International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-6, August 2013

High Level Clones Classification

Manu Singh, Vidushi Sharma

ABSTRACT-In present time's High level clones (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. In this paper we review all such available categories of clones and present them in the form of a High Level Clone Classification. 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.. For this classification of HLC we develop a fuzzy rule-based system and also visualize the results.

Index Terms—High Level Clones, Fuzzy rule-based system, Fuzzy Inference System, Classification of High Level Clone.

I. INTRODUCTION

To keep up with the pace of change in 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 shows 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 [17].
In this paper, our aim is to uncover the less explored area of High Level Clones in terms of classification, fuzzification and identification.

II.HIGH LEVEL CLONES

Different research groups categorize clones with respect to different contexts. In this paper we review all such available categories of clones and present them in the form of a High Level Clone classification. Different classifications can be made based on the objective of the user who is dealing with them. Causes for High Level Clones include similarities in analysis (e.g., due to repeating analysis patterns [5]) and design (e.g., due to repeating design patterns [6]), requirements of the programming language (e.g., due to a repeating coding idiom), and mental templates repeatedly used by programmers.

Manuscript published on 30 August 2013.
* Correspondence Author (s)
Manu Singh, Research Scholar, School of ICT, Gautam Buddha University, Yamuna Expressway, Greater Noida, India.
Dr. Vidushi Sharma, School of ICT, Gautam Buddha University, Yamuna Expressway, Greater Noida, India.

Much work has been done on the detection of simple clones, but little has been done for detecting these higher level similarities. Clone classification can also help in further analysis of high level clone phenomenon like in prioritizing clones for reusing, refactoring, reengineering etc.
We have classified these clones on the basis of available literature. Due to vagueness in definition of clone, in the literature a categorization to the clone definition is attempted in the form of taxonomies. For instance, Mayrand et al. [7] provide an ordinal scale of eight different types of clones, of which some have simple, crisp definitions. For example, the category "DistinctName" refers to the clones where only identifiers names can be differed between the cloned segments. However, their ordinal scale is not sufficient towards a sound definition of clone. For instance, they define a category "SimilarExpression" to identify clones with expressions that differ but yet are still "similar". Similarly, Balazinska et al. [8] provide 18 different categories of clones based on what kind of syntax elements has been changed and also how much of the methods have been duplicated. While most of their categories are specific to a single change in the code, they still have categories "One long difference" to mean one unit token-sequence difference in an expression or in a statement or in other part of the function body, "Two long differences" to mean changes in two units and "Several long differences" to mean changes in three or more units, all of which involve kind of vagueness.
We could classify data in different higher level of abstraction like group of simple clones (structural clones), behaviour clones, Model clones (UML clones), concept clones and structural clones. 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 we can abstract the behavioural clone and concept clones in runtime clones because both can be detect at runtime. behavioural clones is used to depict similar run time behavior and how a program should work, also fall under the category of concept clones.

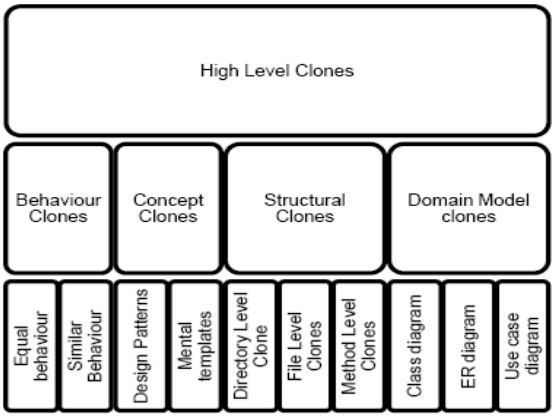

Figure 1 : Classification of High Level Clone

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

A. Behaviour Clones

Juergens et al. [9] 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 [10] describes behavioural clones to 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, Figure 2(a), 2(b) shows different representations of this example but they all give similar output of figure 2(c).

```
#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;
}
```

Figure 2 (a): Swap Nnumbers by Using Bitwise operator

```
#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;
}
```

Figure 2 (b): Swap Nnumbers by Using Temporary Variable

```
Enter the value of x and y 1 2
Before Swapping
x = 1
y = 2
After Swapping
x = 2
y = 1
```

Figure 2 (c) Output of Figure 2(a) and 2(b)

B. Concept clones

From the experience of Marcus [11], these clones manifest themselves as higher level abstractions in problem or solution domain like an ADT (Abstract Data Type) list. It is worth noting that the high level concept for which the clone is being detected depends on the user's understanding of the system. Their detection is improved with internal documentation and program semantics, where a semantic similarity is defined in terms of similar input and output conditions. Design solutions repeatedly applied by programmers to solve similar problems are called "mental templates". Design patterns and mental templates (or "skeletons") of how a program should work, also fall under the category of concept clones. Mental templates reflect common programming wisdom or programmer's specific experiences.

