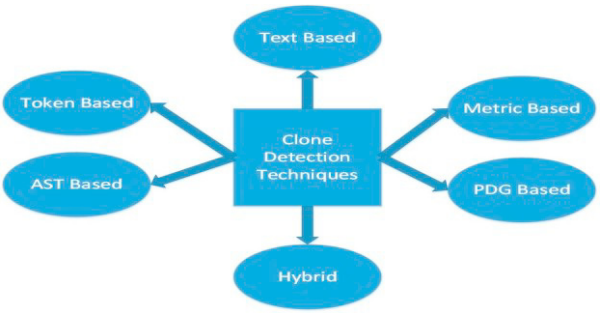


Fig.4.(a) Steps for Clone Detection



(b)Clone Detection Techniques

Table 2. Comparative Review of Clone Detection Techniques

Techniques ➡ / Parameters ⬇	Text Based	Token Based	Abstract Syntax Tree (AST) Based	Program Dependency Graph (PDG) Based	Metric Based	Hybrid
Portability (Ability to run on multiple platforms)	High	Medium	Low	Low	Metrics dependent	Depends on type of technique used
Efficiency (Quality of results)	High	Low	High	High	High	High
Integrality(Level of difficulty involved in integrating the technique in current environment)	Low	High	Low	Medium	Medium	Medium
Transformation (Method of pre-processing)	Removes white space and comments	Tokens are generated from source code	AST is generated from source code	PDG is generated from source code	Metrics value are calculated after AST is generated from source code	Depends on the hybrid technique
Comparison Based (Type of input taken by clone detection technique)	Lines of code	Token	Nodes of tree	Nodes of program dependency graph	Metrics value	Depends on the hybrid technique
Computational Complexity(upper bound on time taken for clone detection)	Depends on algorithm	Linear	Quadratic	Quadratic	Linear	Depends on the hybrid technique
Refactoring Opportunities (Refactoring can be done easily or not)	Good for exact matches	Good	Good for refactoring as it finds syntactic clones	Good for refactoring	Manual inspection required	Depends on the hybrid technique
Representation (How source code will be represented)	Normalized source code	In the form of tokens	In the form of abstract syntax tree	In the form of program dependency graph	Set of metrics values	Depends on the hybrid technique
Language Dependency (Language targeted by clone detection technique)	Adaptable	It needs a laxer but no syntactic knowledge required	Parser required	Parser required	Parser required	Depends on the hybrid technique
Type of Clone Detected	Type 1,2 and 3	Type 1,2 and 3	Type 1,2,3 and 4	Type 1,2,3 and 4	Type 1,2,3 and 4	Depends on the hybrid technique
Output Type	Clone pairs and Clone Classes	Clone pairs and Clone Classes	Clone pairs, Clone Classes, AST nodes	Clone pairs and Clone Classes	Clone pairs, Clone Classes, metrics value	Depends on the hybrid technique

5. Clone Management

The combination of various activities like clone classification, refactoring, visualization and tracking is known as clone management. In short, it includes all the activities that includes detecting, avoiding and removing the present clones. Clone management comes with a number of benefits like improvement in customer satisfaction and software system quality. It also helps developers during various activities like debugging and modification.

