RESEARCH ARTICLE OPEN ACCESS

A Review: Clone Detection in Web Application Using Clone Metrics

Harpreet Kaur[1], Rupinder kaur[2]
Department of Computer Science and Engineering
Yadavindra College of Engineering (YCOE)
Guru Kashi Campus, Punjabi University
Talwandi Sabo, Bathinda
Punjab-India

ABSTRACT
In software engineering, the concept of code reuse is very common. Code reuse is the concept of copying and pasting the code in multiple places in the same software or different software without modification. In the last few decades numerous code clone detection technique and tools have been proposed for capturing duplicated redundant code, which is also known as software clone. In this study, we propose an efficient clone detection technique which is used to detect clones in various programming language. This method of clone detection can also be implemented to more complex application such as web applications. A tool is developed in JAVA for the system and detects the higher-level clone called Directory Clones in JAVA.
Keywords:- Software engineering, code reuse, code clone detection techniques, higher-level clone.

I. INTRODUCTION

Software clones appear in code due to reasons like:

- Code reuse by copying pre-existing codes
- Coding styles is similar.
- Instantiations of definitional computations.
- Failure to use/identify abstract data types.
- Performance enhancement of a project.
- Accidently using same technique.

The software life cycle comprises of three steps: first we have to clearly define the Requirement implement these requirements; and then we have to maintain the software and evolve it according to user's requirements. But from the development point of view maintenance is the most crucial activity in terms of cost and effort. Code clones are considered one of the bad smells of software system and indicators of poor maintainability. Various studies show that the software system with code clones is difficult to maintain as compared to non-cloned code software system.

Code clones are the result of copy paste activities which are syntactically or semantically similar. The reason behind cloning can be intentional or unintentional.[2] Copying existing code fragments and pasting them with or without modifications into other sections of code is a frequent process in software development. The copied code is called a software clone and the process is called software cloning. Code clone has no single or generic definition, each researcher has own definition. [5]

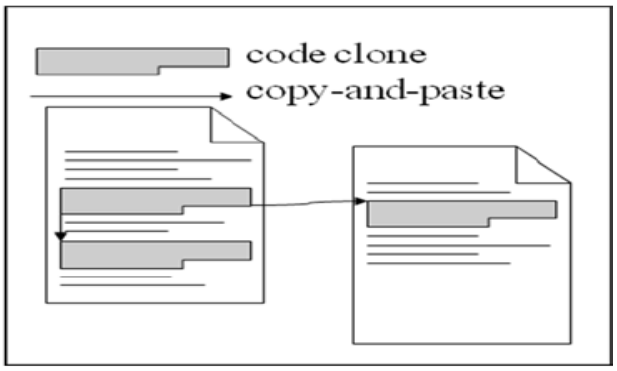


Figure 1.: Code Clone [28]

A. Code Fragment

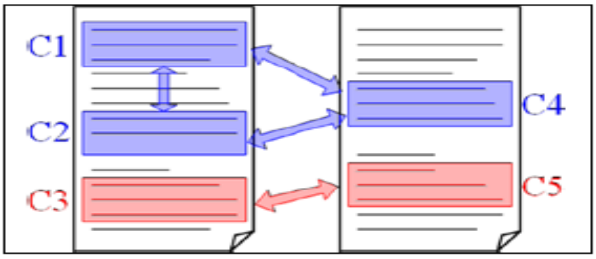
A code fragment (CF) is any sequence of code lines (with or without comments). It can be of any granularity, e.g. function definition, begin-end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base and is denoted as a triple (CF.FileName, CF.BeginLine, CF.EndLine).

B. Code Clone: A code fragment CF2 is a clone of another code fragment CF1 if they are similar by some given definition of similarity, that is, f(CF1) = f(CF2) where f is the similarity function. Two fragments that are similar to each other form a clone pair (CF1; CF2), and when many fragments are similar, they form a clone class or clone group. [3]

C. Clone Pair: A pair of identical or similar code fragments.

D. Clone Set: A set of identical or similar fragments.

A clone relation is defined as an equivalence relation on code portions. For given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. An equivalence class of clone relation is called clone class. That is, a clone class is maximal set of code portions in which a clone relation holds between any pair of code portions.[3]



Clone Pair	Clone Set
(C1, C2)	{C1, C2, C4}
(C1, C4)	{C3, C5}
(C2, C4)	
(C3, C5)	

Figure 2: Clone Pair and Clones Set.[28]

E. Clone Types*:* There are two main kinds of similarity between code fragments. Fragments can be similar based on the similarity of their program text, based on their functionality. The first kind of clone is often the result of copying a code fragment and pasting into another location. In the following the types of clones based on both the textual (Types 1 to 3) [1] and functional (Type 4) similarities are described:

Type-1(Exact clones)*:* Identical code fragments except for variations in whitespace, layout and comments.