C. Structural Clones

These clones may be defined as similar program structures that are formed by lower level, smaller clones, with similar code fragments (i.e., conventional clones described in literature) at the bottom of such hierarchy. This concept of moving from lower level similarities to higher level similarities can be repeatedly applied, leading to the discovery of design concepts at various levels of abstraction. Basit[12] visualizes example of structural clones as in Fig 3.

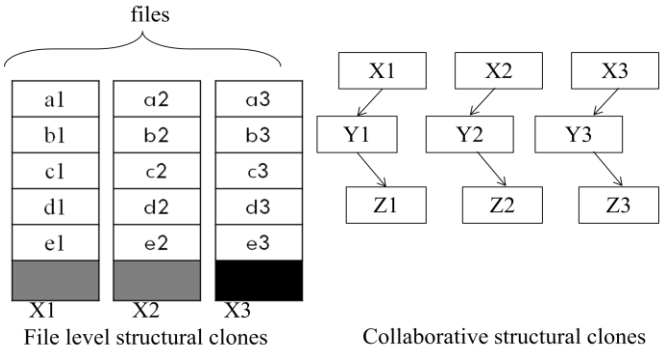

Figure 3 : Structural Clones

This considers a group of simple clone configurations recurring in files X1, X2 and X3 a file-level structural clone set. These file level structural clone set form collaborative (via message passing) structures of structural clones. Basit [13] describe how structural clone analysis extends the benefits of analysis based on simple clones only and discuss the concept that structural clones covers all kinds of large granularity repeated program structures.

D. Domain model clones

These are similarities in UML domain models like Class diagram, ER diagram, Use case diagram etc. It is likely that at some point, the overall model will contain duplicate fragments of sub models or model elements i.e. model clones, as per Harald. [14]. Clones in UML domain models can cause a problem during model based development, hence it is important to detect them. This is best possible through application of any of the reverse engineering tool, followed by domain expert's knowledge.

Case Study: Hospital Management System

Here a case study of Hospital Management System is taken to show UML diagrams to detect the clones at design level and then removing them. The Hospital Management System provides relevant information across the hospital to support effective decision making for patient care and hospital administration.

The HOSPITAL MANAGEMENT SYSTEM (HMS) covers many hospital activities. Every hospital big or small keeps the records of its patients including the registration details of the patient and the fee payments. Patients are categorized as In Patients and Out Patients.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

The entry of patients is determined whether he has arrived in emergency, OPD or for a routine check-up. The patient who gets admitted is provided with a room according to his/her choice. The patient is allotted a doctor according to his illness. The doctor may refer the patient to another doctor with expertise of the illness. On discharge, the patient is required to settle the bills sent by the accounts department of the hospital.

Out Patient module keeps the record of outdoor patients. It has various features like adding new patient details, searching, updating or deleting patient records, preparing medical bill, keep track of patient records and also the bill reports.

In Patient module keeps the record of admitted patients. It has various features like adding new patient details, searching, updating or deleting patient records, preparing medical bill, keep track of patient records and also the bill reports. It also keeps record of discharged patients.

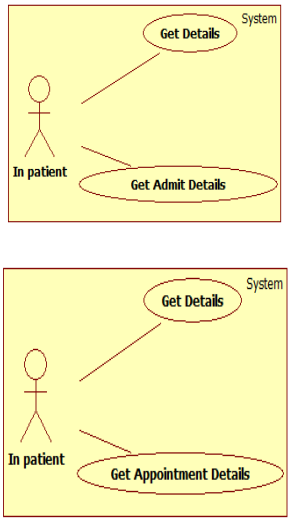

Figure 2. Use case Diagram of InPatient and outpatient category

This Figure shows the use case diagram of the category of patients- InPatients and OutPatients. It can be seen that inpatients and outpatients have same behavior and common attributes. Visualizing these we can see that they are actually clones to some extent. So we can easily detect these clones.

