An Approach to Detect Clones in Class Diagram Based on Suffix Array

Amandeep Kaur,
Computer Science and Engg. Department,
BBSBEC Fatehgarh Sahib, Punjab, India.

Manpreet Kaur ,
Computer Science and Engg. Department,
BBSBEC Fatehgarh Sahib, Punjab, India.

Harjot Kaur,
Computer Science and Engg. Department,
BBSBEC Fatehgarh Sahib, Punjab, India.

***Abstract*— Copy-paste is becoming a very usual practice in software development. Copying a code in one or more place without any change is mainly known as software cloning and the pasted part of code is called clone. Clones in code increase the maintenance cost, resource requirements and make code more error prone. So detection and removal of clone is very important for source code. Models are also affected by the cloning problem. Model Driven Engineering now becomes a standard and important framework in software research area. Unpredicted copy of model elements leads to various difficulties. Models consists design level similarities and are in the same way harmful for software maintenance as code clones are. Therefore, clones are required to be identified from models. Class diagram is the main aspect of modeling and used to describe the static view of an application. Class diagram contains redundant elements which increase complexity of the class diagram as well as maintenance effort. Code quality can be improved if clones are detected from the class diagram. Current work aims to find clones in class diagram using an approach based on suffix array. Firstly, diagram is encoded as XML file and then tokens are extracted. Suffix array is used to compare tokens and matched tokens are known as clones.**

Keywords— clone; code clones; model clones; class diagram; complexity; Suffix array.

I. INTRODUCTION

According to Rattan et al. [15] and Roy and Cordy [19], repeating existing code and pasting them with or without changes into different sections of code is a common process in software system development. The copied code is termed as code clone and therefore the process is named code cloning.

Code clones have high impact on software quality. Biggest difficulty in code clones is that, these are only linked by their similarity and do not have any implicit or explicit link between them which makes software code clones difficult to detect [10]. When we make changes or updations at one place, other similar things remains unchanged accidently that deteriorates the code quality. Therefore, it is very important to find related fragments. However, Storrle [20] had mentioned code duplication or cloning as a form of software reuse and had concluded that software code can be shrinked to a percentage on doing exact matching. In today's technology, software reuse is highly supported by open source software.

A. Reasons for Software clones

There are various reasons due to which clones occurs. Rattan et al. [15] and Roy and Cordy [19] mentioned different reasons as follows:

- Lack of time: Programmers are bound to do copy-paste to meet hard time constraints.
- System's Complexity: Adopting a new system or understanding a complex system only promotes coping existing functionality, code and logic.
- Language loopholes: Due to limitations in programming language, programmers are forced to do copy paste. Many of the languages lack inherent support for code reuse. These loopholes become limitations of a programmer.
- Fear of adaptability of fresh code: Rattan et al. [15] identified that for programmer adopting a new ideas always have fear of getting wrong and lengthy code, which will again lead to reuse of existing code.
- Lack of abstraction: Programmer ignores or avoids abstraction of program due to time limits. Delay refactoring will give rise to high maintenance cost.

B. Advantages and disadvantages of clones

There are some positive points considered by Rattan et al. [15] and Storrle [20] to have clones:

- Sometimes use of templates are encouraged in programming paradigms.
- Hard time constraints have only option to use the existing functionalities.
- Overhead of procedure calls promotes code duplication.

Rattan et al. [15] and Storrle [20] had mentioned problems associated with clone presence, some of them are following:

- High maintenance cost and efforts.
- Increased probability of bug propagation.
- Wrong effect on design.
- Wastage of resources.
- Bad impact on system understanding.

The rest of the paper is organized as follows. Section II presents the background. In Section III, we have described the methodology to detect clones in class diagram using suffix array. In Section IV results are discussed. Section V gives the related work and Section VI concludes the current work and gives future directions.

II. BACKGROUND

A. *Cloning in Models*

According to Storrle [20] and Roy and Cordy [18], as the source code clone detection is large problem for code based development, the same problem also occurs for copied parts of models in model based development. Due to significant difference between programming languages code and models, clone notations and algorithms are difficult to directly transfer between them. Unexpected overlaps and duplications in models are termed as model based clones [11].

