A Light Weight Hybrid Approach to Detect Code Clones

Dr.G.Anil Kumar

Sr. Assistant Professor CSE MGIT Hyderabad T.S. India

Abstract

Code cloning is the process of reusing the code portions of different parts of the same project or proven good portions of different projects. The harmfulness of code clones are presented by the literature of the code cloning. In this paper we are proposing a hybrid light weight approach to detect various types of clones.

CLONE DETECTION PROCESS

Several approaches are proposed by different people and organizations in code clone literature. In this paper the proposed clone detection process is discussed in detail. In other words this paper explains theoretical model of the proposed system.

Clone Detection Process: Proposed Model

In a system's source text, a clone detector locates pieces of code which are of high similarity. The major difficulty is that it is hard to make out beforehand about the code fragments that can be repeated multiple times. Hence, each possible fragment should be compared to every other possible fragment. This is an expensive comparison from the view of computational time. Many measures have been taken to reduce the comparison domain before the performance of the actual comparison.

If the potential cloned fragments are located future analysis is taken up to identify the actual clones. In the proposed method, to make out all varieties of clones which are present in the source code, a hybrid technique is adapted. This approach is based on textual and metric analysis.

Clone detection technique which is text based utilizes the transformation. This includes removal of comments and removal of whitespaces. It is considered as one of the fastest clone detection approaches. This is because the syntactical or semantical analysis of source is not performed by the text based technique. It deals easily with type I clones and type II clones with additional data transformation. Metric based technique follows a different method. Instead of making comparisons directly on the code, metric based technique gathers different metrics of code and makes the comparison of these metric values to detect clones. For detecting similar codes, various clone detection techniques use only metrics these days. The proposed method can be implemented to find clones in JAVA. The method which has system architecture is shown in Figure 3.1

Metric based and text based techniques are used by the proposed approach in order to detect clones. These clones are divided into two stages. For the selection of potential code metric based technique is used in the first stage. In the second stage, metric match is used to select potential clones. After the selection, the potential clones are processed further along with text based technique. The proposed research methodology aims to trace out clones with textual analysis using metrics. Clone detection process can be either textual or one that has metric analysis traditionally. The comparison is done line by line for potential clones in order to determine if two potential clones are truly clones of each other. Soon after the beginning of the process, the method developed examines the given input source code and recognizes the different methods which are present.