Type-2(renamed/parameterized): Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.

Type-3(near miss clones): Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type-4(semantic clones): Two or more code fragments that perform the same computation but are implemented by different syntactic variants.

F. Clone Detection Process:

A clone detector tool must try to find pieces of code of high similarity in a system's source code or text. The main problem is that, it is not known initially which code fragments may be repeated. Thus the detector really should compare every possible code of fragment with every other possible fragment. Such a comparison is expensive from a computational point of view and thus, several measures are used to reduce the domain of comparison before performing the actual comparisons. Even after identifying potentially cloned fragments, further analysis and tool support may be required to identify the actual clones. In this section, an overall summary of the basic steps in a clone detection process is provided. This generic overall picture allows us to compare and evaluate clone detection tools with respect to their underlying mechanisms for the individual steps and their level of support for these steps. Figure 1 shows the set of steps that a typical clone detector may follow in general (although not necessarily). The generic process shown is a generalization unifying the steps of existing techniques, and thus not all techniques include all the steps.

1) Preprocessing*:* At the beginning of any clone detection approach, the source code is divided and the domain of the comparison is determined. There are three main objectives in this phase:

Remove uninteresting parts: All the source code uninteresting to the comparison phase is filtered out in this phase. For example, partitioning is applied to embedded code to separate different languages (e.g., SQL embedded in Java code, or Assembler in C code). This is especially important if the tool is not language independent. Similarly, generated code (e.g., LEX- and YACC-generated code) and sections of source code that are likely to produce many false positives (such as table initialization) can be removed from the source code before proceeding to the next phase [21].

Determine source units: After removing the uninteresting code, the remaining source code is partitioned into a set of disjoint fragments called source units. These units are the largest source fragments that may be involved in direct clone relations with each other. Source units can be at any level of granularity, for example, files, classes, functions/methods, begin-end blocks, statements, or sequences of source lines.

Determine comparison units / granularity*:* Source units may need to be further partitioned into smaller units depending on the comparison technique used by the tool. For example, source units may be subdivided into lines or even tokens for comparison. Comparison units can also be derived from the syntactic structure of the source unit.

For example, an if statement can be further partitioned into conditional expression, then and else blocks. The order of comparison units within their corresponding source unit may or may not be important, depending on the comparison technique. Source units may themselves be used as comparison units. For example, in a metrics based tool, metrics values can be computed from source units of any granularity and therefore, subdivision of source units is not required in such approaches.

2). Transformation: Once the units of comparison are determined, if the comparison technique is other than textual, the source code of the comparison units is transformed to an appropriate intermediate representation for comparison. This transformation of the source code into an intermediate representation is often called extraction in the reverse engineering community. Some tools support additional normalizing transformations following extraction in order to detect superficially different clones. These normalizations can vary from very simple normalizations, such as removal of whitespace and comments [18], to complex normalizations, involving source code transformations [23]. Such normalizations may be done either before or after extraction of the intermediate representation.

(i) Extraction: Extraction transforms source code to the form suitable as input to the actual comparison algorithm. Depending on the tool, it typically involves one or more of the following steps.

Tokenization: In case of token-based approaches, each line of the source is divided into tokens according to the lexical rules of the programming language of interest. The tokens of lines or files then form the token sequences to be compared. All whitespace (including line breaks and tabs) and comments between tokens are removed from the token sequences. CCFinder [23] and Dup [24] are the leading tools that use this kind of tokenization on the source code.

Parsing: In case of syntactic approaches, the entire source code base is parsed to build a parse tree or (possibly annotated) abstract syntax tree (AST). The source units to be compared are then represented as sub-trees of the parse tree or the AST, and comparison algorithms look for similar sub-trees to mark as clones [25].