The various challenges in detecting model clones mentioned by Storrle [20] are:

- To derive a practical definition of model clone.
- To develop and implement an algorithm to detect model clones.

B. *Code Clones versus Model Clones*

- Identification: Source code are easily identified by names and procedures used whereas models use internal identifiers which are equal but not identical.
- Structure: Code is represented as a directory tree of text files or long characters of tokens, on other side models have graph like structure. Tools are used to represent models, so it is important to consider tool-specific representation into account.
- Type Categorization: Roy et al. [17] and Storrle [20] mentioned different types of clones as given in Table 1.

Table1. Types of code clones and model clones

Code Clones		Model Clones	
Type I: Exact Clone	A copy that is identical except most of changes in whitespaces and comments.	Type A: Exact model clone	A copy that is identical except from secondary notations or internal identifiers.
Type II: Renamed Clone	A copy with consistent changes to identifiers, variables types or functions names.	Type B: Modified model clone	A copy with changes to the element names attributes and parts.
Type III: Parameter clone	A copy allowing changes, additions or removal of statements.	Type C: Renamed model clone	A copy that allows actual changes in additions or removal of parts.
Type IV: Semantic clone	A copy of code that performs same function but different syntactic variants are used.	Type D: Semantic model clone	A copy in content, those are due to model part copying or language constraints.

C. *Various Clone Detection Techniques*

There are various clone detection techniques available. The only difference between these techniques is the granularity level of clone. Rattan et al. [15] and Roy et al. [17] proposed some of them as follows:

1) Text-Based Clone Detection: In this technique, the source program is taken as sequence of lines. As discussed by Roy et al. [17], two code fragments which are similar in terms of texts or strings are known as code clone. Tools available for text based detection are: Duploc, DuDe(line based comparison), Simian(detection in different programming languages), SDD (Clone detection in large systems), NICAD (Hybrid clone detector with high precision and recall).

2) Token-Based Clone Detection: Tokens are considered better for comparison according to Roy and Cordy [23]. Source program is transformed into sequence of tokens for comparison. Suffix tree and Suffix array is used mostly as data structure in token based detection. Suffix arrays are considered prior to suffix tree in term of space requirements. CCFinder is a tool which uses suffix tree to find similar tokens. CP-Miner is another token based tool that detects structural clones with high abstraction repeated tokens. Suffix array is widely accepted by many other tools like SHINOBI.

3) Tree Based Clone Detection: Rattan et al. [14, 15] mentioned that tree based clone detection transforms the source code into tree structure. Similar subtrees in the tree are searched using tree matching techniques and reported as clones. Addition or removal of sub-parts i.e. Type 3 clones are easily detected by tree based clone detection.

4) Graph Based Clone Detection: Pham et al. [12] and Rattan et al. [15] mentioned that the semantic information of source code is represented by Program Dependency Graph (PDG). On obtained PDG, subgraph isomorphism is applied to detect similarity. Duplix and PDG-DUP are the tools for finding similar subgraphs.

5) Metrics based clone detection: Various similarity based metrics are applied to suitable form of data structures to perform clone detection. CLAN uses metrics obtained from AST of source code [15].

III. RESEARCH METHODOLOGY

Class diagram is created using UML modeling tool MagicDraw. The model is converted to XML document. The document is parsed to extract the tokens (i.e meaningful information) which are then matched using suffix array. Similar tokens are categorized as clones. Clone analysis is carried out to get numbers of clones and number of instances of each clone. Clusters, the group of clones repeated, are also reported with their occurrence value. Clone coverage and percentage of class similarity is also calculated from the clone detection results. Fig.1 gives the overview of methodology followed.

