Abstract—Software systems are getting more complex as the system grows where maintaining such system is a primary concern for the software industry. Code clone is one of the factors making software maintenance more difficult. The copy of code fragments and then reuse by pasting with or without minor modifications or adaptations and this type of reuse approach of existing code is called code cloning and the pasted code fragment (with or without modifications) is called a clone of the original. Code clones are common activities in software development. The major risk of cloning is that it risks the maintenance process. Cloning is basically the means of software reuse and software reuse is the basic need of today’s environment. That is why code cloning has been extensively used in large software industries. So to detect clones and refactor them is a major concern. Clones are considered harmful for software maintenance and evolution because it increases the complexity. Moreover, too much cloning increases the system size and often indicates design problems. Detection of clones decreases software maintenance cost and increases understandability of the system. Many approaches have been developed to detect clones and the most common among all is text based comparison and token based techniques. Though these techniques are efficient but they take lot of time and are expensive too and the tree based and AST based techniques are very complex to deal with. The aim of the proposed work is 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 inorder to reduce code clones.

Keywords—Code clone, Clone detection, metric calculation, redesigning, class metric, function metric

I. INTRODUCTION

Code cloning is found to be a more serious problem in industrial software systems. Code clones may adversely affect the software systems’ quality, especially their maintainability and comprehensibility. For example, cloning increases the probability of update anomalies (inconsistencies in updating). Moreover, too much cloning increases the system size and often indicates design problems. Copy fragments of code which are exactly the same as or similar to each other are called code clones, i.e., Clones are often the result of copy-paste activities. Such activities are very easy and can reduce programming effort and time. There is “accidental cloning” also but it is not the result of direct copy and paste activities but by using the same set of APIs to implement similar protocols. Moreover, the error free code becomes buggy after cloning[10]. Now attempts are being undertaken to detect clones, and once identified, they can be removed through source code refactoring. Clones are classified on the basis of textual and functional similarity [2][3]. The detection of all these types of clone varies according to the clone detection techniques used. For example, text based approach can detect type1 clone only [4]. Token based approach detects type1 and type2 clones. These techniques cannot detect type3 and type4 clones. Generally for detecting type3 clones Abstract syntax,Tree based approach [6] and for detecting type-4 clones Program Dependence graph approach [7] is used. Another technique for clone detection is a metric based approach that calculates metrics of source code inorder to detect clones.

The process of clone detection can be done manually by comparing lines one by one but for the large program this process will become very time consuming. Hence a tool should be designed for detecting the clones because it is very important from the maintenance point of view. Various tools had been proposed for the detection of clones that are based on the different algorithms [1][9].

A clone detection system should give precise and useful information about clones present in code since a large number of clones are expected to be found in large software system. In the proposed work metric based approach for detecting potential clones is used as all the other technique requires a large amount of comparison and time. Potential clones are detected with the help of this approach and by then redesigning of code can be done to reduce clones to make the system more effective. Section II describes the related work. Section III illustrates how the tool is implemented for Java program and section IV illustrates the results. Section V describes the conclusion and the future scope.

II. RELATED WORK

Clone Detection is an active research area since 1990’s [1][11]. Code clone detection is directly related to maintenance of software, code redesigning and thus making the code more efficient. The survey on clone detection shows the various techniques and algorithms to detect clones [4][12]. Yue Jia et al. [13] proposed that the algorithm used to detect clones should be precise and efficient. Chanchal K. Roy et al. [4] in his survey paper broadly classified the
various techniques for code clone detection. Clone detection approaches are broadly classified into five techniques [4] which are described below:

A. Textual comparison
The Textual approach uses no transformation of the source code before applying the comparison, and source code is used directly in the clone detecting process. Though text based approach is the efficient technique but it can detect type I clones only [12]. This approach cannot be assured because it cannot detect the structural type of clones having different coding but same logic.

