Detection of Behavioural Clone

Manu Singh
School of ICT
Gautam Buddha University, Yamuna Expressway,
Greater Noida, India

Vidushi Sharma, Ph.D
School of ICT
Gautam Buddha University, Yamuna Expressway,
Greater Noida, India

ABSTRACT
High level cloning in a system is the aggregation of four classes of high level similarities these four classes are behavioural Clones, concept clones, structural clones and domain model clones. Behavioural clones are used to depict similar run time behavior. This paper presents a method for the detection of behavioural clone. The proposed system detects behavioural type of higher level clones in two file of same or different directories. This is done with the help of clone detection tool designed in DOT NET. The work is implemented as a generalized tool which accepts different programming language as input and the existence of clone can be detected across the source code of different languages. The main distinctive feature of the proposed methodology is the use of command prompt to determine runtime behaviour of cloned code.

Keywords
Behavioural Clones, higher level clone, simple clone, classification of high level clones.

1. INTRODUCTION
To keep up with the pace of changing technology and functional requirement, the system need to be timely upgraded and maintained. As the software is modified it becomes more and more complicated and difficult to maintain due to duplication in code. This duplication is sometimes called cloning in software and occurs at different levels of abstraction and may have different origin [1]. Literature review reveals that 5 to 50% of the source code is cloned [2]. Several techniques have been proposed to detect similar clone fragments also called simple clones [3] but analysis of similarities at higher levels of abstraction still remains a nascent area. High level clones may be used to extract important information about a software system design and implementation phase which may be further used to understand program, evolution of program, reuse and reengineering opportunities [4]. Despite of software clone research, clone management remains far from industrial adoption, and this area has gained more attention now [5].

The paper is aimed at exploring High Level Clones in terms of behavioural clone.

2. RELATED WORK
High level clones (HLC) are a growing area that uses a hierarchical organization of different levels of clones. Many research groups categories clone with reference to many contexts. A large amount work has been done on the detection of simple clones, but high level similarities are not yet explored.

In the literature several clone detection approaches have been projected for simple clones [6]. A basic classification of simple clone is identified see Fig.1. a) Approaches based on Text or String: In which source code is considered as sequence of string or lines. Two code fragments are compared with each other to search longest common subsequences of same string or text. b) Approaches based on token sequence: In which the whole source code is changed into a tokens sequence and scanned to locate cloned subsequences. c) Approaches based on AST: In which the whole source code is parse into an abstract syntax tree (AST) and detect similar subtrees. d) Approaches based on PDG: In which the whole source code is transformed into a Program Dependency Graph (PDG), on which an isomorphic subgraph matching algorithm is applied for searching similar subgraphs. e) Approaches based on metrics: metrics are used for code fragments; instead of comparing code directly metric vectors are compared.

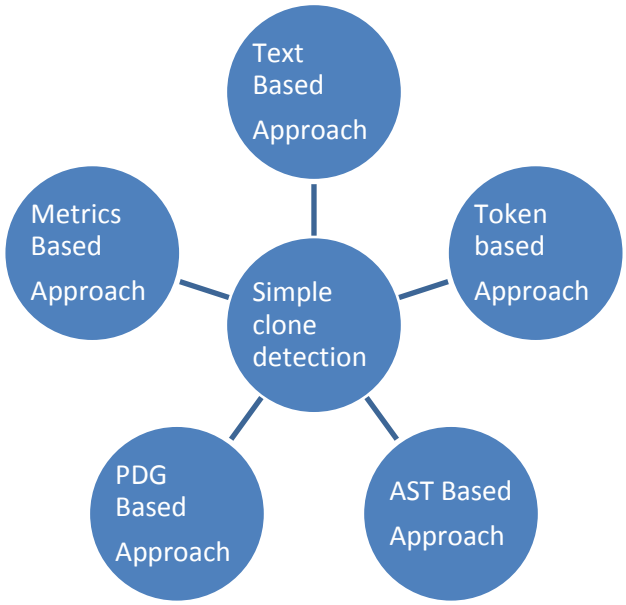

Fig. 1: High Level Clone Classification

The main distinctive feature of proposed technique is to use of command prompt to determine runtime behaviour of cloned code. This paper presents a method for detection of behavioural clone. The detection process operates on the premise that if two code fragments are semantic clones, then their input-output behaviour would be identical and if input output bahaviour is identical then their output is also identical. This is done with the help of clone detection tool designed in .net called High Level Clone (HLC) detector.