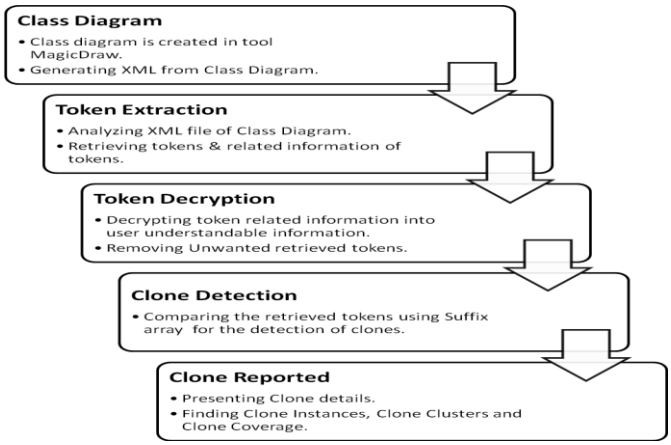

Fig.1 Steps of the Methodology

Fig.2 shows us with the interface of our tool. In step 1, we browse the XML file which is representation of class diagram from which we want to detect clones. Then using button 'Start Clones Detection', clone detection process is started. To check the report, button named 'View Report' is used. 'Exit' button can be used to exit the tool.

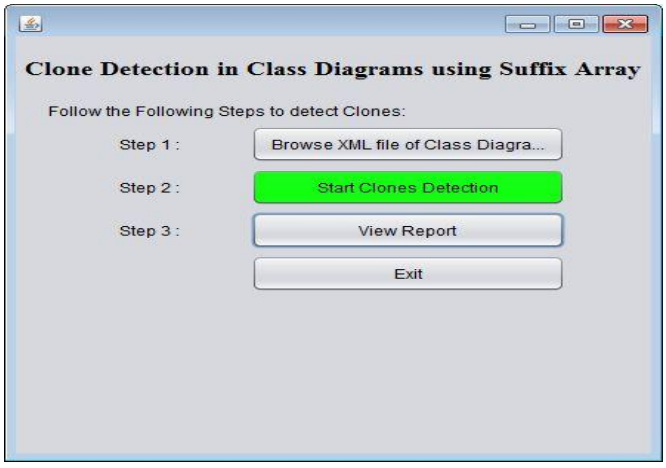

Fig.2 Interface of tool

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

This section discusses the results obtained after clone detection in UML class diagram. The methodology is explained in section 3. The class diagram of library management system is taken as a subject system which has 31 classes with 271 attributes and 62 operations. This UML class diagram is given as input to the clone detection approach based on suffix array. The clone detection results are presented on the basis of following parameters:

a) Clone Candidates and their instances.
b) Clone Clusters
c) Clone Coverage
d) Class Similarity
e) Memory
f) Runtime

A. Candidates

This parameter gives the number of clones detected from class diagram. The tokens are extracted from XML file of class diagram of library management system which are compared using clone detection approach based on suffix array. Similar tokens are reported as clones.

Clone Candidates: 65

1) Clone Instances: Clone instances parameter specifies the total number of occurrences of each clone which are present in various classes. Fig.3 shows the clones with their respective instances. Clones are differentiated from each other by using clone id. Clone id is an integer number assigned to each detected clone to give unique identification. Fig.3 shows clone with clone id 50 has maximum number of instances i.e. 26

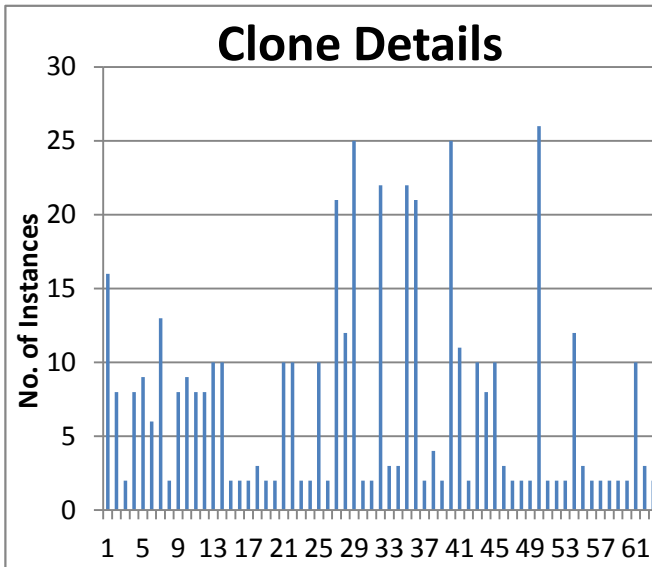

Fig.3 Instances of various Clones

Table 2 provides us information about clones and their instances exist in various ranges. It has been clearly shown that 36 clones have 2 to 6 instances and there are 7 clones having more than 17 instances.