Control and Data Flow Analysis: Semantics-aware approaches generate program dependence graphs (PDGs) from the source code. The nodes of a PDG represent the statements and conditions of a program, while edges represent control and data dependencies. Source units to be compared are represented as sub-graphs of these PDGs.

(ii) Normalization: Normalization is an optional step intended to eliminate superficial differences such as differences in whitespace, commenting, formatting or identifier names.

Removal of whitespace: Almost all approaches disregard whitespace, although line-based approaches retain line breaks. Some metrics-based approaches however use formatting and layout as part of their comparison.

Removal of comments: Most approaches remove and ignore comments in the actual comparison.

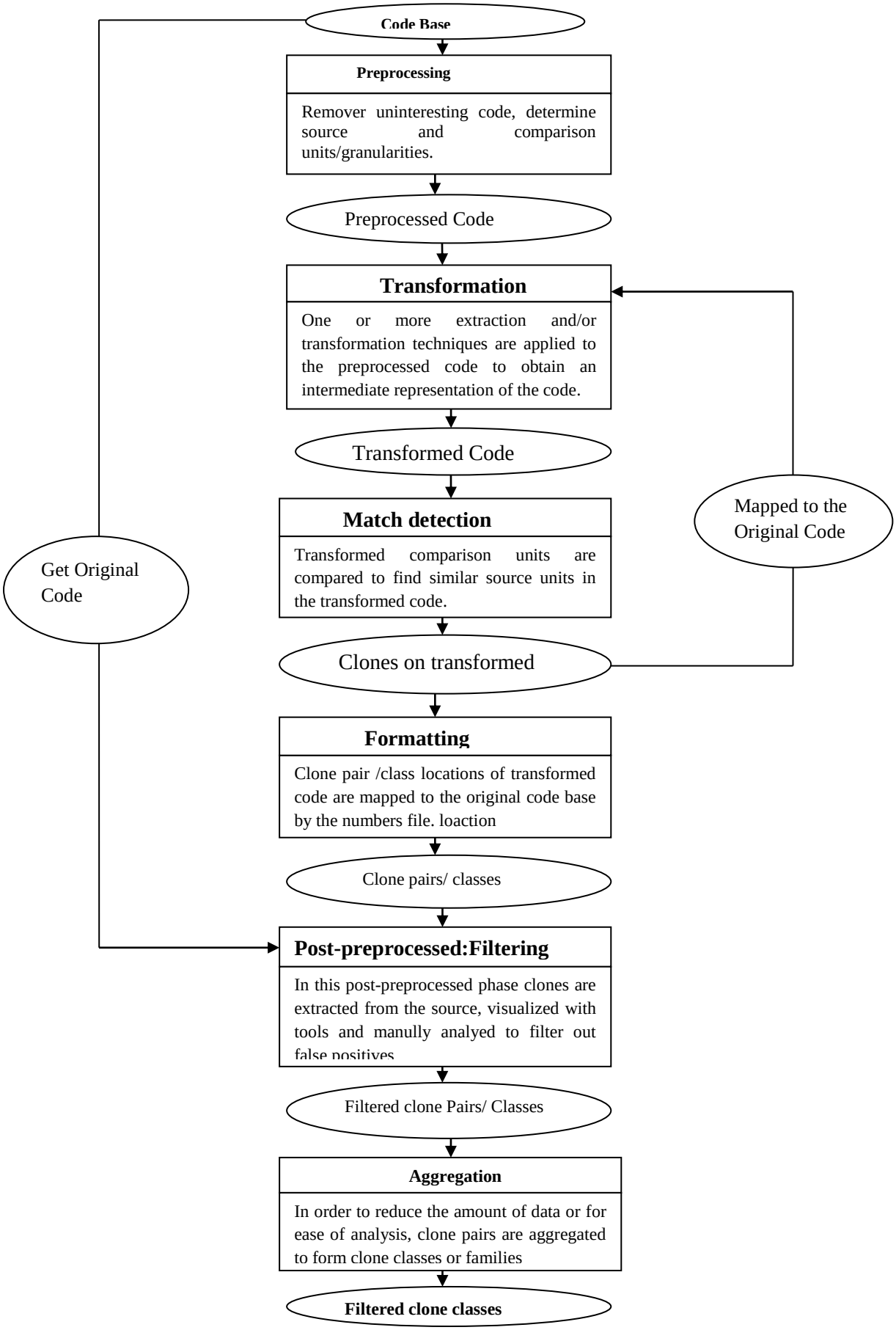
Normalizing identifiers: Most approaches apply an identifier normalization before comparison in order to identify parametric Type-2 clones. In general, all identifiers in the source code are replaced by the same single identifier in such normalizations. However, Baker [22] uses an order-sensitive indexing scheme to normalize for detection of consistently renamed Type-2 clones.