3. CLASSIFICATION OF HIGH LEVEL CLONING
High level clone (HLC) is an emerging concept that uses a hierarchical organization of fine gained clone fragments (Simple clones) to form coarser-grained clones (High Level Clone). Different research groups categorize clones with respect to different contexts. High Level Clones are classified [7] as in Fig.2. Classification can serve various purposes like studying the more frequently occurring high level clones, prioritizing different

types of high level clones, devising re-engineering strategies for different types of high level clones etc. High level cloning in a system is the aggregation of four classes of high level similarities. These four classes are behavioural Clones, concept clones, structural clones and domain model clones. In this classification one can abstract the behavioural clone and concept clones in runtime clones because both can be detect at runtime. Behavioural clones which are used to depict similar run time behavior and how a program should work also fall under the category of concept clones.

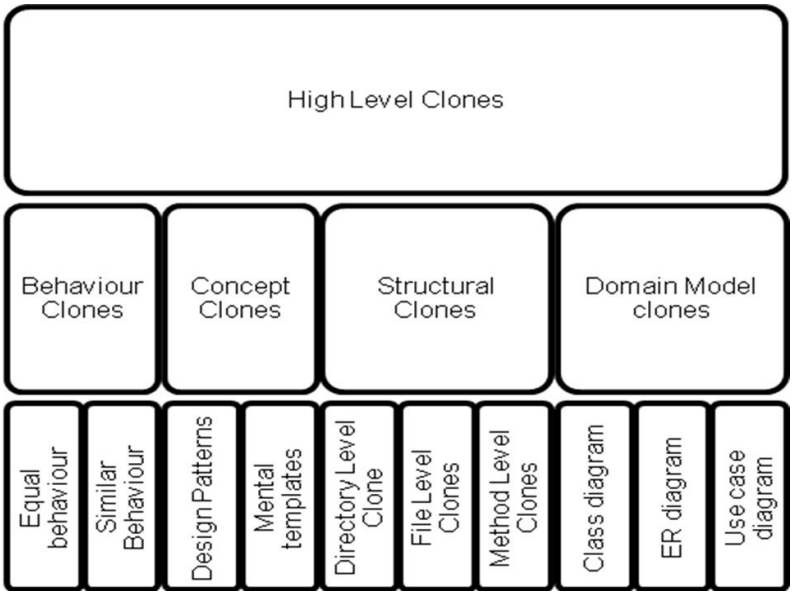

Fig. 2: High Level Clone Classification [7]

3.1. Behavioural Clone

Two code segments are Behavioural clone if they behave same it means execution is same for those two code fragments but their implementation is different. Juergens et al. [8] call two pieces of code behaviourally equal, if they have the same sets of input and output variables and are equal with respect to their function interpretation. So, for each input valuation they have to produce exactly the same outputs. Kwon [9] describes behavioural clones as the clones which depict similar run time behaviour. These can be detected as: if any two programs having same Directed Acyclic Graph (DAG) modeling data dependency or same control flow dependency, then they are behaviour clones of each other. Simply we can say that if two codes give similar output by giving same input then they can call behaviour clone. This is not necessary that they are representationally same. They may or may not be representationally same. Such type of clone could be represented by our example of swapping two variable values, Fig.3 and Fig.4 depicts different representations of this example but they all give similar output. In Fig.3 program implementation use bitwise operators to swap two numbers but in Fig.4 third temporary variable is used for swapping. These implementations of programs are different but execution is same for these two code segments.

```
#include <stdio.h>

int main()
{
  int x, y;
  printf("Enter the value of x and y\n");
  scanf("%d%d", &x, &y);
  printf("Before Swapping\nx = %d\ny = %d\n",x,y);

  x = x ^ y;
  y = x ^ y;
  x = x ^ y;

 printf("After Swapping\nx = %d\ny = %d\n",x,y);
 return 0;
}
```

Fig. 3 Swapping numbers by Using Bitwise operator

```c
#include <stdio.h>
int main()
{
   int x, y, temp;
   printf("Enter the value of x and y\n");
   scanf("%d%d", &x, &y);
   printf("Before Swapping\nx = %d\ny = %d\n",x,y

   temp = x;
   x = y;
   y = temp;

   printf("After Swapping\nx = %d\ny = %d\n",x,y)
   return 0;
}
```

Fig. 4 Swapping numbers by Using Temporary Variable

One can see that both fragments have the same behavior with equal input. For example: With x=5 and y=8 as input, the output of both computations will be swapping of values of both variables, whether it is implemented by using third variable or by using bitwise operators. With no structural or textual similarities, the code fragment has to be a behavioural clone.

3.2. Proposed Algorithm for Behavioural Clone Detection

Step 1 Take two source codes [S1, S2] as an input in which clones are to be detected.

Step 2 Normalize source codes by removing the comments and white spaces.

Step 3 Filter source codes obtained after normalization by removing the elements of the code such as void main(), header files, getch(), etc.