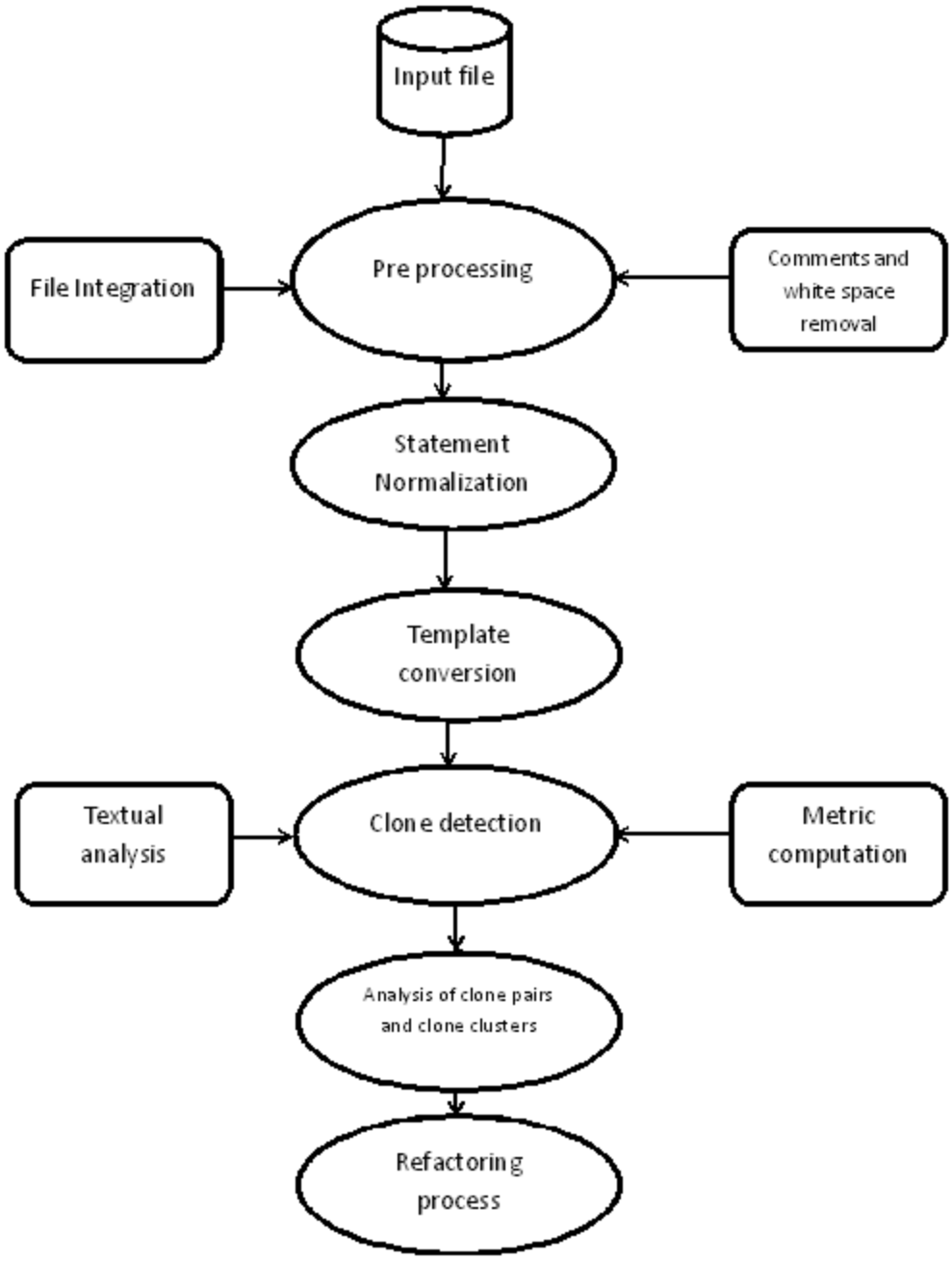

Figure 3.1: Clone Detection Architecture (block diagram)

There are many phases in clone detection process. These phases involve input and pre-processing, conversion of template, computation of metrics and lastly detection of clone types. If the pairs show similarity when textual comparison is done, those pairs are listed as clones. The developed method for detection does not use any external parsers. In comparison to the methods, it needs less overhead when it comes to complexity, and processing time in terms of precision and recall values.

Preprocessing and Input Selection

In this phase the source code which is uninteresting for the comparison phase is filtered. File integration, standardization of source code and source code normalization are included in this phase. The grouping of all the files of a project into one large single file for external examination is done in file integration. The concatenation of all the files which belong to the same project is transformed into one large file convenient for external parsing in file integration.

This involves white space removal, comments and pre-processor statements. Once the uninteresting code is removed, the remaining source code is divided into source units. Aset of disjoint fragments are the source units. Thelargest source code fragments which are directly involved in clone relation with another clone fragment are the source units. The presence of source units can be of any level of granularity. Examples are classes, files, methods or functions, sequences of source lines of code, statements and blocks of code.

Source units need to be further divided into smaller units. This again depends on the comparison technique which is used by the method. For instance, source units can be further divided into lines or even tokens for comparison. The comparison units might be derived from the syntactic structure of the source unit. For instance, an '*if*' statement might be divided into conditional expression and else blocks.

It may be or may not be prominent the way in which the units are compared (i.e. order of comparison) within its corresponding source unit. This process depends on the comparison techniques used. Sometimes source unit as a whole can be utilized as a comparison unit. As an example the metric values of a metric based method might be computed from other source units. Henceforth subdivision is not at all required in those kinds of approaches. In a standard format the source code is restructured in order to maintain the similarities between the cloned fragments.

These steps are akin to normalization procedure and come out with gain in the recall value. Whitespace is disregarded in almost every approach, despite maintenance of line breaks in line based approaches. However, formatting and layout are used by some metric based approaches for making the comparison. In actual comparison, comments are removed or ignored by many of the existing approaches. Instead, they apply normalized variable prior to making comparisons to trace out parametric Type II clones. During normalizations, all the identifiers are replaced by single identifier in the source code.

Statement Normalization

In order to identify parametric Type II clones, this approach uses identifier normalization before comparison is made. In these normalizations, usually all the identifiers are replaced by one common identifier in the source code. In the method that has been followed all identifiers are replaced with variable 'S' as it is shown in Figure 3.2. In order to normalize statements many other methods are used. An order sensitive indexing scheme is used by Baker in order to detect consistently renamed Type II clones.

