Accuracy Enhancement in Code Clone Detection Using Advance Normalization

[1]Ritesh V. Patil, [2]S. D. Joshi, [3]Digvijay A. Ajagekar,[4]Priyanka A. Shirke, [5]Vivek P. Talekar, [6]Shubham D. Bankar

[1]Research Scholar,Computer Engg. , [2]Professor,Computer Engg., [3,4,5,6]Computer Engineering Department.
[1]Bharath University, Chennai, [2]BVDUCOE, Pune , [3,4,5,6]P. D. E. A.'s College of Engineering, Pune.

***Abstract*—Code clone is reuse of code with or without some modification. Software cloning is a methodology which can use particular part of code which is previously available but with some modification in that code to perform same functionality. Generally clones are considered very hazardous to the quality of source code. Code clone creates serious maintenance issues. Main drawback of code replicas is that alterations to one code segment may need to be propagated to several other similar ones. Various techniques have been proposed to find duplicated code. Many existing system can detect the Type-I, Type-II and Type-III clones to some extent. Normalization and code reduction technique are helpful to find the clones with greater optimization. Code reduction technique is efficient because it saves the storage space. Here proposed system presents the Advance Normalization which is an additional feature for even better accuracy in finding clones. Due to presence of insignificant zero in the code, detection tool gets affected in terms of accuracy. Advance normalization provides solution for elimination of the cipher problem.**

***Index Terms*—Advanced Normalization, Code clone, Code reduction, Normalization, Reused Fragments.**

I. INTRODUCTION

Clone is keyword which is used for duplication. The method of searching code twins in the software system is called as clone detection technique. In software development, reuse of code by copy-paste activities in source code is a common approach. So software clones are the results of this reuse approach. Code clones are considered as a repeated part of code in software system and that cloned code has several adverse impacts on the maintenance life cycle of software system. Therefore, it is beneficial to eliminate redundant code fragments in the software system.

Detection of code clone is really important to increase quality of source code. The code clone detection tool must be scalable, good in its availability and it should provide good precision. Clones are differentiated on the basis of textually similar and functionally similar clones. Type-I, Type-II, Type-III clones are textually similar and Type-IV is functionally similar.

Type-I (Exact clones): Identical code pieces except for changes in white spaces layout and comments.

Type-II (Renamed clones): These clone types are similar in syntax except for literals, variables and identifiers.

Type-III(Modified clones): These clone types are copied fragments with some alterations such as changed, added or deleted statements.

Type-IV(Semantic clones): These are code fragments that are similar functionally synonymous but their implementation is by different syntactic variants.

Table 1- Type-I (Exactly Similar)

File-1	File-2

Java code for Division:
 	
class Division
{
public static void main(String args[])
{
int x=20,y=10;
System.out.println("Division is "+(x/y));
}
}
	Java code for Division:
	
	Class Division
	{
	public static void main(String args[])
	{
	int x=20,y=10;
	System.out.println("Division is "+(x/y));
	}
	}

Table 2-Type-II (Variable Renaming)

File-1	File-2

Java code for Division:
class Division
{
public static void main(String args[])
{
int x=20,y=10;
System.out.println("Division is "+(x/y));
}
}
	Java code for Division:
	class Division
	{
	public static void main(String args[])
	{
	int m=20;
	int n=10;
	System.out.println("Division is "+(m/n));
	}
	}

Table 3-Type-III (Gapped Clones-Insertion/Deletion of statements.)

File-1	File-2

Java code for Division:
class Division
{
public static void main(String args[])
{
int x=20,y=10;
System.out.println("Division is "+(x/y));
}
}
	Java code for Division:
	class Division
	{
	public static void main(String args[])
	{
	System.out.println("Enter the two nos");
	int p=20;
	int q=10;
	int v=p/q;
	System.out.println(v);
	}
	}

Table 4- Type-IV (Semantic Clones)

File-1	File-2

Java code for Division:
class Division
{
public static void main(String a[])
{
int x=20,y=10;
System.out.println("Division is "+(x/y));
}
}
	Java code for Division:
	class Division
	{
	public static void main(String a[])
	{
	int x=Integer.parseInt(a[0]);
	int y=Integer.parseInt(a[1]); int z=x/y;
	System.out.println("Division is=")
	System.out.println(z);
	}
	}

II. Related Search

There are various methods for clone detection technique:

1) Text based Approach:-

Sequences of lines or strings of target source program are used. Matching of two code fragments is done with one other to find sequence of same text/strings [1].