B. Token based comparison
Token Based Approach needs a parser or lexer to normalize the code in the form of the tokens. First of all source code is transformed in the form of tokens by using parser or lexer used to convert the code into tokens, then the comparison is applied on the changed code (intermediate representation). This technique is more efficient as compared to the text based approach if blank spaces and comments are present in the source code. But it is not accurate because while conversion of source code in the token sequence various false positive may introduce in the code. Various tools are proposed for clone detection that is based on the token based approach. Toshihiro Kamiah et al. [5] proposed a token based approach tool named CCFinder but this technique requires a parser to transform the code into tokens.

C. Abstract Syntax Tree Based Comparison
Abstract syntax tree based approach converts source code into an abstract syntax tree and traverse the tree for finding a similar sub tree. If similarity is found then the code for this sub tree is termed as clone [14]. 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.

D. Program Dependency Graph Comparison
Program Dependence Graph shows control flow and data dependencies. Once the PDG is obtained from the source code, an isomorphic graph comparison [7] is applied to find the clones, and the 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.

E. Metric 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 [8]. 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 [15]. In the presented work a tool for metric based clone detection is proposed. The metrics calculated by this tool are useful for detecting clones in the Java software [15] and it is easy to use too.

There are basically two kinds of similarities between two code fragments. Two code fragments can be similar based on the similarity of their program text or they can be similar in their functionalities without being textually similar.

Textual Similarity : Based on the similarity of the text we have the following types of clones :

Type I: code fragments that are identical except for changes in whitespace and comments.

Type II: Structurally identical code fragments except for the changes in identifiers, literals, types, layout and comments.

Type III: Copied fragments with modifications. Statements can be modified, added or removed in addition to variations in identifiers, literals, types, layout and comments are called type III Clones.

Functional Similarity : If the functionalities of the two code fragments are identical or similar we call them semantic clones and can be referred as Type IV clones.

Type IV: Two or more code fragments that perform the same function but are implemented through different syntax are called type IV clones.

III. PROPOSED WORK
The present method is based on a metric based approach for clone detection. First of all the metrics are calculated from this tool and then the comparison algorithm is applied on these metrics to detect clones and after detection of clones redesigning of code is performed to reduce the clones. This method consists of the following four steps :
Step 1: Giving input(byte code) to Code Clone Detector
Step 2: Calculation of Metrics
Step 3: Comparison of files for finding potential clones
Step 4 : Redesigning of code

A. Giving Input to Code Clone Detector

In the first step the code is adapted according to requirement of tool means tool requires as input the .class file for the program. So the Java source code should be compiled to make it adaptable according to tool’s requirement. The tool takes a byte code as an input program rather than the Java source code because the byte code is platform independent. Rather than using AST and PDG approach to detect type-3 and type-4 clones which are quite complex, the proposed approach tries to detect clones by applying metric based technique. The perception is derived from the fact that while converting the code into byte code the compiler would map ‘for’ and ‘while’ loops into the same unified representation. As compiler will generate a unified representation of code by reordering, and this unified representation reduces dissimilarity in code segments, so it yields better results. The first page that is opened is FileChooserDemo in which .class file is chosen for calculating metrics. The first button opens the file and the second button is provided to delete the previous data that is present in the database because Code Clone Detector provides a provision to save the calculated metrics in a database.

B. Calculation of Metrics

After giving the input to the tool, the next step is to perform calculation of metrics. It is in the form of various metrics that helps in identifying the potential clone. Bruno Lague et al. [15] provided various metrics to perform Java software evolution and clone detection and these metrics are calculated with the help of this tool. Besides these metrics some object oriented metrics are also calculated from the proposed tool that helps in identifying potential clones. The metrics that were listed by Bruno Lague et al. [15] is calculated. The metrics that are calculated with the help of proposed tool are in the form of class metrics and function metrics are:

![Fig. 1 Startup page of Code Clone Detector tool](image1)

![Fig. 2 Selecting Student.class and giving it as an input to tool](image2)