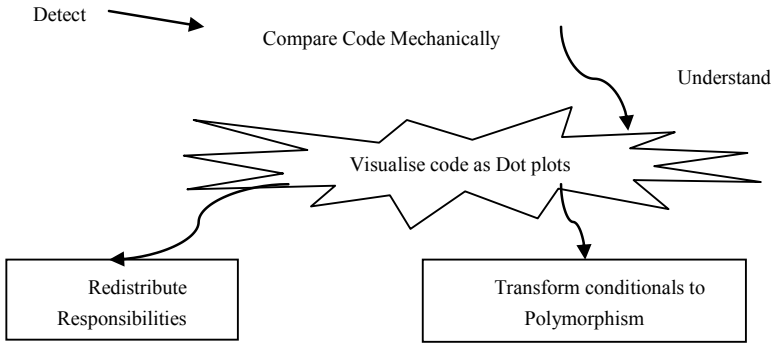


Fig. 5. Management of Duplicate Clones

Table 3. Related Research and Limitations

Author	Year	Code Representation	Comparison Method	Limitation	Output
J. Mayrand [20]	1996	Metrics	21 function metrics	Less precision and recall. It requires addition of metrics and the reduction of delta values to minimize the number of false positives. Worked for only procedural systems.	Clone pairs, Clone classes
Baxter [10]	1998	Abstract Syntax Trees (ASTs)	Exact-Tree Matching technique, and conventional transformational methods	Ignores small subtrees. Less precision and recall.	Clone pairs
R. Komondoor [9]	2001	Abstract SyntaxTrees (ASTs)	PDGs and program slicing to find non-contiguous clones	Long running time and Countless variants of ideal clone are identified.	Variants of the ideal clones
Toshihiro Kamiya [16]	2002	Metrics	Clone detection with transformation rules and a token-based comparison along with optimization techniques to improve performance and efficiency.	Does not accept source code in multiple languages.	Size of metrics
Chanchal K. Roy[21]	2009	Metrics	Hybrid clone detection method and Scenario-based comparison of clone detection tools	Lacks in efficient and error-free copy/paste and renaming	Precision and recall
Yoshiki Higo [22]	2011	Gemini	Incremental Code Clone Detection: A PDG-based Approach	Does not include actual software maintenance and lacks in functionalities	Precision and recall
Girija Gupta [23]	2013	Metrics	Code Clone Detection and Redesigning	It can detect clones only in java files.	Clone pairs
Kanika Raheja [24]	2015	Metrics	Code Clone Detection model on java byte code using hybrid approach	Does not work directly on the source code	Potential clones
Xiao Cheng [25]	2016	Abstract SyntaxTrees (ASTs)	CCSync and rule-directedapproach	76% of generated revisions are identical with manualrevisions	Precision over 92% and recall over 84%

In Clone Management, first step is to document the code segments and clone relationships occurring in those code segments. After that, there is a need to track the code base consistently during software development and incorporate the changes in documentation. Then clone documentation is analysed properly using some visualisation technique to remove or keep the clones [19]. In the next step refactoring is performed. If refactoring leads to change in program behaviour, then it should be roll backed. Another way to manage clones is clone prevention by adopting minimal copy paste use.

5.1. Clone Detection and Management Process

There are some general steps which are required for clone detection and management. Those steps have been summarized in the form of pseudocode given in Table 4. For e.g. if we want to detect and manage code clones in two files α and β, then α and β will be supplied as input to the program for detecting and managing clones. The program will make use of some clone detection technique π for detecting the clones on the basis of threshold value Ω. Threshold value sets the lower limit on the degree of similarity between two fragments of code. If the detected degree of similarity i.e. £ between two fragments of code c1 and c2 is greater than or equal to Ω, then c1 and c2 will be considered as clone pair. Detected clone pair will be stored in the database DB. Similarly all the clone pairs will be detected and stored in the database DB. After that clone pairs will be categorized in to harmful and useful clones. Useful clones will be retained and harmful clones will be refactored or removed using some algorithm.

Table 4. Pseudocode for Software Clone Detection and Management

Function CloneDetectionAndManagement (α, β, π, Ω, DB)

Inputs:

- Two source code files to be compared i.e. α and β.
- Detection Technique to be used i.e. π.
- Threshold value for clone detection i.e. Ω
- Database to store clones

Outputs:

- Clone pairs c1 and c2
- Clone detected in percentage i.e. £
- Maintenance Overhead of clones
- Rank wise Clones
- Refactored source code file α and β

Step I: Input source code files α and β to clone detection and management system.

Step II: Use a Clone detection approach π for detecting clones in source code file α and β.

Step III: If (£>=Ω) then go to step 4 else go to step 8.

Step IV: Export the detected clones to database DB and update DB.

Step V: Find maintenance overhead of clones.

Step VI: Compare and rank clones according to maintenance overhead.

Step VII: Refactor or remove clones.

Step VIII: Stop

6. Research Gaps

Code clones are substantial problem for code based development and continuously increasing at a tremendous rate. The major challenges for identification of code clones are 1) detection of semantic clones 2) detection of near miss clones that result due to changes in the code fragments 3) to identify the clones based on different levels of granularity like function, block, model and subsystem 4) detecting hidden clones i.e. detecting smaller size clones that are part of larger clone pairs 5) efficient code clone management to reduce maintenance effort.

7. Conclusion and Future Work

The main reason behind code cloning is copy and paste activity done by programmer. As code clones lead to Increase in maintenance cost and bug propagation, there is a need to remove them. On the other hand, practice of cloning in programming can increase program productivity and helps in plagiarism detection. So, there is a need to study the clones properly and categorize them in to harmful and useful clones. Harmful clones can be removed and useful ones can be retained by refactoring them using appropriate technique. There are different types of clones that exist in the source code and number of techniques to detect them has been proposed by different authors from time to time. In this paper, we have tried to give a brief review of key areas related to clones that will help researchers to get started with clones. Comparative review of various clone detection techniques summarized in the form of a table will help the researchers in selection of appropriate technique according to their needs. Also they can optimize the existing techniques as there is no single technique which can detect all types of clones easily. Different clone detection techniques can be hybridized with various optimization algorithms like bacterial foraging, lion optimization etc. to get optimal results. Type 1 and 2 clones are easy to identify as compared to type 3 and 4 clones. So, there is a need for tools and techniques which can detect type 3 and 4 clones efficiently. Also, there is not much research in the area of clone management and model clones. These areas can be explored. This study in the area of code clone detection and management can be replicated for model clones as well.