Pretty-printing of source code: Pretty printing is a simple way of reorganizing the source code to a standard form that removes differences in layout and spacing. Pretty printing is normally used in text-based clone detection approaches to find clones that differ only in spacing and layout.

Structural transformations: Other transformations may be applied that actually change the structure of the code, so that minor variations of the same syntactic form may be treated as similar [23].

(iii) Match Detection: The transformed code is then fed into a comparison algorithm where transformed comparison units are compared to each other to find matches. Often adjacent similar comparison units are joined to form larger units. For techniques/tools of fixed granularity (those with a predetermined clone unit, such as a function or block), all the comparison units that belong to the target granularity clone unit are aggregated. For free granularity techniques/tools (those with no predetermined target clone unit) aggregation is continued as long as the similarity of the aggregated sequence of comparison units is above a given threshold, yielding the longest possible similar sequences. The output of match detection is a list of matches in the transformed code which is

Below figure 1:3 show A generic clone detection process



represented or aggregated to form a set of candidate clone pairs. Each clone pair is normally represented as the source coordinates of each of the matched fragments in the transformed code.

(iv)Formatting: In this phase, the clone pair list for the transformed code obtained by the comparison algorithm is converted to a corresponding clone pair list for the original code base. Source coordinates of each clone pair obtained in the comparison phase are mapped to their positions in the original source files.

(v) Post-processing / Filtering: In this phase, clones are ranked or filtered using manual analysis or automated heuristics.

Manual Analysis: After extracting the original source code, clones are subjected to a manual analysis where false positive clones are filtered out by a human expert. Visualization of the cloned source code in a suitable format (e.g., as an HTML web page [23]) can help speed up this manual filtering step.

Automated Heuristics: Often heuristics can be defined based on length, diversity, frequency, or other characteristics of clones in order to rank or filter out clone candidates automatically.

(vi) Aggregation: While some tools directly identify clone classes, most return only clone pairs as the result. In order to reduce the amount of data, perform subsequent analyses or gather overview statistics, clones may be aggregated into clone classes.

II. OVERVIEW OF CLONE DETECTION TECHNIQUES

The area of clone detection has considerably evolved over the last decade, leading to approaches with better results, but at same time with increasing complexity using tool chains. Some existing techniques for clone detection are Textual comparison, Token comparison, Abstract Syntax trees comparison, Program dependency graph comparison, Metrics based comparison. No clone detection tool has been proposed for the detection of all four types of clones. This is a proposal for a new technique for code clone detection, which helps us to detect clones in web application environment made by PHP or JSP. Our proposal is the hybrid combination of metrics based approach and Textual Comparison. [4][29]

A. Textual Comparison: The textual or text-based techniques use little or no transformation on the source code before the actual comparison, and in most cases raw source code is used directly in clone detection process. Though text based approach is the efficient technique but it can detect type 1 clone only. This approach cannot be assured because it cannot detect the structural type of clones having different coding but same logic. Examples: Solid SDD, NICAD, Simian1, DuDe etc.

B. Token Based Comparison:

Lexical approaches (or token-based techniques) begin by transforming the source code into a sequence of lexical "tokens" using compiler-style lexical analysis. The sequence is then scanned for duplicated subsequences of tokens and the corresponding original code is returned as clones. Lexical approaches are generally more robust over minor code changes such as formatting, spacing, and renaming than textual techniques. The technique allows one to detect Type1 and Type2 clones and Type3 clones can be found by concatenating Type1 or Type2 clones if they are toxically not father than a user- threshold away from each other.
Examples:CPFinder Dup,CCFinder etc.