Table 2. Number of Clones and their Instances

Clone Instances	No. of Clones
2-6	36
7-11	18
12-16	4
>17	7

B. Clone Clusters

Clone cluster defines the group of clones repeating together in various classes. Clusters are helpful to find maximum similarity in the classes. Table 3shows various clusters with their Id's, length (number of clones in the cluster) and instances (number of occurrences). Results show that the biggest cluster having 14 elements is repeated in two classes. Most of the clusters are of length two. Cluster having Cluster id 20 is of length three which is repeating in 25 classes. Cloned elements present in clone cluster can be put into super class to remove the redundancy in various classes. Fig.4 shows graph representation of clone clusters.

Table 3.Clone Cluster Details

Cluster ID	Cluster Length	No. of Instances
1	14	2
2	8	2
3	3	2
4	4	2
5	2	2
6	7	2
7	2	2
8	11	2
9	2	2
10	2	3
11	2	3
12	3	3
13	13	8
14	2	8
15	8	8
16	7	9
17	6	10
18	2	21
19	2	22
20	3	25

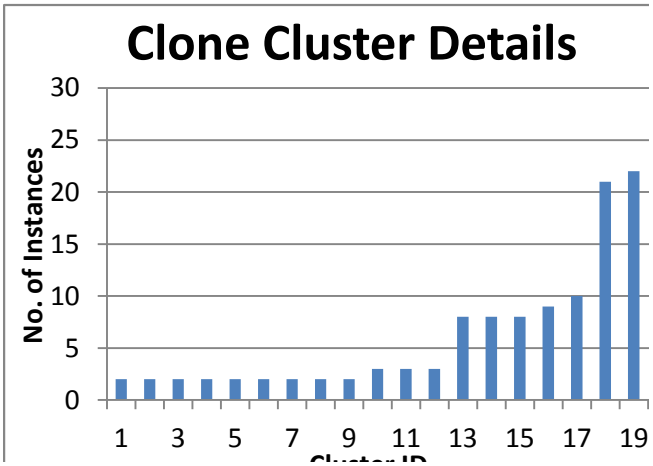

Fig.4 Clone clusters with their instances.

C. Clone Coverage

Clone coverage parameter gives the percentage of clones associated with a class. This parameter helps us to know the extent of cloning in various classes of our subject system. Table 4 shows clone coverage of each class. Clone coverage is calculated from cloned elements to the total elements of the class.

Results of clone coverage, as shown in Table 4 report maximum clone coverage i.e. 90% in the class having class id 7 and minimum clone coverage .04% in class with class id 27. It has been shown that there are four classes without clones i.e 0% clone coverage. Clone coverage helps us to check the extent of cloning in each class.

Table 4. Clone Coverage

Class ID	Clone Coverage
1	53%
2	57%
3	34%
4	48%
5	85%
6	62%
7	90%
8	76%
9	35%
10	69%
11	70%
12	72%
13	72%
14	72%
15	69%
16	76%
17	86%
18	52%
19	62%
20	73%
21	64%
22	73%
23	64%
24	71%
25	54%
26	72%
27	.04%
28	0%
29	0%
30	0%
31	0%