Step 4 Compute Metrics for S1 and S2 such as total no. of input variables (N1 and N2 respectively), LOC (L1 and L2 respectively and Complexity (CC1 AND CC2) respectively

Step 5 If N1==N2 and L1==L2 and CC1==CC2 then

goto step 6

else

goto step 12

Step 6 If line similarity exist then

goto step 7

else

goto step 8

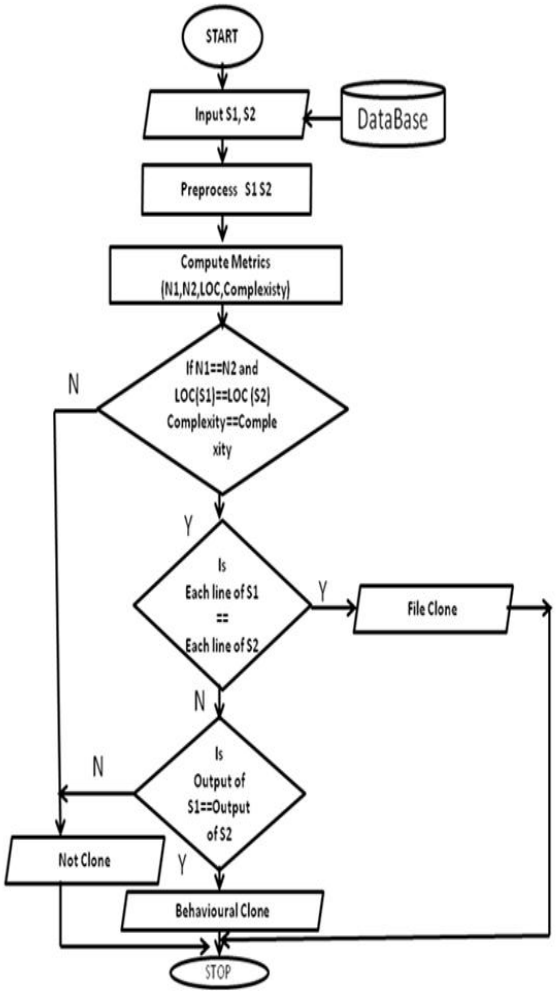

Fig. 5: Detection of Behavioural Clone

Step 7 Print "Codes are similar File clone exist" and exit

Step 8 Print "Codes are not similar may be behavioural clone exist" goto step 9

Step 9 if output of S1==output of S2 then

goto step 10

else

goto step11

Step 10 Print "Behavioural clone exist" and exit

Step 11 Print "Behavioural clone does not exist" and exit

Step 12 End

In the flowchart Fig.5 after taking two source files for clone detection comment lines, white spaces, keywords, header files and some commonly used library routines etc. are removed from both the source codes. Total number of input variables, line of code and complexity of each source code are computed. If matrices of both files are equal then comparison of each source code is done line by line till end of file. If all lines are similar then one can say it is file clone else if lines are not same then one can say implementations of S1 and S2 are different, hence execute the source codes to compare output. If the output is same then the clone is behavioural clone.

3.3. Result and Discussion

The paper presents a method for detection of behavioural clone. The detection process operates on the premise that if two code fragments are semantic clones, then their input-output behavior would be identical. The system has been tested with text files, C files, JAVA files and C# files as input and the results are produced based on the similarity between files. A sample result is shown below which states that the behavioural clone exists between comparison of two file and shown runtime similarity between two differently implemented codes. This tool allows the detection and analysis of the differences at runtime between two similar code fragments.

Table 1 : Metrcis Computation for Source codes

File id	Input variables	Line of source code	File Complexity	Potential File Clone Candidate	Potential Behavioural Clone Candidate
S1	2	20	1	No	No
S2	6	235	6		
S1	**2**	**20**	**1**	**No**	**Yes**
S3	**2**	**58**	**1**		
S2	6	235	6	No	No
S3	2	90	1		

According to the proposed metric similarity analysis shown in table 1. S1 and S3 source codes are not potential file clone, but value of input variables and complexity of both source codes are similar i.e. N1=2 ,N2=2 and CC1=1, CC2=1. According to the given behavioural clone definition two code segments are behavioural clone if they behave same it means execution should be same for those two code fragments but their implementation is different. The proposed technique verified the situation by execution of both codes. For verification three source codes (S1, S2, and S3) are taken in different combinations and are checked by the tool developed and evaluated on metrics. It is concluded that S1,S2 and S2,S3 are neither file clone nor behavioual clone whereas S1 and S3 are not file clone but one can say that S1 and S2 are behavioural clone. The execution shows that the end result of executing both S1 and S3 codes is similar. It means that S1 and S3 are behavioural clones. Fig.6. shows the results shown by the tool. The screen shots depicts that both source code S1 and S3 are behavioural clone as output of source code is same.