2) Token-based Approach:-

Tokens or lexemes of source code are generated for finding the clones. These tokens are formed by using lexer or parser. Compared to text-based approach, lexeme oriented approach is highly robust [2].

3) Tree based Approach:-

Tree based approach involves constructing a parse tree of source program [3]. Comparison of nodes with other nodes is prepared to detect the replicated code.

4) Program Dependency Graph(PDG):-

PDG is abstraction oriented approach. It involves considering the semantics of source code [4,5] and having a source

code which is highly abstract.

5) Metric based Approach:-

In the metric based approach, different metric vectors are gathered present in the source code information. Instead of comparing code directly, these metrics are matched [1]. If the metric value is equal, then those codes are said to be clones.

6) Hashing technique:-

Tool uses optimization technique by using hash function for string which minimizes the computation complexity.

Bongki Moon proposed the "Parallel Processing of Data with Map Reduce" technique. To hide details of parallel execution and allow users to concentrate only on data processing strategies is main idea of MapReduce model [6,7]. The MapReduce model consists of two primitive functions: Mapping and Reduction. To read the input and for storage of output, there is utilization of Goole File System (GFS) as underlying storage layer in MapReduce [8]. For each map, mapper can be assigned. Mapper can carry out the normalization task. Mapper produces the intermediate results. The intermediate outputs are read by and merges them by the intermediate keys, so that group of keys having same values is formed. Some advantages of this technique are that it is simple to use, flexible, a single fixed dataflow, independent of the storage.

Code cloning is the replicating process of code parts by using copy paste activities with minor adaption in software development. One of the approach is used for elimination of code twin is refactoring [9,10]. That refactoring method can provide only exact method and pull Up method [10]. If the information is passed about source code to refactoring, then it only removes the functional type's specific clone [11]. This approach has two stages for removal of code clone. In the stage one, there is detailed analysis of detected code clones. In the second stage, they focus on more interactive refactoring process for code twin elimination process.

Minhaz Fahim Zibran et.al presented the research of cloned code analysis, management and refactoring. Author says that copying a fragment might cause error propagation. So, existing code clone can increase the maintenance effort. For big software system, the clone finding is complex [12] in process of maintenance of software. In first study [13], there is analysis of overall status of code cloning in software process such as programming language/paradigm, system size and evolution of code clone. In the second study [14], a fine grained analysis, which investigated how the individual clone fragments changed and evolved across subsequent releases of evolving software systems?

Another detector tool is named as VisCad. This is comprehensive code twin analysis tool for visualization that support NiCad which is near-miss hybrid clone detection tool [15].The VisCad tool effectively used for high scale clone visualization. VisCad works on top of NiCad and utilizes the results reported by NiCad. VisCad not only visualizes clone detection results from NiCad but also detect clones from highly abstract source code level. Radial map, scatter plot and tree map are supported by VisCad. VisCad only find exact and renamed clones of large system.VisCad can be implemented in java and run in Java Runtime Environment (JRE).

III. OBJECTIVES AND CHALLENGES

The clone detection tools so far studied are mainly impotent to detect all the possible types of clones. Mainly the detection of functionally synonymous clones is a challenge and so is our objective. The achievement of accuracy in finding clones as well as to construct a system for efficient clone detection which makes swift search of these replicated fragments. The efficient handling of the available memory is also challenge to be focused because disc management is important factor for enhancing the speed of the tool functioning.

IV. METHODOLOGY

The proposed system presents a methodology in which the system proposed makes it easy to search the code twins with greater scalability. The detection of Type-I and Type-II clones here is achieved by simply using Map-Reduction algorithm and indexing in the inverted manner. Such processes make the exact clones easy to find. Further to detect other clone types i.e. Type-III and Type-IV clones, we have proposed a system which can associatively work with inverted indexing. The important feature in the proposed clone finding tool is Optimization. This optimization is gained by estimation of the Block of Control and Executive statements.

The Block of Control and Executable statement is the part of code which contains the main program logic. A program code has five sections:

A. Declarative Section
B. Assignment Section
C. Executable Section (Process Block)
D. Output Section
E. Exception Handling Section

According to the observation, the declaration part gives clue of included package, or header that defines a function, class or interface which is utilized in Control Process Block.

The variables are not feasible for declaring them as clones because they may have many combinations. The Program flow is Input then Process and then Output. If two programs accept a set of input say **X**, produces same outputs say **Y** then both programs are definitely treated as clones. All this logical similarity is Process block oriented and not based on input. Identifying the important and correct part of code, which defines certain process, is called as process block. The variable declaration is not considered.