References

[1] Fowler, Martin, and Kent Beck (1999), "Refactoring: improving the design of existing code, *Addison-Wesley Professiona*l.

[2] Chanchal Kumar Roy, and James R. Cordy (2007), "A survey on software clone detection research" *Queen's School of Computing TR*541 (115): 64-68.

[3] Koschke, Rainer. "Survey of research on software clones(2007), " *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] Baker, S.Brenda "On finding duplication and near-duplication in large software systems(1995), " *Reverse Engineering, Proceedings of Working Conference IEEE*2:86-95.

[5] Rattan, Dhavleesh, Rajesh Bhatia, and Maninder Singh (2013), "Software clone detection: A systematic review" *Information and Software Technology*55 (7): 1165-1199.

[6]Duala-Ekoko, Ekwa, and Martin P. Robillard(2013), "Clone region descriptors: Representing and tracking duplication in source code." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20 (1): 3.

[7] Zhang, Gang, et al(2012), "Cloning practices: Why developers clone and what can be changed." *Software Maintenance (ICSM), 2012 28th IEEE International Conference:*285-294.

[8] Chatterji, Debarshi, Jeffrey C. Carver, and Nicholas A. Kraft(2013), "Cloning: The need to understand developer intent." *Proceedings of the 7th International Workshop on Software Clones*. IEEE Press: 14-15.

[9] Komondoor, Raghavan, and Susan Horwitz(2001), "Using slicing to identify duplication in source code." *International Static Analysis Symposium*. Springer, Berlin, Heidelberg: 40-56.

[10] Ira D.Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. (1998), "Clone detection using abstract syntax trees" *Software Maintenance, Proceedings., International Conference on*. IEEE:368-377.

[11] Bellon, Stefan, et al(2007), "Comparison and evaluation of clone detection tools." *IEEE Transactions on software engineering*33(9).

[12] Wagner, Stefan (2013), ''Software product quality control. Berlin"*Springer*.

[13] Kapser, Cory J., and Michael W. Godfrey (2006), "Supporting the analysis of clones in software systems." *Journal of Software: Evolution and Process* 18 (2): 61-82.

[14] Kapser, Cory J., and Michael W. Godfrey (2008), "Cloning considered harmful" considered harmful: patterns of cloning in software." *Empirical Software Engineering* 13(6): 645.

[15] Koschke, Rainer, Raimar Falke, and Pierre Frenzel (2006), "Clone detection using abstract syntax suffix trees." *WCRE'06. Working Conference on reverse engineering IEEE*(13)*:* 253-262.

[16] Monden, Akito, Daikai Nakae, Toshihiro Kamiya, Shin-ichi Sato, and Ken-ichi Matsumoto (2002), "Software quality analysis by code clones in industrial legacy software." In *Software Metrics, 2002. Proceedings. IEEE Symposium* (8):87-94.

[17]Bellon, Stefan, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo (2007), "Comparison and evaluation of clone detection tools." IEEE Transactions on software engineering 33 (9).

[18] M. Mondal, C. K. Roy, and K. A. Schneider (2015), "A comparative study on the bug-proneness of different types of code clones," in Proc. InternationalConference on Software Maintenance and Evolution (ICSME)IEEE: 91–100.

[19] Chatterji, Debarshi, Jeffrey C. Carver, Nicholas A. Kraft, and Jan Harder (2013), "Effects of cloned code on software maintainability: A replicated developer study." *Working Conference on Reverse Engineering (WCRE)*(20) :112-121.

[20]Jean Mayrand, Claude Leblanc and Ettore Merlo. (1996), "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics." *icsm*(96) : 244.

[21]Chanchal K.Roy, James R. Cordy, and Rainer Koschke.. (2009), " Comparison and evaluation of code clone detection techniques and tools: A qualitative approach" *Science of computer programming*74(7) : 470-495.

[22] Higo, Yoshiki, Ueda Yasushi, Minoru Nishino, and Shinji Kusumoto. (2011), "Incremental code clone detection: A pdg-based approach" *Reverse Engineering (WCRE) Working Conference on Reverse Engineering*, *IEEE*18: 3-12

[23] Girija Gupta and Indu Singh (2013), "A Novel Approach Towards Code Clone Detection and Redesigning" *International Journal of Advanced Research in Computer Science and Software Engineering*3(9): 331-338.

[24] Raheja, Kanika, and Raj Kumar Tekchandani.(2013), "An Efficient Code Clone Detection model on java byte code using hybrid approach"*IET* : 1-04.

[25] Cheng, Xiao, Hao Zhong, Yuting Chen, Zhenjiang Hu, and Jianjun Zhao (2016), "Rule-directed code clone synchronization" *Program Comprehension (ICPC), IEEE International Conference* (24) *:*1-10.