In normalization there is another process. In that process the source code is converted to a standardized form, which removes differences in layout and spacing. This approach is generally used while clone detection process is text –based. This process is to detect clones which differ in lay out and spacing. Italso generates an exclusive text file for potential clones in the source code unit. In the literature of code cloning, this process is called as pretty printing of source code. Ifany code fragment changes the structure of the code, other transformations are applied to it. With this, minimal variations of the similar syntactic form may be considered as a clone. For instance, in variable declaration of the source code, removal of keywords like global, static etc. takes place.

Template Conversion

The process in which, the transformation of input source code into a set of predefined statements is known as template conversion. This is nothing but converting the original source code into a standard intermediary form. Examples are renaming of variables, data types, names of functions etc. The format which is used in textual analysis is named *template*. The selected candidates are compared textually during the detection of Type II cloned methods. During the clone detection process, the function identifiers, definition, names of variables, types etc. are edited. This is done because just the textual comparison is not sufficient.

After the conversion of the source code to a template is completed, the template file and the source file are stored in the database. This storage of files is used for application of metrics on it. This transformation varies from simple to complex. The simple transformation would include removal of the white space whereas the complex transformation would include extensive source code transformations. For every single comparison unit an attribute vector is computed for those intermediary representations when metric based methods are applied.

SOURCE CODE	***TEMPLATE***
inttemplconv(pmtra, buff1,leng, buff2)	*DAT FUN_NAME(S,S,S,S)*
char buff1[];	*DAT S;*
intleng;	*DAT S;*
intpmtrb;	*DAT S;*
char buff2[];	*DAT S;*
{	*{*
int i;	*DAT S;*
int j;	*DAT S;*
While(i<=leng)	*LOOP*
{	*{*
If(buff1[pmtra+j]!=buff2[pmtrb+j])	*IF*
return TRUE;	*RETURN;*
};	*};*
i++;	*ASSIGNMENT STATEMENT;*
j++;	*ASSIGNMENT STATEMENT;*
Tembuf(pmtra)='\0';	*ASSIGNMENT FROM FUNCTION CALL*
return TRUE;	*RETURN;*
}	*}*

Figure 3.2: An Example for Template Conversion

Metric Computation

Collection of different metrics from a specific code fragment, such as a class or function and then group these metrics together into a vector which is usually called metric vector. Then the clone detection process compares this metric vector instead of actual source code. In this method all types of clones including Type IV functional clones are detected.

In this metric computation every code fragment is given specific metric values. The comparison of metric values instead of original source code directly indicates that these metric vectors are used for detecting similar code with an allowable distance. Text based techniques are simple traditional way of detecting code clones. It takes a line of source code as a unit of code representation. To increase the performance of a clone detection technique, textual approach transformation of lines of code is required using hashing function. With this process uninterested code is removed before comparison.

Inorder to detect Type I, Type II, Type III and Type IV clone methods, a series of 12 current method level metrics are utilized which are as follows

i. LOC per method (i.e. Number of effective lines of code in a method)

Count the number of lines of code
Subtraction of blank lines
Subtraction of comment lines
Subtractions of lines which consist of only block constructs. (i.e. '{' & '}' are the begin and end block constructs in java programming)

ii. Number of local variables present in a method.

A variable which is visible only in the block of code in which it appears is considered as local. The scope is also local. A local variable is valid only within that function block in a method.

iii. Number of conditional statements in a method

The features of a programming language in computer science are conditional expressions, conditional statements and conditional constructs. These features perform computations which are diverse depending on a programmer specified Boolean condition evaluation. It evaluates a Boolean value *true* or *false*.

iv. Number of loops identified in a method.

If a statement is executed as many times as required, then it is considered to be a looping statement.

The looping statement is useful when some constraints which have a specific value need to be checked.

v. Number of passing parameters in a method.

The function called includes the specification of the function name which is followed by the function call operator and other values of data which the function expects to receive. These values become the parameters which are essential for the function. The whole process is known as passing arguments to the function.

vi. Number of function calls in a method.

An expression which not only contains a simple type name but also a parenthesized argument list is a function call. The argument list may contain many expressions which are separated by commas. This could also be empty in some cases.

vii. Number of times the function has been called from other methods.