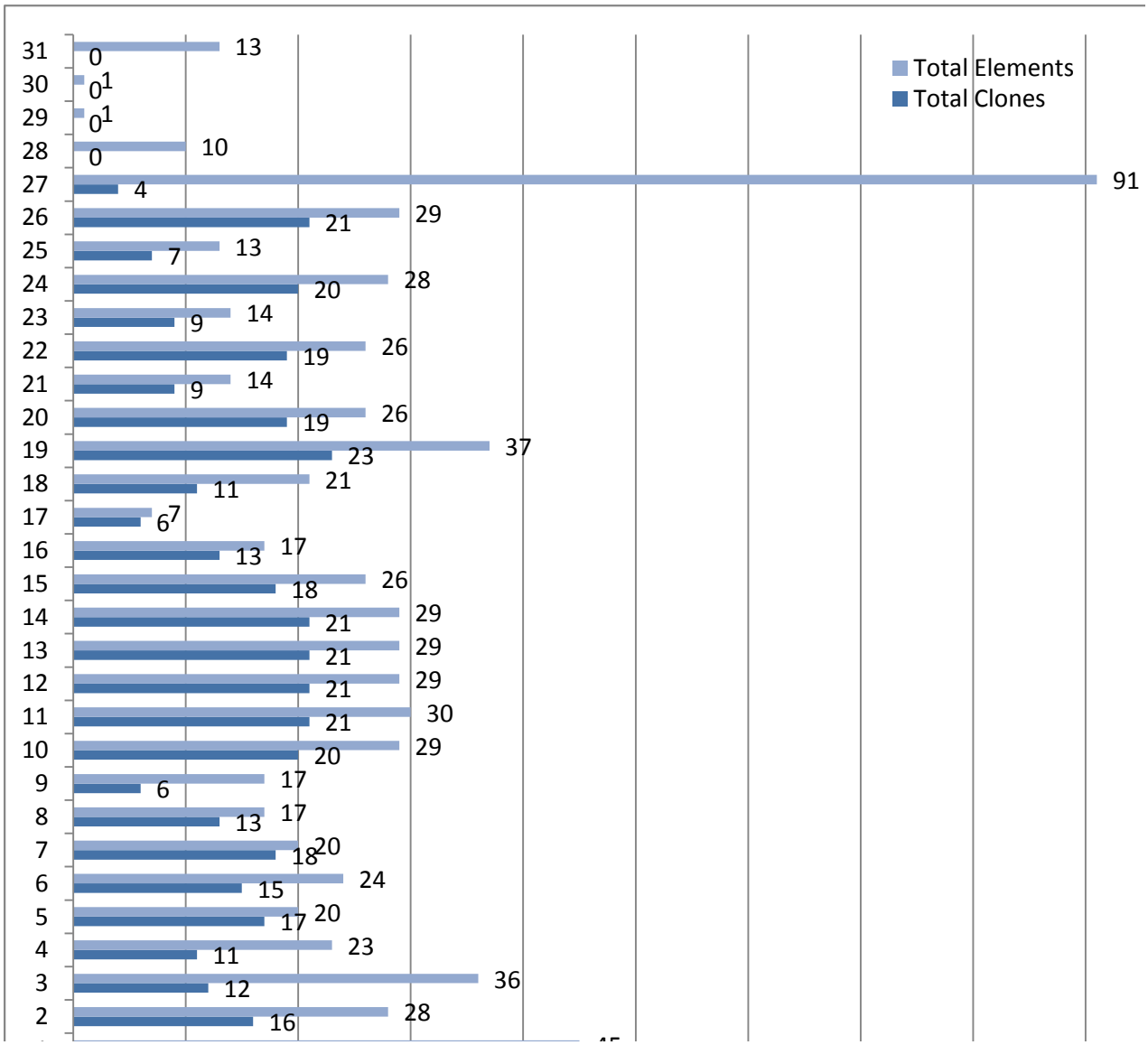

Fig.5 Details of cloned elements w.r.t. total number of elements of class.

D. Class Similarity

Class similarity parameter defines the percentage of similar tokens between two classes. Table 5 gives the detail about percentage of similarity and number of classes lying in that percentage. Maximum numbers of classes are 70 to 80 % similar to each other. There is no class which is having more than 90% similarity.

Table 5.Class Similarity between classes

Class Similarity	No. of Classes
51% - 60%	7
61% - 70%	13
71% - 80%	21
81% - 90%	6
91% - 100%	0

