Volume 4, Issue 7, July 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Clone Detection in Web Application Using Clone Metrics

Harpreet Kaur
Department of CSE,
YCOE, Punajbi University,
GKC, Talwandi Sabo, India

Rupinder Kaur
Department of CSE,
YCOE, Punajbi University,
GKC, Talwandi Sabo, 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, web Application, higher-level clone.

I. INTRODUCTION

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].

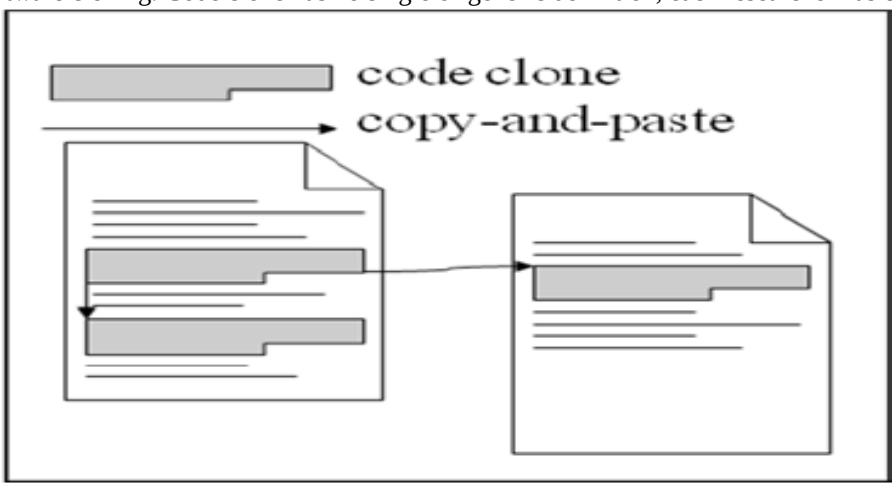


Fig-1 Code Clone

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) [3].

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].

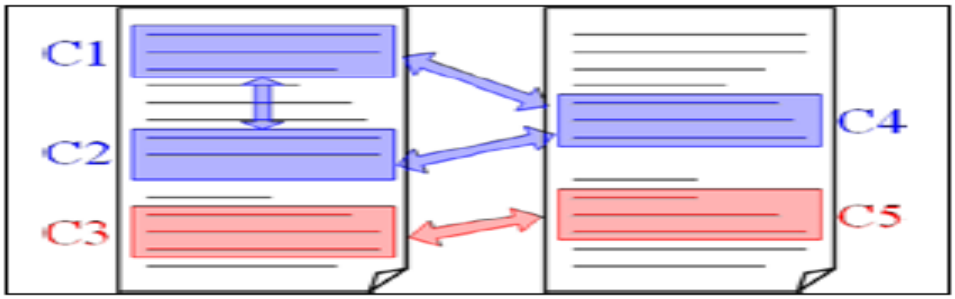
C. Clone Pair:

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.



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

Fig-2. Clone Pair and Clones Set

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 [3].

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. [4][5]

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 [20].

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 [21], to complex normalizations, involving source code transformations [22]. 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 [24] and Dup [23] 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 [21] 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 [22].

(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.