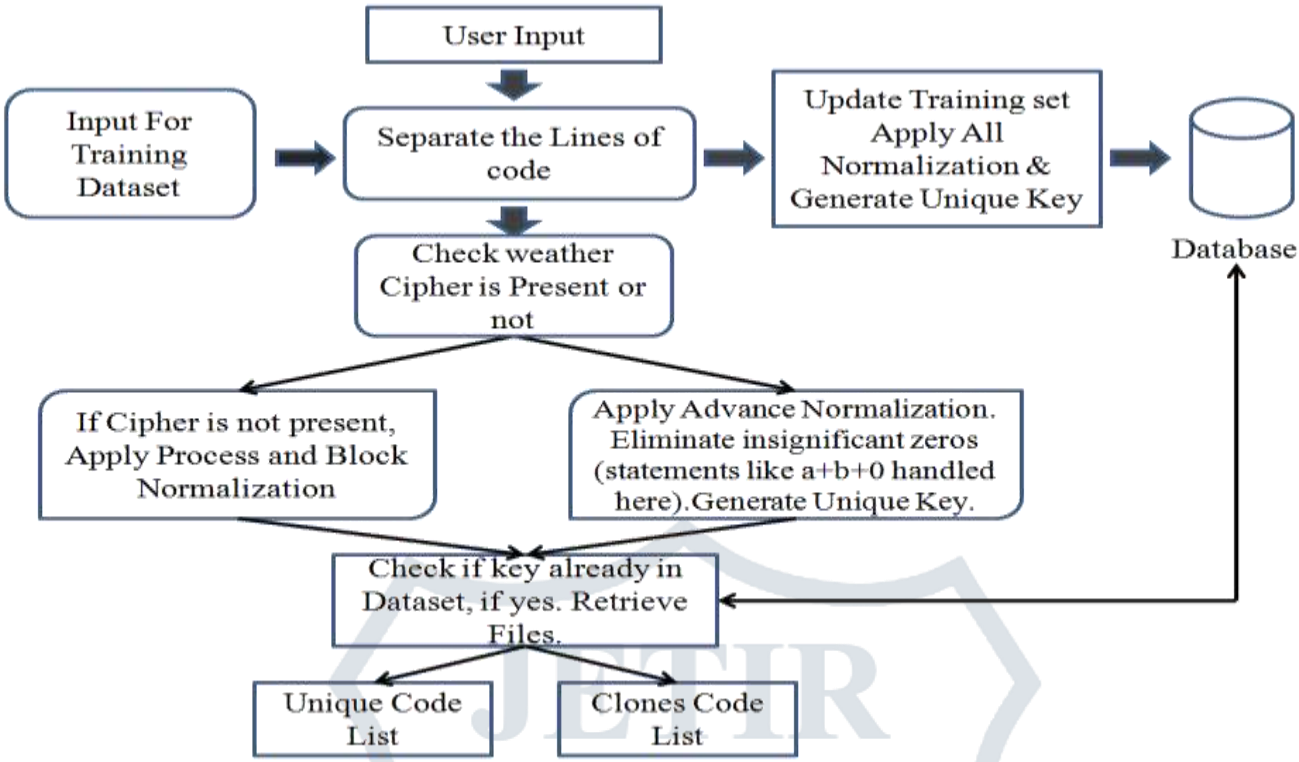

Fig.1 Architecture of Proposed System

Normalization is applied for greater accuracy. Process block is analyzed by assessment of code complexity, sum of variables used in process, likeness check for control statements, with the support of control flow graph, and token based method as explained Equations.

The Server:

The master activity of a server is to form a training set. The input code given to the server is initially tokenized. Using line separators, each line is differentiated into tokens and we may also call them as line vectors. These tokens then undergo for reduction process. The process normalization and variable normalization are performed on the line vectors. After reduction of code by extracting the features of the token, a unique key is generated for each line vector and is stored in the repository. The storage is in a way that indexing is made in inverted manner.

Client Section:

The client is the part which will accept the code which is to be examined for checking of clone presence in it. The code given input to the client also has to undergo several processes so that to generate the appropriate result of clone detection. The processes are as follows.

1.Input-

The code given as input to the client is the code which is questioned for twin code or clone.

2.Line Vector Generation-

Here, the input code is separated into different tokens of lines. Then the use of line separators is applied. The separated code into different tokens makes them divided and easy for further analysis.

The important also is to identify the block of executable statements. All the meaningful part of code lies in this block. Further analysis and estimation of this block make it happen to detect the clones.