C. Abstract Syntax Tree Based Comparison: Syntactic approaches use a parser to convert source programs into parse trees or abstract syntax trees which can then be processed using either tree matching or structural metrics to find clones. The result obtained through this comparison is quite efficient but it is very difficult and complex to create an abstract syntax tree and the scalability is also not good.
Examples: CloneDr, Deckard, CloneDigger etc.

D. Program Dependency Graph Comparison: Program dependence Graph show control flow and data dependencies. Once the PDG is obtained from the source code, an isomorphic graph comparison is applied to find the clones, and original code slices represented by a sub- graph which are returned as a clone. This approach is more efficient because they detect both semantic and syntactic clones. But the drawback with this approach is that for large software it is very complex to obtain the program dependence graph and the cost is also very high.
Examples: Duplix, GPLAG etc.

E. Metrics Based Comparison:

This approach calculates the metrics from source code and uses these metrics to measure clones in software. Rather than working on source code directly this approach use metrics to detect the clones. Many tools are available for calculating metrics of source code. Columbus is the tool which calculates metrics that are useful in detecting clones, but this tool does not work for Java programs. And

the tool available for the calculation of Java code metrics is Source Monitor but the metrics provided by this tool are not so efficient in providing the result for detection of clones. Other tools that are available for calculating Java code metrics are very complex like Datrix which are designed for extending the quality of Java code. The metrics calculated by this tool are useful for detecting clones in the Java software and it is easy to use too. Metrics are calculated from names, layout expressions and control flow of functions. Metrics-based approaches have also been applied to finding duplicate web pages and clones in web applications.

Table1: Classification of code clone techniques. [29]

Type of comparison	Text based	Token based	AST Based	PDG Based
Category	Textual	Textual	Semntic	Semntic
Supported	Type1	Type 1,2	Type 1,2,3	Type 1,2,3
Portability	Good	Average	Poor	Poor
Integrality	Depends on algorithm	Good	Depends on algorithm	Medium
Efficiency	High	Low	High	High
Complexity	O(n)	O(n)	O(n)	$O(n^3)$
Meaning of (n)	Lines of code	No. of token	Node of AST	Node of PDG

.

III. RELATED WORK

Andrea De Lucia et al. [1] This paper presents an approach for reengineering Web Applications based on clone analysis that aims at identifying and generalizing static and dynamic pages and navigational patterns of a web application. Clone analysis is also helpful for identifying literals that can be generated from a database. A case study is described which shows how the proposed approach can be used for restructuring the navigational structure of a Web Application by removing redundant code. A tool to identify and analyze cloned patterns in web applications using clone analysis and clustering of static and dynamic web pages. The tool has been implemented for WAs developed using PHP or JSP technology. It supports the user to filter out details that do not contribute to the analysis of cloned patterns.

Chancal K.Roy et al. [2] A qualitative comparison and evaluation of the current state-of-the art in clone detection techniques and tools, and organize the large amount of information into a coherent conceptual framework. We then classify, compare and evaluate the techniques and tools in two different dimensions. First, we classify and compare approaches based on a number of facets, each of which has a set of attributes. Second, we qualitatively evaluate the classified techniques and tools with respect to taxonomy of editing scenarios designed to model the creation of Type-1, Type-2, Type-3 and Type-4 clones.

Deepak sethi et al. [3] the code clone or duplicated code is one of the main factors that degrades the design and the structure of software. We can implemented using standard parsing technology, detects clones in arbitrary language constructs, and detects the number clones without affecting the operation of the program. Clone detection can also be implemented to more complex applications such as web based applications. Solid SDD tool provides a way of visualizing clone detection results in a manner that is observably different from the popular visualization using scatter plots.

Gupta et al. [4] to design and implement a Code Clone Detector tool to detect clones. The novel aspect of the work is done by using metric based approach on Java source codes. For calculating metrics Java byte code is used and after that source code refactoring is done in order to reduce code clones. Since the byte code is taken which converts the source code into uniform representation and it is given as an input to the tool for calculating metrics value, so up to some extent it is able to identify the semantic clones. Moreover byte code is platform independent which makes this tool more efficient than the already existing tools. As

abstract syntax tree based approach and program dependence graph approach takes a lot of time and they are complex too for detection of clones so the proposed tool have reduced the work by identifying potential clones with more ease.