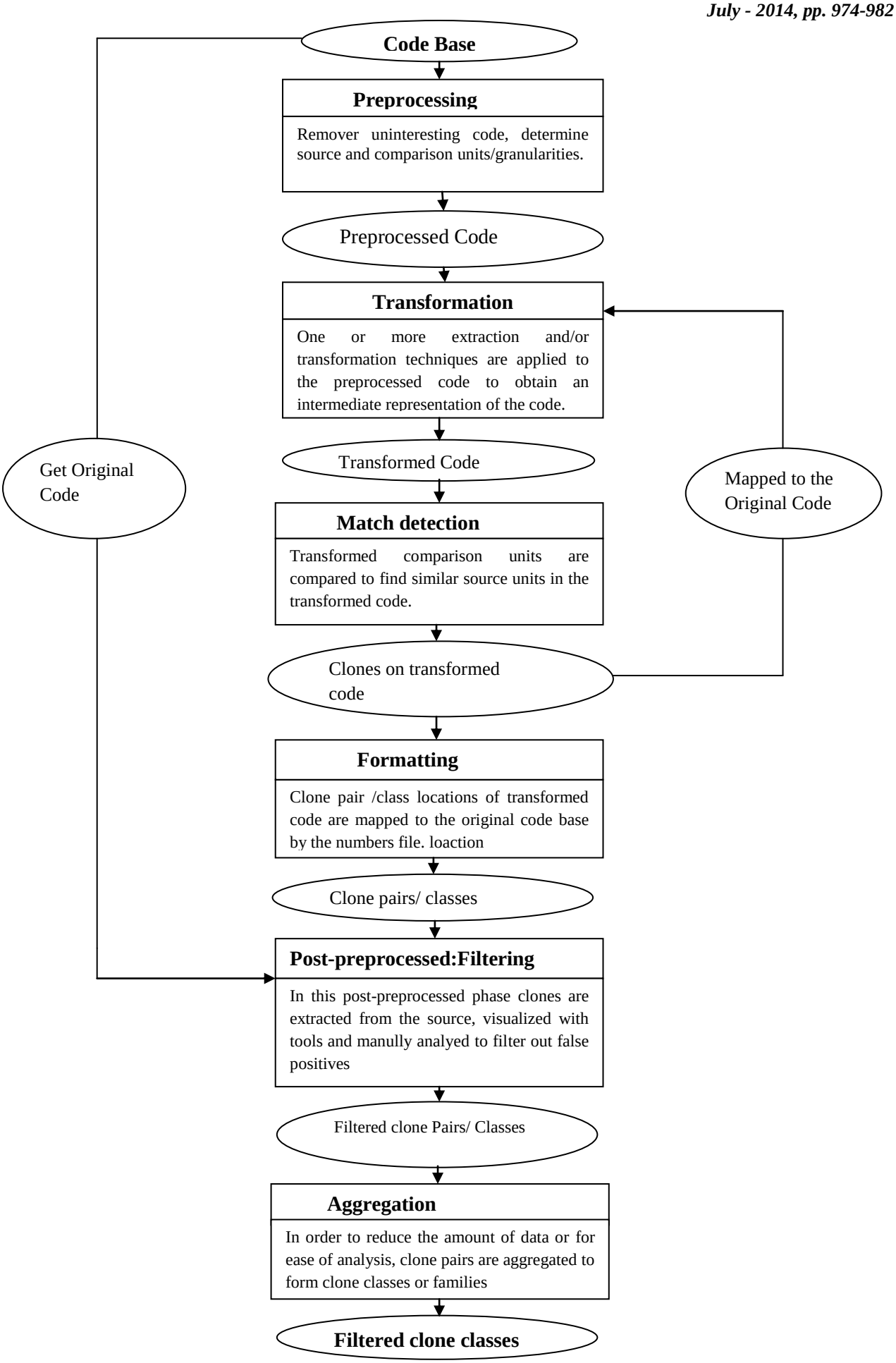


Fig-3 A generic clone detection process.

The output of match detection is a list of matches in the transformed code which is 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 [22]) 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. ROBLEM FORMULATION

Code clone detection in software testing, this can be included in software testing in bug detection. The previous research shows that the detection system carries non user friendly interface and is hard to maintain the SDLC. Further the disadvantage is on small projects, the detection system only relies on big software testing. We are going to implement a software testing tool which detects the code clone. Our research follows to develop a tool which is user friendly and is easy to maintain also and is not limited to small and big software. The debugger will tell the user for solution to be carried out and tells if there any clone exists in system or not as clone code could lead to big risk issues. This method of clone detection can also be implemented to more complex applications such as web based applications. i.e a website code related to PHP or JSP or it can be a application which is linked with internet not a standalone application. We will detect code clone in web based environment.

III. OBJECTIVE

1.) To analyze the behaviour of source code of software.
2.) To perform and analysis of how java language detects clone of web based application made on PHP, JSP.

IV. RELATED WORK

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. [13]

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.[1][12]

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. [3]

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[3].
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.[3][13]

V. PROPOSED WORK

This research will focus on providing solution for said problem: - **Simple line matching** (representative for the string-based techniques) gives a crude overview of the duplicated code that is quite easy to obtain, hence is most appropriate during problem detection and problem assessment.
The objective of the system is to detect the functional similarity between directories having JAVA files. This is done by using identified metrics. A tool is developed 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.

We have developed the user friendly interface of hybrid clone detection by using one of most commonly used programming language i.e. Jav. The code is based on the algorithm as given below which will test the PHP code from

```
file #1 and file #2
Read file#1
Read file#2
Compute loc1, CC1, f1n1, f1N1, f1n2, f1N2
Compute loc2, CC2, f2n2, f2N2, f2n2, f2N2
If (loc1==loc2||CC1==CC2|| f1n1==f2n2
||f1N1==f2N1||f1 n2==f2n2||f1N2==f2N2) Then
While not EOF (File1)
 Read line fl1
 While not Eof (file2)
 Read line fl2
 If (fl1==fl2) Then  Hybrid line Clone Detected
 Print line fl1
 Exit
 Else
 Read next line
 End if
 While end
 Read next line
 While end
Else
Msg box “Metrics does not match”
End if
```

In this algorithm, the line of code of first application program is compared with line of code of second application program. Here file#1 and file#2 are used to indicate first and second application program respectively. In this algorithm we are comparing every line of code of first application program with each line of second application program. But before actual comparison, the metrics of two application program are computed and based upon the analyzed metrics the comparison is carried out. So in this case metrics is of major consideration. After computation of the metric the actual comparison is carried out that will select the cloning of line code if exist.