3. Reduction-

The input code has to undergo from different reduction processes. The code is reduced into its small form or reduced form. The unnecessary part of the code like white spaces and comments are removed, this removal is known as Normalization. Different normalization like variable normalization and process normalization are carried out on the code and those line vectors are transformed into a reduced form.

4. Advance Normalization-

Normalization may reduce code but still if code has a constant or literal like zero in an arithmetic statement then it may create uncertainty when the unique keys are to be made from them. For Example, Consider two statements from different files:

[1] a = b + c ……(from File 1)

[2] p = q + r ……(from File 2)

Both of these statements would be reduced into S1 = S2 + S3, which when checked for clones would match and provide correct result. But, if there is any occurrence of zero in any of one statement, then the scenario becomes different given as below:

- a = b + c + 0 ……(from File 1)
- p = q + r ……..(from File 2)

Now, the statements given above are actually clones but the key which will be created will be S1 = S2 + S3 + 0 for File 1 and S1 = S2 + S3 for File 2. Despite of being clones, the result will be negative and will not be declared as clones.

For this reason, any operations like addition, subtraction or else having unnecessary zero in it, should undergo advance normalization.

Here, such statements will be identified and then removal of zero and other required steps would be taken in the consideration. This specialized process will be called as Cipher Elimination. This process will execute under the Advanced Normalization Block.

The input code after passing through the line separator goes to the Cipher Checker. This block checks for the following possible probabilities in which an arithmetic statement may have a zero without any significance.

Identification of six possible cases:

[1] + 0	[2] - 0	[3] + 0 +
[4] + 0 -	[5] - 0 +	[6] - 0 -

Advance Normalization is needed to be applied whenever there is occurrence of any of the above given cases. This block makes the statement to undergo Cipher Elimination Process. The process removes the insignificant zero from the statement and passes it to the simple Normalization which involves Process and Block Normalization. If cipher is not present in line vectors, then it proceeds through simple normalization only.

5. Key Generation-

The reduced code is acquired after normalization and reduction processes. For the purpose of unique key generation, the features extraction is performed and keys are generated based on this.

After the key generation, these unique keys are matched with the keys present in the training dataset stored in database. The matching of the keys shows the existence of clones. If they don't match, then there isn't clone in the program. After this matching of keys, the list of the cloned part or detected clones is prepared and also the original or unique part list is prepared.

In this way, the finding of the duplicated code is performed using reduction inverted indexing with advance techniques. The tool accuracy dependant on these techniques strengthens it because the number of disc input and output cycles are reduced to an enormous extent. The disc space occupied is minimum. In case of huge software codes, it becomes feasible to use this kind of tool. The tool is more scalable to find the code twin from whole software program. Moreover, the precision achieved is also at an exemplary state.