James R Cordy et al. [5] The NiCad Clone Detector is a scalable, flexible clone detection tool designed to implement the NiCad (Automated Detection of Near-Miss Intentional Clones) hybrid clone detection method in a convenient, easy-to-use command line tool that can easily be embedded in IDEs and other environments. NiCad is very efficient in its resource usage, and can handle even the largest systems (over 60 million lines) in 2 Gb of memory on a standard single-processor laptop. DebCheck command-line tool. DebCheck can check a system of a few hundred files in less than five minutes on a standard 2 GB single processor home computer. DebCheck will extract all of the C functions embedded in C source files of the system and check every one of them for near-miss clones in the Debian open source distribution, the world's largest packaged collection of open source code.

Saif et al. [6] we have presented a new code clone detection technique. Our technique is capable of identifying clones within large source codes and is distinctive in its ability to detect code duplication independent of the source language. We are also working on some of its future directions including the removal of the clones detected from the source code.

Tariq Muhammad et al [26] We have presented Dynamic web pages composed of inter-woven (tangled) source code written in multiple programming languages (e.g., HTML, PHP, JavaScript, CSS) makes it difficult to analyze and manage clones in web applications. Despite more than a decade of research on software clones, there are not many studies towards the investigation of code clones in web applications. In this paper, we present an in-depth study on the patterns (i.e., forking and templating) of exact and near-miss code clones in two industrial dynamic web applications having distinct architecture. The findings of our study confirm the believed patterns for cloning and suggest that specialized techniques and tool support are necessary for effectively managing clones in the tangled source code of dynamic web applications. We present an exploratory study on the patterns of both exact (Type-1) and near-miss (Type-2 and Type- 3) code clones in two industrial web applications, which underwent two different development styles. One was developed using the traditional style where HTML mark-up and PHP code were put together on dynamic web pages. The other was developed following a more sophisticated approach using the MVC (Model-View-Controller) pattern that resulted in a relatively more modularized implementation.

Y.Ueda *et al.* [7] developed a maintenance support environment based on code clone analysis called Gemini. CC-Finder then represents the information of the detected code clones to the user through various GUIs.

IV. CONCLUSION

Clone detection is live problem in an active search area with plenty of work on detecting and removing clones from software. It usually caused by programmer's copy and paste activates. Code clone detection and removal is still not settled well. In this paper, we conducted a literature review on code clone.

In future we developed tool in JAVA for the system and it detects the higher- level clone called Directory Clones in JAVA. The novelty of this system is that it combines both the metric based and text based techniques in detecting the files clones in JAVA. Various metrics have been formed and their values are used in detection process. If match exists in the metric values then the textual comparison is performed to con firm the clone pair.

REFERENCES

[1] Andrea De Lucia, Rita Francese, Giuseppe Scanniello, Genoveffa Tortora, "*Understanding Cloned Patterns in Web Applications*," Proceedings of the 13th International Workshop on Program Comprehension (IWPC'05), IEEE.

[2] Chanchal K. Roya, James R. Cordy, Rainer Koschke,"Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Science of Computer programming, ELSEVIER, pp 470-495, 2009.

[3] Deepak Sethi, Manisha Sehrawat, Bharat bhushan Naib, "*Detection of Code Clone using Datasets,"* IEEE TRANSACTIONS ON SOFTAWRAE ENGINEERING, TSE-0079-0207,R2.

[4] Girija Gupta, Indu Singh, *"A Novel Approach Towards Code Clone Detection and Redesigning,"* IJARCSSE, pp. 331-338,September-2013.

[5] James R Cordy, Chanchal K. Roy, *"The NiCad Clone Detector & DebCheck: Efficient Checking for Open Source Code Clones in Software Systems,"*19th IEEE International Conference on Program Comprehension, IEEE, 2011.

[6] Salf Ur. Rehman, Ramran khan,Simon Fong, Robert Biuk-Aghai, *" An Efficient New Multi-Language Clone Detection Approach from Large Source Code,"* IEEE 2012.

[7] Y. Ueda, T. Kamiya, S. Kusumoto, K. Inoue, *"Gemini: Maintenance Support Environment Based on Code Clone Analysis",* 8th International Symposium on Software Metrics, pages 67-76, June 4-7, 2002.