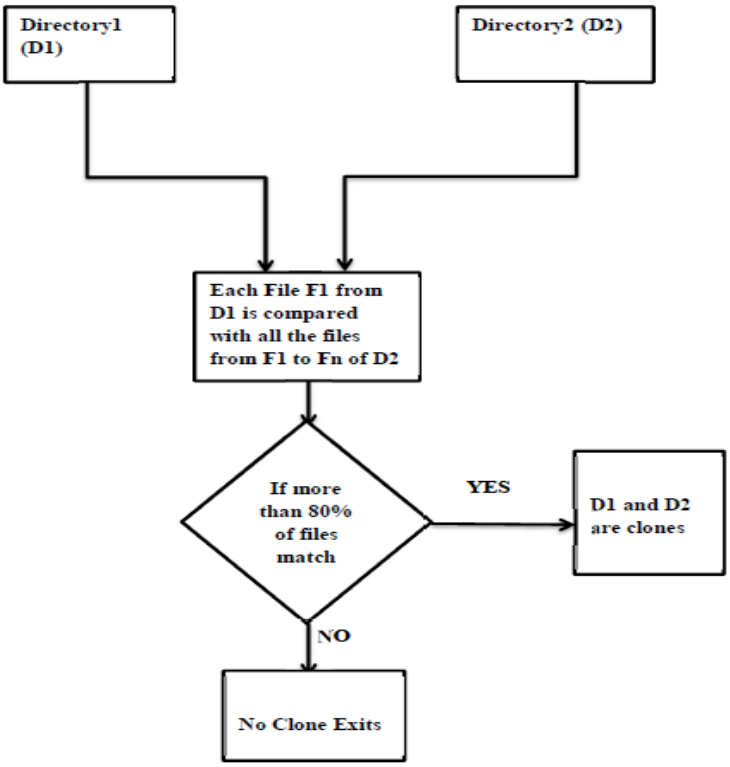


Fig.4. Code Clone Detection Process in Java file

VI. RESULTS AND DISCUSSION

First we developed a web application project using PHP, JSP. Then load this project in java Tool. Then choose the two files of PHP project and detect the code clone. It shows the common code between the two files.