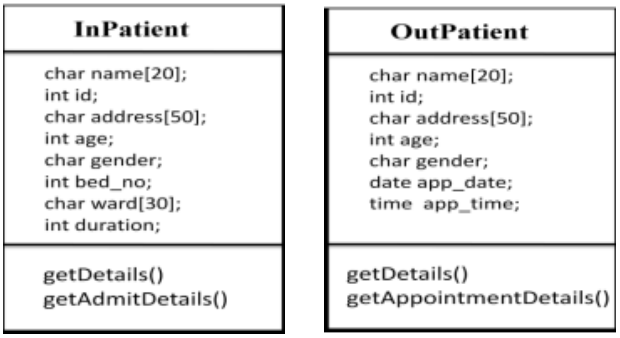

Figure 5. Class Diagram of InPatient and OutPatient Class

This Figure shows a representation of a part of the Hospital Management System highlighting the class structures of InPatient and OutPatient Class. In both the classes named InPatient and OutPatient It is seen that there are occurrences of common methods and variables. The method getDetails and various attributes like name, id, age, contact and others that are identical in both classes InPatient and OutPatient.

III.CONCLUSIONS AND FUTURE WORK

Software clones appear at various levels- simple clones are similar program fragments, in respect of similarity; while structural clones may be viewed as collaborations of lower level (simple) clones. A high level clone is an attempt to move towards high level similarity patterns, yet firmly rooted in patterns of concrete similarities at implementation level. A structural clone may indicate a cloned concept (in the requirements or design space). A 'high level concept clone' stems from a similarity in concepts. Detecting high level clones is directly related to domain analysis. To find the relationships between different repeating entities, semantic links between them need to be traced. In addition to code, other related artifacts like design diagrams and code comments can also be quite useful for locating and analyzing high level clones. We aim to extend the technique of semi automated detection of higher level similarities in software and present the results in a manner such that they can be utilized for providing insights into prospective reengineering (refactoring), reuse or anomaly free updating of the system. In future, we aim to expand the high level clones' taxonomy by including new types subtypes. We aim to demonstrate benefits of high level clone analysis through more case studies.

REFERENCES

[1] H. A. Basit and Stan 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 and Michael W. Godfrey, "Cloning considered harmful", Working Conference on Reverse Engineering '06, 2006
[5] Fowler, M., Analysis Patterns, Addison-Wesley, 1996.
[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design patterns: Elements of reusable object-oriented software, Addison-wesley, 1997.
[7] Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96), pages. 244-253, Monterey, CA, USA, November 1996.
[8] Balazinska, Merlo, Dagenais, Lague, Kontogiannis. Measuring Clone Based Reengineering Opportunities. In Proceedings of the 6th International Software Metrics Symposium (METRICS'99), pages 292-303, Boca Raton, Florida, USA, November 1999.
[9] E. Jürgens, F. Deissenboeck, and B. Hummel: Code Similarities Beyond Copy & Paste, in proceedings of CSMR, pages 78-87, 2010.
[10] T. Kwon and Z. Su. Modeling high-level behavior patterns for precise similarity analysis of software. In UC Davis techical report CSE-2010-16,2010.
[11] Marcus, A., and Maletic, J. I. : Identification of high-level concept clones in source code. in proceedings of the International Conference on Automated Software Engineering (ASE), pages 107-114, 2001.
[12] H. A. Basit, Damith C. Rajapakse and Stan Jarzabek : Structural Clones-Higher-level Similarity Patterns in Programs, SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, May, Lisbon, 2005.
[13] H. A. Basit, Usman Ali and Stan Jarzabek : Viewing Simple Clones from Structural Clones' perspective, in IWSC, Honolulu, 2011
[14] Harald S. Towards clone detection in UML domain models, pages 285-293,2010

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

[15] T. Kwon and Z. Su. Modeling high-level behavior patterns for precise similarity analysis of software. In UC Davis techical report CSE-2010-16, 2010.
[16] E. Jürgens, F. Deissenboeck, and B. Hummel: Code Similarities Beyond Copy & Paste, in proceedings of CSMR, pages 78-87, 2010
[17] M. Zibran and 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.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)