[8] Md. Rakibul Islam, Md. Rafiqullslam, Md. Maidull slam, Tasneem Halim, "*A Study of Code Cloning in Server Pages of Web Applications Developed Using Classic ASP.NET and ASP.NET MVC Framework",* Proceedings of 14th International Conference on Computer and Information Technology (ICCIT 2011) 22-24 December, 2011, Dhaka, Bangladesh

[9] Dhavleesh Rattan, Rajesh Bhatia, Maninder Singh,"*Software Clone Detection: Systematic Review,"*Information And Software Techmology, ELSERVIER, pp 1165-1199, 2013.

[10] Z. Li, S. Lu, S. Myagmar and Y. Zhou, *"CP-Miner: Finding copy-paste and related bugs in large-scale software code,"* IEEE, 2006, pp1-2.

[11] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano, *"Extracting code clones for refactoring using combinations of clone metrics".* In Proc. of the IWSC 2011, pages 7-13, 2011

[12] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner,*"Do code clones matter?,"*ICSE'09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 485–495.

[13] Ms. Kavitha Esther Rajakumari, Dr. T. Jebarajan, *"A Novel Approach to Effective Detection and Analysis of Code Clones,"* IEEE, 2013.

[14] Fabio Calefato, Fillppo lanubile, Teresa Mallardo, *"Function Clone detection in Web Application:A Semi automated Approach,"* Journal of Web Engineering, Vol.3, No.1, pp.003-021, 2004.

[15] C. Kapser, M.W. Godfrey, *Supporting the analysis of clones in software systems: research articles,* Journal of Software Maintenance and Evolution 18 (2) (2006) 61–82

[16] S. Thummalapenta, L. Cerulo, L. Aversano, M.D. Penta, " *An empirical study on the maintenance of source code clone*", Empirical Software Engineering 15 (1) (2010) 1–34.

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

[18] Rainer Koschke, *"Survey of Research on Software Clones",* Dagstuhl Seminar Proceedings.

[19] Mohammed Abdul Bari, Dr. Shahanawaj Ahamad, *"Code Cloning: The Analysis, Detection and Removal",* International Journal of Computer Applications (0975 –8887) Vol. 20, No.7, April 2011.

[20] G.Anil kumar, Dr.C.R.K.Reddy, Dr. A. Govardhan, Gousiya Begum,*" Code Clone detection with Refactoring support Through Textual Analysis,"* International Journal of Computer Trends And Technology- Volume 2 Issue2-2011.

[21] M. Rieger, "*Effective Clone Detection without Language Barriers",* Ph.D. Thesis, University of Bern, Switzerland, 2005.

[22] B. Baker, "*A Program for Identifying Duplicated Code",* in: Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, 24:49-57 (1992).

[23] C.K. Roy and J.R. Cordy, "*NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty- Printing and Code Normalization*", in: Proceedings of the 16th IEEE International Conference on Program Comprehension, ICPC 2008, pp. 172-181 (2008).

[24] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, *"Comparison and Evaluation of Clone Detection Tools"*, Transactions on Software Engineering, 33(9):577-591 (2007).

[25] I. Baxter, A. Yahin, L. Moura and M. Anna, *"Clone Detection Using Abstract Syntax Trees*", in: Proceedings of the 14th International Conference on Software Maintenance, ICSM 1998, pp. 368-377 (1998).

[26] Tariq Muhammad, Minhaz F. Zibran, Yosuke Yamamoto, Chanchal K. Roy, *" Near-Miss clone patterns in web applications : AN Empirical study with Industrial Systems",* 2013 26th IEEE Canadian Conference Of Electrical And Computer Engineering (CCECE),2013 IEEE.

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

[28] C.K. Roy, J.R. Cordy, "Near-miss function clones in open source software: an empirical study, Journal of Software Maintenance and Evolution:" Research and Practice 22 (3) (2010) 165–189.

[29] Prajila Prem, *"A Review on Code Clone Analysis and Code Clone Detection."* International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, *Issue 12, June 2013.*

[30] *T. Kamiya, S. Kusumoto and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Transactions on Software Engineering, 28(7):654-670 (20)*