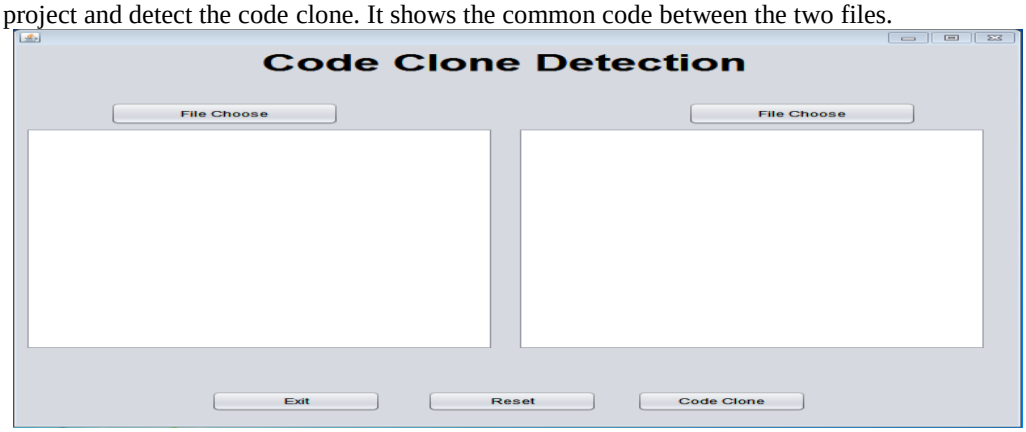


Fig-5 Start-up Page of code clone detector tool

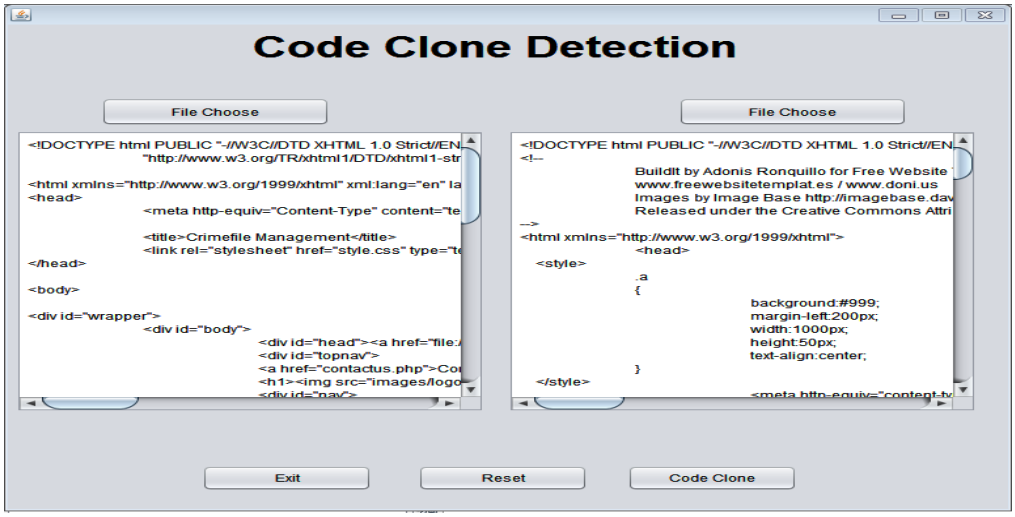


Fig-6 Selecting files and giving it as an input to tool.

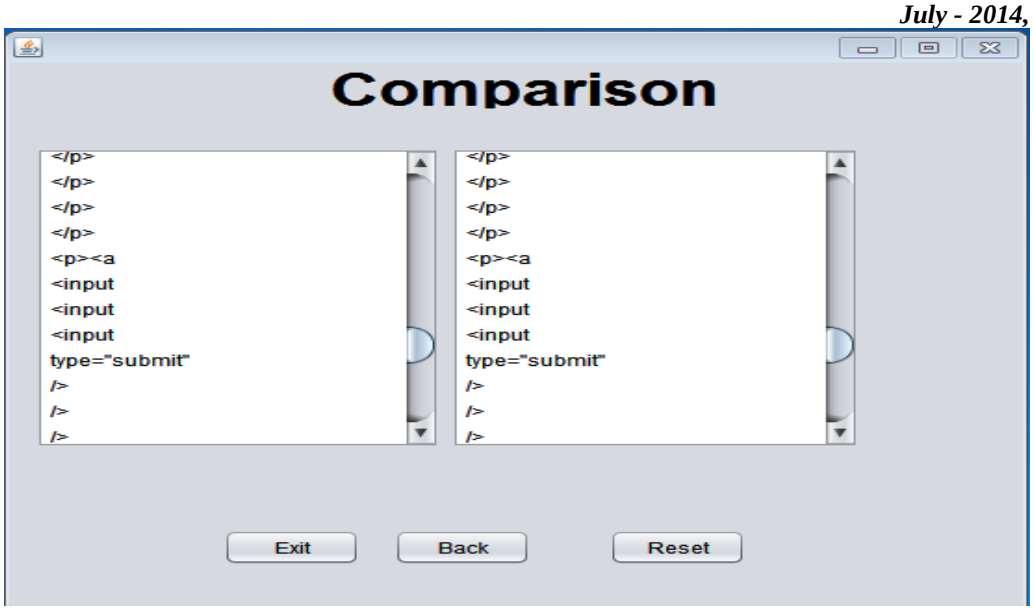


Fig-7 Comparison of two files of PHP project and show common code

VII. CONCLUSION AND FUTURE SCOPE

After successful implementation of this thesis it is conclude that code clone detection carried out as a successful approach for finding the clones in source code of particular language. Since this work of thesis the tool created is language dependent which supports every language's source code and detects the clones according line by line wise. Further this language independent tool is successful for detecting small projects source code clone which was a major issue in most of the previous language code clone detection tools. No doubt to help out large number of small scale and medium sized industries towards their growth and productivity of their products. Since code clone is also a major problem in today's industries as developers can copy code by reverse engineering process and copy and pasting codes from previous things which they had worked on, this tool helps to detect the clone in every aspect. Lastly the time analysis was carried out to test the tool how much time it would take to detect the clone w.r.t line of code. For future perspectives this analysis can be further extended another language since this tool is developed in java, it would be interested to know how this tool behaves if languages is changed from java to dot net. It would be also interesting to know how another algorithmic approach behaves with comparison along with heuristic methods.

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] Kanika Raheja, Rajkumar Tekchandani, "*An Emerging Approach towards Code Clone Detection: Metric Based Approach on Byte Code*," IJARCSSE, Vol.3, May 2013.

[3] Prajila Prem, "*A Review on Code Clone Analysis and Code Clone Detection,"* IJEIT, Vol.2, Issue 12, June 2013.

[4] 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.

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

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

[7] 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.

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

[9] 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.

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

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

[13] 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.

[14] 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

[15] 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.

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

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

[18] 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.

[19] 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.

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

[21] 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).

[22] 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).

[23] 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).

[24] 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 (2002).

[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).