If a function is declared and defined it can be called from anywhere and any point within the program, main function, from another function and even from itself. This involves specification of the name of the function followed by function call operator and other data values which are expected to be received by the function.

viii. Number of return statements in a function.

The processing of the current function is ended by a return statement. It returns control to the caller function from the called function. A return statement which contains an expression is included in a value returning function.

ix. Number of inheritance in each method.

Inheritance is a way to compartmentalize and reuse code by creating collections of attributes and behaviours called objects that can be based on previously created objects.

x. Number of virtual functions in a method

A virtual function or virtual method is a function or method whose behaviour can be overridden within an inheriting class by a function with the same signature.

xi. Number of overriding functions in a method.

Amethod can only be written in Subclass, not in same class. The argument list should be exactly the same as that of the overridden method. The return type should be the same or a subtype of the return type declared in the original overridden method in the super class.

xii. Number of overloading constructors in a method

Overload constructor is multiple constructors which differ in number and/or types of parameters.

However, for each diverse method that is identified, the metrics are computed and the corresponding values are stored in a database. Once the metric values are computed, same set of values are recognized by making a comparison of the records in the database. These metric values may be same for two different methods which are not similar, manual analysis has to be done to finalize duplicated functions. The proposed approach was able to identify most of the duplicate methods. Textual comparison is done for the short listed set of candidates to identify as clone pairs. Table 3.1 shows the metric values for the code shown in Figure 3.2.

Table 3.1: Metric values for the code shown in Figure. 3.2

Sl. No.	Metrics	Value
1.	No. of lines of code	18
2.	No. of arguments passed	4
3.	No. of local variables declared	6
4.	No. of function calls	1
5.	No. of conditional statements	1
6.	No. of looping statements	1
7.	No. of return statements	2

Textual Analysis

In textual analysis line by line comparison is done which means total lines are textually compared to each other using hash function for strings. The hash function is used to identify the duplicated entries in a Table; the same has been used here to identify the duplicates in code. This is done by using an efficient string matching algorithm (suffix tree string search algorithm using hash function to map similar strings).

The transformation of the source code is not used in textual approach before the application of comparison. In the process of clone detection, the source code is taken directly for comparison. The text based approach which is considered to be an efficient technique can detect only Type I clones.

This approach cannot be taken for granted as the structural type of clones which have diverse codes but same logic cannot be detected. Inorder to trace Type II clones, textual analysis techniques can be modified and utilized. These modifications can include normalization of the statements. After textual analysis the clones that are type I are identified. In textual analysis line by line comparisons are made but to identify different clones the language constructs must be compared. These constructs are divided into tokens.

Token Parsing Technique

In this approach the source code has been transformed into tokens which are similar to tokens of lexical analyzer which is present in a compiler. In this approach parameter tokens like java variables and identifiers are divided into transformed source code for direct comparison, and the other tokens which are non-literals are identified by applying a hash function on them. These tokens are encoded with a position index of their presence in the source code line. Finally these indexed tokens are represented in suffix tree or abstract syntax tree for comparisons and clones are detected

Clone Detection Using Tree Based Analysis

Actual process of clone detection starts with a string matching technique. Token parsing and graph matching techniques are applied on the processed data. Though many string matching techniques are available, thisapproach concentrated on *suffix tree* construction method to find exact strings, substrings and parameterized strings.

Token parsing is the process of dividing the strings into tokens as per their language constructs. These tokens have been compared with other tokens of the original program. Any way the white spaces and comments are removed in the earlier phases of the process, only tokens of the language constructs are compared. Finally Graph matching technique is applied, which is done with ease using Abstract syntax tree method. Abstract syntax tree (AST) method is used in string matching technique for textual detection method of clone detection technique.

Suffix tree method alone cannot find all clones. In the proposed method, an Abstract Syntax Tree is used in combination with suffix tree to find all

the clones. The following section explains how a string matching technique works.

String Matching Technique

To make string based techniques independent of programming languages basic string transformation and comparison algorithms are used. The techniques which are used in this category are different in the string comparison algorithm. Comparison of calculated signatures per each line is one way of identifying substrings that are matching. The other way of comparison is line matching, in which there are two variants. This type of comparison is considered as a representative category because general string manipulations are used.