![Fig. 3 Selecting Employee.class and giving it as an input to tool](image3)
Class Metrics
1. Total Number of functions in class
2. Total Number of if statements in class
3. Line of Code
4. Total Number of variables in class
5. Total Number of public variable in class
6. Total Number of private variables in class
7. Total Number of protected variables in class
8. Total Number of friend variables in class

Function Metrics
1. Name of functions present in a class
2. Number of variables present at function level
3. Total numbers of lines in a function
4. Return type of function
5. Number of arguments passed to function
6. Number of times a function is called

When we open a file the metrics calculated will be stored in a database. These calculated metrics are mapped into the excel sheets. Then mapping to excel sheet is made to store the data for future use and comparison is applied to these excel sheets and the calculated metrics of Student.class is shown in Fig 4 and Fig 5. Metrics are calculated both at the function level and class level and Fig 6 and Fig 7 shows calculated metrics of the second file Employee.class.

![Fig. 4 Metrics at Class Level of Student.class](image1)

![Fig. 5 Metrics at Function Level of Student.class](image2)

![Fig. 6 Metrics at Class Level of Employee.class](image3)
C. Comparison of files for Finding Potential Clones

After the calculation step is completed and the calculated metrics are mapped into excel sheets then the comparison is performed to select potential clones. The comparison is performed on the basis of similarity of the metric value of both files. This Code Clone Detector tool represents as output which metric’s value is same and on the basis of these results potential clones are identified.

Fig. 7  Metrics at Function Level of Employee.class

Fig. 8 Browsing the two excel files for which comparison is performed

Fig. 9 Comparison of Class Level Metrics of both files

After loading the files in buffer, there is an option to compare the files. Fig.9 shows the comparison of various class level metric on the basis of similarity. Similarly the function level metric is calculated and then compared to find the similarity as shown in the Fig.10. After analyzing the results obtained from a comparison of metrics at class level and function level, potential clones are detected.

Fig. 10 Comparison of Function Level Metrics of both files
If Clones are present in a particular field of class level or function level metrics after comparison of both files then that field shows, the no of clones present and if there is no clone then it is represented by blank field. For example at Class Level comparison of .xls files the clones detected are in the Total Function, proV and TotalV fields and at Function Level two functions Loan Taken and main functions are detected as clones.

D. Redesigning of Code

After identifying the cloned method then redesigning is applied. The bodies of the cloned methods are replaced by method calls to the new method Cloned Method. Those method calls will forward to Cloned Method the requests originally handled by the clones. Finally the new source files are generated in which clones are much reduced.

After the third step of comparison of two files Employee and Student at class level and function level now redesigning of source code of both files is done by making Loan as a separate class and defining LoanTaken as a static function and calling this function in Employee and Student class with the name of its class Loan by dot operator by doing this number of clones detected are reduced. The new files generated are named as Employee1.java and Student1.java.

IV. RESULTS AND DISCUSSION

The files written in Java language are taken as input. Moreover byte code is used as input to the tool and byte code uses the same unified representation. As compiler will generate a unified representation of code by reordering and this unified representation reduces dissimilarity in code segments, so it yields better results. So up to some extent the tool will be able to detect semantic clones as well. Now metric based approach is applied to detect potential clones. The class metric’s value obtained through this tool are shown in fig 13. The function metric’s value are shown in fig 14.
V. CONCLUSION AND FUTURE SCOPE

The tool Code Clone Detector developed works only for the Java language code and it is easy to use. In this presented work metric based approach is used to identify the potential clones and after that redesigning of code is done to reduce the detected 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 metric’s 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.

In future this approach can be integrated with other approaches to make this a hybrid approach to efficiently detect semantic clones and make it more efficient and cost effective. This work can be extended to other languages of interest.

ACKNOWLEDGMENT

We would like to express our gratitude to all those who gave their support to complete this paper. We express our gratitude to JCD college for providing us labs to complete our work and all the faculty members who supported us.

REFERENCES