V. RESULT

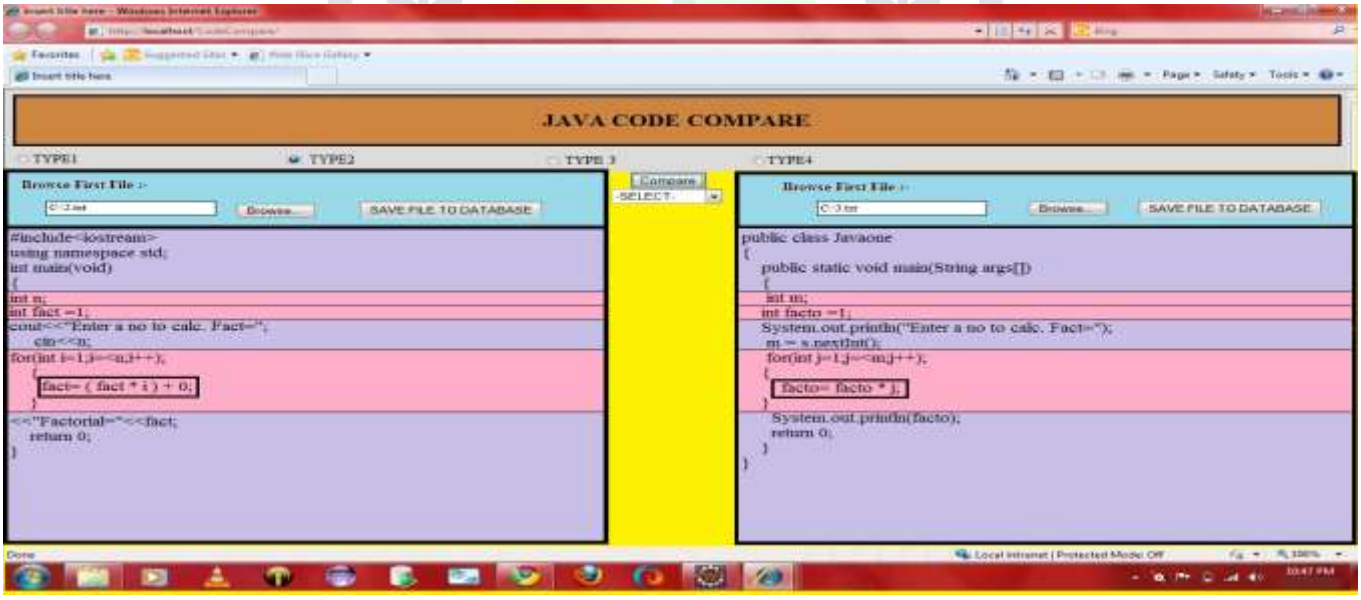

Fig. 2 The Result of code clone detection.

The Figure 2 shows, output of the implemented system. Highlighted part is considered for cipher elimination. Due to the advance normalization, there is 4-5% enhancement in the precision of code clone detection.

VI. CONCLUSION

The proposed system detects maximum clone types with greater efficiency and accuracy. The use of normalization and further advanced normalization brings precision to the system. The Type-I and Type-II simply are detected using the Inverted Indexing. The Type-III is detected using Variable Normalization for similarity analysis and the Type-IV which is semantic clone type is identified using the analysis of structural code complexity. The special feature of the proposed system of advanced normalization

and its underline process of cipher elimination make the clone detection further more precise and helps to deal with the issues raised due to presence of insignificant zero.

VII. ACKNOWLEDGMENT

The authors are thankful to Dr.V.Khanaa and Dr. Suhas Patil for their help in providing useful comments and recommendations on the previous survey of clone detection. The authors also thankful to Dr.Akhil Khare for a fruitful discussion on clone detection.

REFERENCES

[1] G.Kodhai.E, Perumal.A, and Kanmani.S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones", *Proceedings of the International Joint JournalConference on Engineering and Technology*, pp. 99-103, 2010

[2] 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, 2008

[3] L.Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalableand Accurate Tree-based Detection of Code Clones," Proc. 29thInternational Conference on Software Engineering, May 2007, pp.96-105..

[4] RaghavanKomondoor and Susan Horwitz. Using Slicing to Identify Duplication in source Code. In Proceedings of the 8th International Symposium on Static Analysis (SAS'01), Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.

[5] Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309, Stuttgart, Germany, October 2001.

[6] Abouzeid et al .HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proceedings of the VLDB Endowment, 2(1):922–933, 2009.

[7] Pavlo et al . A comparison of approaches to large-scale data analysis. In Proceedings of the ACM SIGMOD, pages 165–178, 2009.

[8] S. Ghemawat et al . The google file system. ACM SIGOPS Operating Systems Review, 37(5):29–43, 2003.

[9] Sandro Schulze, Martin Kuhlemann,"Advance analysis for code clone removal", *University of Magdeburg, Germany*,2009.

[10] M. Fowler, Refactoring - Improving the Design of Existing Code. Addison Wesley, 2000.

[11] R. Koschke, \Survey of Research on Software Clones," in Proceedings of Dagstuhl Seminar06301: Duplication, Redundancy and Similarity in Software, 2006, p. 24_

[12] Baker, "On Finding Duplication and Near-Duplication in Large Software Systems", *in Proceedings of the 2nd Working Conference on Reverse Engineering*, WCRE 1995, (1995).

[13] M. F. Zibran, R. K. Saha, M. Asaduzzaman, and C. K. Roy. Analyzing and Forecasting Near-miss Clones in Evolving Software: An Empirical Study. In ICECCS, pp. 295 – 304, 2011.

[14] R. K. Saha, M. Asaduzzaman, M. F. Zibran, C. K. Roy and K. A. Schneider. Evaluating Code Clone Genealogies at Release level: An Empirical Study. In SCAM, pp. 87 – 96, 2010.

[15] C. K. Roy and J. R. Cordy. NiCad: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In ICPC, pp. 172 –181, 2008.

[16] C. K. Roy and J. R. Cordy. Near-miss Function Clones in Open Source Software: An Empirical Study. Journal of Software Maintenance and Evolution 2(3): 165 – 189, 2010.Eason, B. Noble, and I.N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955.