Fig.7 Results summary

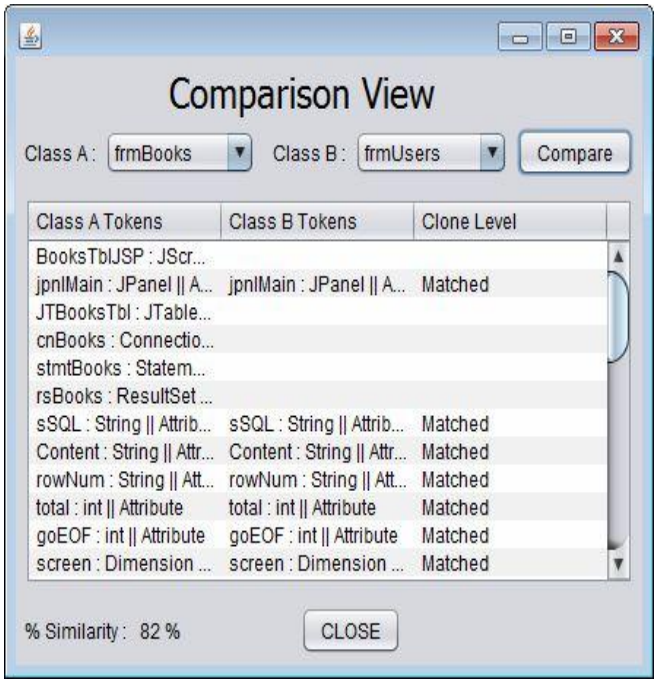

Fig.6 Class Comparison View

Fig.6 is the class comparison view showing the token comparison between two classes and the matched tokens i.e. clones between them.

E. Result Summary

As shown in interface of tool, after completing the clone detection phase, report of results are prepared. Fig.7 is the summary of result showing all the important fields. It presents us with the number of clones, clone ID`s, number of clone instances of each clone, Clusters with their elements and number of instances. The report is very useful to summarize the results.

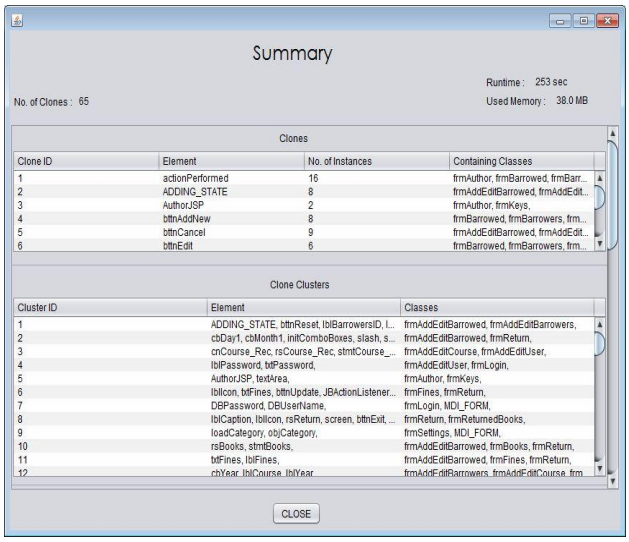

Fig.7 Results summary

F. Memory

This parameter specifies the space used by the clone detection approach.

Used memory: 38 MB.

G. Runtime

This parameter specifies the time taken by the clone detection approach to give various results.

Run time: 253 sec.

V. Related Work

Storrle [20] presented a formal definition of model clones, fragments and clone group. He proposed a clone detection algorithm for UML domain models. According to him, as code clones are problem for code base development, model clones are also increasing problems in model based development. He proposed a model element heuristics and clone detection algorithm based on detailed study of actual model structures. He implemented the approach in MQ_{lone} tool. According to the approach, UML models are seen as a set of heavy nodes that carry major information not similar as graphs with light nodes. Therefore, graph based clone detection cannot be applied to the UML domain model.
He defined four heuristics to find the similarity. First is the Name approach that looks for name similarity only but not provides good detection results. Second heuristic is Name2, it performed best in terms of quality and run time comparing all kind of attributes not just name. Third is INDEX, that defines similarity heuristics to apply on model elements but with shorter identifier but it will not work for long identifiers and XMI models. Last, INDEX-2 is defined which replace all the identifiers with their element name before heuristics are applied. In nutshell, NAME-2 provides the best result in terms of precision and false positives. The study can be applied to all kinds of UML models (e.g. Class diagrams, state machines, activity diagrams). Small and medium sized models works well with the approach, large models are yet to be run on the approach.

Clones exist in software due to copy paste and hard time constraints. Rattan et al. [15] presented the standard literature review on code clone detection. Empirical evaluation of clone detection tools and technique is presented with comparison. They presented the study with nine different types of clones, thirteen intermediate representations and twenty four match detection techniques. The emphasis of the study is to increase awareness of the potential benefits of software clone management. The presented study firstly defines the current status of clone detection in software field. They describe various intermediate representations or transformation techniques with code granularity levels e.g. set of statements, set of tokens, set of methods, set of blocks, set of procedures. Also various match detection techniques are defined with clone granularity level. The most frequently occurring match detection techniques are suffix tree, suffix array, metric and feature vector clustering.