The discovery of code fragments which compute the same result is a major problem in clone detection. In order to go ahead with this, the parts which are willing to be compared must be first fragmented. Later it has to be determined whether the fragment pairs are equivalent. The determination of a single fragment is not possible, so determination of two arbitrary program fragments under the same circumstances is impossible. Hence, it is difficult or not possible in theory to determine that they compute identical results. Deep semantic analysis traditionally bounded by time limits can be utilized for equivalence detection. This can be done because false negatives are acceptable. In order to go ahead with equivalence detection semantic definitions, theorem provers, etc., are essential. These are considered as considerable infrastructure for detection of equivalence.
If many false positives cannot be produced, simpler definitions of code equivalence may be sufficient. This denotes that clone detection can be done by more syntactic methods. The source lines can also be compared. If source lines are equal, it is assumed that the cloning process has not introduced any changes in identifiers, comments, spacing or other non – semantic changes. Hence this restricts clone detection to exact matches.

Due to this, detection of near miss clones is failed. Apractical possibility is to compare program representations with explicit control and data flows. In order to modify large software systems and to build transformational tools (DMS) semantic designs are used [9]. Before source programs are transformed, such tools are typically parsed into ASTs as a first step. Investigating and comparing syntax trees is chosen because of the early product state of the proposed method. This enables avoiding confusion and uninteresting changes at the lexical level.

The clone detection process consists of few steps. In the first step, the source code is parsed and an AST is produced for it. In order to find clones, three main algorithms are applied. The first algorithm is Basic algorithm. This is used to detect sub-tree clones. The second algorithm is sequence detection algorithm. This is used essentially to detect statement and declaration sequence clones. The third algorithm traces out for more complex near-miss clones. This attempts to generalize combinations of other clones. The remaining clones which are detected can be pretty printed. Clone removal is not carried out. The above mentioned algorithms find three different string matching patterns.

i. Exact string matching
ii. Parameterized matching and
iii. Substring matching

Exact String Matching

The first variant of line matching is Exact string matching. In this, both detection phases are straight forward. Only minor transformations using string manipulation operations are applied. They operate using very limited knowledge about possible language constructs. The removal of empty lines and white spaces is known as transformation. When comparison is done, all lives are compared with one another with the help of a string matching algorithm. This results in large search space. The space is generally reduced using hashing buckets. Before all the lines are compared, they are hashed into one of the possible buckets. Later, all the pairs in the same bucket are compared.

Parameterized Line Matching

Another variant of line matching is parameterized line matching. This detects code fragments which are both identical and similar. They can be considered as changeable parameters because identifier names and literals undergo a change when a code fragment is cloned. Hence, identical fragments are allowed. These fragments differ only in the naming of these parameters. The set of transformations is extended with an additional transformation to enable parameterizations. This replaces all identifiers and literals with one common identifier symbol like *$P*. The comparison becomes independent of the parameters because of this additional substitution. Hence, no additional changes are required to the comparison of algorithm of itself.

Substring Matching

After source text normalization substrings are generated. The process of substring generation is

controlled by the user. In a substring, the length of the characters has to be mentioned where as in a large system the number of lines has to be mentioned as a parameter for generating substrings. With these substrings a suffix tree is constructed to find the potential clones. Sometimes these identified substrings are overlapped due to the length that has been mentioned.

Analysis of Clone Pairs and Clone Clusters

Identifying the potential clones and clone pairs is done by making a line by line comparison for the normalized and standardized source code clone detection method for Type I. In this process, the identical code fragments are chosen leaving aside differences in comments, layout and white spaces. The comparison of templates is done for Type II clones. In the process, only the syntactically identical fragments are taken, leaving aside the identifiers, types, literals, layout, white space, and comments. For the fragments if there are some modifications and some similarities, they must be considered as Type III clones by comparing template with exact code. The code fragments which are copied with some changes like modified, removed or added statements along with differences in types, identifiers, white space, literals, comments and layout are considered as Type III clones.

If the code fragments are totally different but the output is similar for different inputs, then such fragments are considered as Type IV clones. When the functionalities of the two code fragments are similar or identical, then they are regarded as Type IV clones.

When similar computation is performed by two or more code fragments and the implementation is done by various syntactic variants, then they are said to be Type IV clones. Clustering is done separately for each identified clone method. These clusters are numbered uniquely. A clear image as to how the methods have been cloned can be understood by clustering. This enables a review process that is easier.