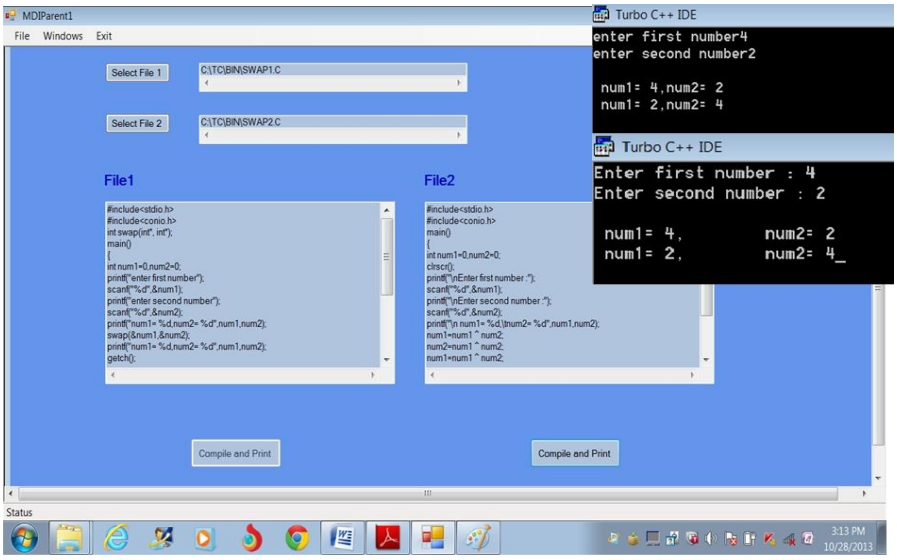

Fig 6: Behavioural Clone Detection System

Looking at the above example in terms of input output behavior similarity. The codes that give similar output on being fed with similar inputs are called behavior clones. They may or may not be representationally similar, but often their independent development creates redundancy [10]. It is assumed that behavioural clones are different from Type 4 simple clones, as type 4 clones are derived from other code but these clones are developed independently. This kind of similarity in behaviour is not detectable by existing clone detection tools. The tool is able to detect this kind of cloning by extracting runtime behavoiur of code. For this purpose required command prompt where code is executed. So the code is compiled and executed. After that the decision is made whether the code is similar code or not.

4. CONCLUSION

Code clone are similar program fragments that comes in various forms in software systems. Maintenance phase play an important role in software life cycle process. Software maintenance cost contributes to the total development cost. The proposed algorithm starts by finding simple clones (that is, similar code fragments). Gradually move to higher-level similarities by using data mining technique. Approach presented is both scalable and useful. Simple clones as well as high level clone information leads to better program understanding, helps in different maintenance related tasks, and points to potential reusable components. Higher level clones are also candidates for association with generic design solutions. After such association, programs are easier to understand, modify and reuse.

This implemented system combines both the text based and metric based techniques for behavioural clone detection. Metric based technique is a straight forward one, so it is a light weight technique. The text based technique is the one which gives high precision. In addition to behavioural clone this work can also be extended by detecting the other types of high level clones in our future paper.

5. REFERENCES

[1] H. A. Basit, S. Jarzabek, "A Case for Structural Clones", International Workshop on Software Clones (IWSC), 2009.

[2] B. S Baker, "On finding duplication and near duplication in large software system", proceedings of Second IEEE Working Conference on Reverse Engineering, 1995.

[3] William S. Evans , Christopher W. Fraser and Fei Ma, "Clone detection via structural abstraction" Software quality journal Volume 17, Number 4, 2009.

[4] Cory Kasper, Michael W. Godfrey, "Cloning considered harmful", Working Conference on Reverse Engineering, 2006

[5] M. Zibran, C. Roy, "The Road to Software Clone Management: A Survey", Tech. Report 2012-03, Department of Computer Science, University of Saskatchewan, Canada, pp. 1-62, 2012.

[6] C. Roy, J. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Science of Computer Programming, vol. 74, pp. 470–495, 2009.

[7] M. Singh, V. Sharma, "High Level Clones Classification" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-6, August 2013.

[8] E. Jürgens, F. Deissenboeck, and B. Hummel, " Code Similarities Beyond Copy & Paste", in proceedings of CSMR, pages 78-87, 2010.

[9] T. Kwon and Z. Su. , "Modeling high-level behavior patterns for precise similarity analysis of software", In UC Davis technical report CSE-2010-16, 2010.

[10] Nicolas Gold, Jens Krinke, Mark Harman,David Binkley, Proceedings of the 4th International Workshop on Software Clones IWSC '10, ACM, 2010