They presented the list of tools with the methodology adopted e.g. (CCFinder takes tokens as input and matching is carried out with suffix tree). Comparison and evaluation of various clone detection tools and techniques are done to identify an efficient clone detector. They enlisted the tool comparison with the outcome or results in terms of precision, recall, no. of candidates, true negatives, false position, rejected candidates. They have extended the study from code clone detection to model based clone detection.

UML models are also affected by cloning. Rattan et al. [16] introduced an approach to detect clones in UML models. According to them, the motive behind this work is to attain high level of abstraction in model driven software process to avoid complexity and high rate of duplication in models due to increasing size of models and to support the study of clone detection in UML models. They have defined the definition, background and general theories on model clones and model clone detection. According to them, reasons of clones in models are copy/paste, language limitation, complexity and time constraints. Their proposed technique is scalable to implement and only relevant tokens are extracted from XMI file and are stored in tree. Only relevant clones are reported as we are storing key elements of UML diagram. Future scope of the study is to extend the prototypical implementation and display the results of clone detection in more user friendly manner. The proposed technique can be extended to large class diagrams, state chart diagrams and activity diagrams [16].

Deissenboeck et al. [4] proposed an approach to automatically identify duplicates in graphic models. The proposed tool is applied on case study of BMW group. They have illustrated various challenges raised when model clone detection is carried out and also presented methods to address these challenges. They proposed model clone detection algorithm, which reports full names of affected model elements. The Clone groups are inspected and accessed to find out the relevance output.

The tool provides the model clone detection with the integrated environment having visualizing effect of detection results. Clone group is easily inspected and highlighted clone instances with their location and extent are shown. They have presented the technique to improve scalability of subsystem, to improve relevance of the detected clones by providing specific ranks and easy clone inspection.

Deissenboeck et al. [5], presented an approach for the automatic detection of clones in large models. Their approach is based on graph theory and is applied to graphical data flow languages. They have taken the industrial case study MIN and demonstrate the applicability of their approach on Matlab/Simulink models. Matlab/Simulink models are widely used in model-based development of embedded systems in the automotive domains. In the case of embedded systems, main part of the code is generated from the domain-specific modeling languages. To support model based development and maintenance, it is very essential to detect clones in models. They proposed solution for the increasing size and complexity of products that relies on model-based development methods. Their technique contains an algorithm and corresponding tool to identify similarity in a model based graphs having weight based filtering heuristics that provide relevant output. Future work is proposed to improve the results by fine tuning the tool and algorithm. Further implementation have to be carried out on larger case studies to get better understanding of strengths and weakness of proposed algorithm.

VI. CONCLUSION AND FUTURE SCOPE

Large adaptability of model based development in software field is promoting model based clone detection. In this work, we are detecting clones in class diagram by using suffix array and analyzing various results of clone detection.

The present work reports that class diagram contains number of redundant elements. Similar attributes or operations present in two different classes are known as clones. The result has shown that there are number of clones present at multiple places in the class diagram. These clones can affect quality of source code generated from that class diagram and hence increases maintainability. The result has shown that there are clone clusters present in class diagram. Clusters are helpful to get idea of clones repeated always together. These clone clusters are good candidates for making super class so that unnecessary redundancy can be removed from class diagram. We report clone coverage which helps to know the extent of cloning in various classes. Maximum clone coverage is 90% and there are 4 classes having no clones. Class similarity is also being calculated which reports that 21 classes of the subject system are 71-80% similar.

So we can conclude that finding redundancy or clones from the class diagram will help the developer to know the extent of cloning in class diagram and clones can be removed to improve the maintenance effort because maximum developers interact with the system through diagrams only. Awareness of clones will help in developing reusable mechanism. Hence it is concluded that detecting clones in models reduces clones in codes which reduces error and maintenance cost.

In future, our present work can be used to explore clone detection in state chart, activity diagram and sequence diagrams. We can also rate reported clones as relevant or irrelevant so that maintainer can have more useful information. Class diagram with large number of classes can also be taken to check the working of our approach. Categorization of clones into various types can also be done.