Clone Refactoring

The clarifications of systems which are affected because of duplicated code are supported by very few methods. A code duplication detection method is proposed to guide the process of refactoring. The purpose is not to completely automate the process, but to help re-engineering process. This is to stress the fact that a first analysis of the situation can be done and provide a solution if possible.

Extract method and *Pull up method* which are considered as existing refactoring patterns are used to remove code clones. In *Extract method*, a fragment of source code is extracted and redefined as a new method. This type is applied to very lengthy method or a part which is too complex. *Extract method* is used as a common new method in order to extract code clone fragments.

In *Pull–Up Method*, the methods which are defined in child classes are pulled up to its parent class. Because of design pattern, this pattern is performed. When two or more child classes have a common parent class and when they have a clone method, pull up method is used. This is used to remove clones.

In order to trace out the refactoring pattern applicable to each code, the characteristics are measured. For instance,*Extract Method* means extraction of a code fragment. The variables defined outside it are not referred and assigned in it. When such variables are used, it is compulsory to provide them as parameters for the new method. Hence, the amount of such variables is measured.

The removal of identical methods in child classes to the parent class is known as *Pull–Up Method*. It is obligatory that the child classes have a common parent class. Hence, the position and distance of clones in the class hierarchy is measured. This characterization enables to determine how each clone can be removed. The decision of how a code clone must be addressed for refactoring depends totally on the metric analysis.

These methods of refactoring approaches are clearly discussed in detail in the literature of code cloning.

Conclusion

The proposed methodology uses a systematic approach like any other model to detect clones. It allows preprocessing the statements to remove white spaces, comments and normalization to reduce number of comparisons. It is effective to detect all types of clones by textual and metric analysis. It also uses template conversion to reduce the syntax tree comparisons so that it is recognized as light weight method. It shows a solution in the form of refactoring for the problem of code cloning.

REFERENCES

[1] IEEE. Standard for Software Maintenance.IEEE Standard 1219, 1998.
[2] ISO/IEC. Software Engineering - Software Maintenance. ISO/IEC 14764, 1999.
[3] L. Arthur. Software Evolution: The Software Maintenance Challenge. Wiley, 1988.
[4] S. W. L. Yip and T. Lam.A software maintenance survey. In Proc. of the 1st Asia-Pacific Software Engineering Conference, pages 70–79, Dec 1994.

[5] S. Chidamber and C. Kemerer.A metric suite for object-oriented design. IEEE Transactions on Software Engineering, 25(5):476–493, Jun 1994.

[6] ClearCase. http://www-306.ibm.com/software/awdtools/clearcase/.

[7] Robert Tairas, "Clone detection and refactoring", Proceeding of OOPSLA '06 Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 780-781, New York, USA, 2006

[8] Chanchal K. Roy, James R. Cordya and Rainer Koschkeb, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Journal Science of Computer Programming, Vol. 74, No.7, pp. 470-495, May 2009.

[9] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna and Lorraine Bier, "Clone Detection Using Abstract Syntax Trees", Proceedings of the International Conference on Software Maintenance, pp. 368, Washington DC, USA 1998

[10] G.Anil Kumar, Dr.CRK.Reddy, Dr.A.Govardhan "Code duplication in Software Systems: A survey", International Journal of Software Engineering Research & Practices Vol.2, Issue 1, Jan 2012

[11] M. Fowlor. Refactoring: improving the design of existing code. Addison Wesley, 1999.

[12] R. H. Page. http://www.refactoring.com/.

[13] MagielBruntink, Arie van Deursen,Remco van Engelen, and Tom Tourwe, "On the Use of Clone Detection for Identifying Crosscutting Concern Code", Ieee Transactions On Software Engineering, Vol. 31, No. 10,pp. 804-818, October 2005

[14] Abouelhoda M.I., Kurtz S.andOhlebusch E, "The enhanced suffix array and its applications to genome analysis", In Proc. Workshop on Algorithms in Bioinformatics, vol. 2452,pp. 449–463, Berlin, 2002

[15] Hamid Abdul Basit and Stan Jarzabek, "Detecting Higher-level Similarity Patterns in Programs", European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 1-10 Lisbon, Sept. 2005

[16] Lingxiao Jiang, Zhendong Su and Edwin Chiu, "Context-based detection of clone-related bugs", Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 55 – 64, New York, USA, 2007.

[17] Chanchal Kumar Roy and James R Cordy, "A Survey on Software Clone Detection Research", Computer and Information Science, Vol. 115, No. 541, pp. 115, 2007