REFERENCES

[1] H.B. Abdul and S. Jarzabek, “Detecting Higher-level Similarity Patterns in Programs” ESEC-FSE’05, ACM, Lisbon, Portugal, 2005.
[2] H.B. Abdul, S.J. Puglisi, W.F. Smyth, A. Turpin and S. Jarjabek, “Efficient Token Based Clone Detection with Flexible Tokenization” , ESEC/FSE’07, ACM, Cavtat Croatia, 2007.
[3] E.P. Antony, M.H. Alafi and J.R. Cordy, “An-Approach to clone detection in Behavioral Models” Queen’s university, Kingston, Canada, AAC-WCRE, 2013.

[4] F. Deissenboeck, B. Hummel, E. Juergens, M. Pfaehler and B. Schaetz, "Model Clone Detection in Practice", IWSC`10, Cape Town, South Africa, pp.37-44, 2010.
[5] F. Deissenboeck, B. Hummel, E. Juergens, B. Schatz, S. Wagner, J.F. Giard and S. Teuchert, "Clone Detection in Automotive model-Based Development" ICSE` 08,ACM, Leipzig,Germany.pp.603-612, 2008.
[6] R. Falke, R. Koschke and P. Frenzel, "Empirical Evaluation of Clone Detection Using Syntax Suffix Trees", Empirical Software Engineering, vol. 13, no. 6, pp. 601-643, 2008.
[7] B. Hummel, E. Juergens, and D. Steidl, "Index-Based Model Clone Detection", Proceedings of 5th International Workshop on Software Clones, Honolulu, USA, pp-21-27, 2011.
[8] H.J. Lin and L.F. Peng, "Quick Similarity Measurement of Source Code based on Suffix Array", International Conference on Computational Intelligence and Security" ,DOI 10.1109/CIS.2009.175, 2009.
[9] H. Liu, M. Zhiyi, L. Zhang and W. Shao, "Detecting Duplications in Sequence Diagrams Based on Suffix Trees" Software Institute, School of Electronics Engineering and Computer SciencePeking University, Beijing , China.
[10] M. Kaur, D. Rattan, R. Bhatia and M. Singh, "Comparison and Evaluation of Clone Detection Tools: An Experimental Approach" CSI journal of computing, vol.1, no. 4, pp. 44-55, 2012.
[11] M. Kaur, D. Rattan, R. Bhatia and M. Singh, "Clone detection in Models : an Empirical Study" , 3rd IBM Collaborative Academia Research Exchange(I-CARE), New Delhi, India, Oct 2011.
[12] N.H. Pham, A. H, T.T. Nguyen, J.M. Nguyen, Kofahi and T.N. Nguyen," Complete and Accurate Clone Detection in Graph-based Models", ICSE'09,Vancouver,Canada, IEEE, 2009.
[13] H. Petresen, "Clone Detection in Matlab Simulink Models", IMM-M.Sc, Berlin, 2012.
[14] H.C. Purchase, L. Colpoys, M. McGill, D. Carrington and C. Britton, "UML class diagram syntax: an empirical study of comprehension", Australian Symposium on Information Visualization, Sydney,vol.9, 2001.
[15] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review", Information and Software Technology vol.-55, pp.1165-1199., 2013.
[16] D. Rattan, R. Bhatia and M. Singh, " Model Clone detection based on tree comparison", IEEE , 2012.
[17] C.K. Roy, J.R. Cordy and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Science of Computer Programming, vol.74, no. 7, pp. 470-495, 2009.
[18] C.K. Roy, J.R. Cordy and R. Koschke, "An Empirical Study of Function clones in Open Source Software Systems", Proceedings of 15th Working conference on Reverse Engineering, pp-81-90, 2008.
[19] C.K. Roy, J.R. Cordy and R. Koschke, "A Survey on Software Clone Detection Resarch", Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, 2007.
[20] H. Storrle, " Towards Clone Detection in UML domain models", DOI:10.1007/s10270-011-0217-9.
[21] T. Yamashina, K. H.Uwano, Y.Kamei. Fushida, M. Nagura, S. Kawaguchi and H. Lida, "Shinobi: A Real Time Code Clone Detection Tool for Software Maintenance", nasa institute of science and technology.