Volume 2, Issue 5, May 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Framework for Mining High Level Clones

Niyati Baliyan* *School of ICT* *Gautam Buddha University* niyati.baliyan@gmail.com

Vidushi Sharma *School of ICT* *Gautam Buddha University*

Abstract— **Software similarities or clones are known to hinder the software maintenance process. Clones manifest themselves in at fine as well as coarse granularity. The significance of extracting coarse grained similarity or high level clones lies in its benefits to reengineering, reuse, reverse engineering and program understanding. The problem of mining process for high level similarities thus becomes important. We propose a framework for mining high level clones in software. A case study has been applied to few phases of the mining process, it can be a starting point for understanding and analyzing high level clones for their semi automated detection and refactoring.**

Keywords— **Software similarity, Clones, Mining, CloneDR tool, Reuse.**

I. INTRODUCTION

The idea of software clones is linked with similarities in software which can be viewed as having fine or coarse granularity, may occur at different levels of abstraction and may have different origin [1]. Reports [2] indicate that 5 to 50 % of the source code is cloned. Though the reasons for cloning at in software are many, it is known that if clones in software systems are not properly identified and dealt with, they may prove problematic by making software maintenance process difficult [3]. There has been abundant research in code clones' analysis; however, analysis of clones at higher level of software abstraction may further benefit us. Detection of such *high level clones* leads to reusable solutions, reengineering, improved program understanding and evolution [4].

A major challenge faced while analysing clones of higher abstraction level is handling the bulk of clone information generated by the clone detection tools. In addition, since the idea of high level clones is both broad and complex, the detection process cannot be completely automated. Human intervention at some stage or the other renders it semi automated. Jiang and Hassan [5] define clone mining as the process of revealing clones of interest, from the results obtained by clone detection tools.

A framework for mining high level clones can help researchers cope with the clone information generated by tools, and also outline various phases requiring human intervention. Analogous to a traditional data mining framework which focuses on extracting unknown information from large data sets, a clone mining framework focuses on extracting unknown yet useful clone patterns from large clone sets[6].

In this paper, we outline the high level clone mining process and illustrate it with the help of a case study(Java web application for travel itinerary planner) applied to CloneDr tool[7]. The clone mining framework aids the analysis of high level similarities, which may prove useful for reusability and better maintenance. The paper augments the thus far research in semi automated analysis of high level clones and their mining.

II. OUR CLONE MINING FRAMEWORK

An overview of our clone mining framework is illustrated in Fig. 1.

The framework consists of four major phases:

1. Specification of clones of interest
2. Noise removal
3. Data condensation
4. Action

Prior to the first phase, some pre-processing is done, with the aim to identify potential clones and remove non essential differences obtained from parsing, as explained later. In addition, filtering of unnecessary detection results needs to be done, this is called post processing.

Heuristics' application is known as pre-processing. It implies applying heuristics or repository knowledge for better clone detection. Examples of such heuristics can be string or token matching or isomorphism of program structure graph. For instance, rejection from membership of clone class can be based on exceeding minimum token length. The granularity for clone can be set as-method, class, control structure or template etc. In case of complete duplication of code blocks i.e. structural clones [8], a code tokenizer or parser is more useful.

Clone detection results need to be reduced, abstracted and highlighted, after being produced from semi automated detection of high level clones. For this, the challenge is to be able to handle that clone data and thereby analyze and use

clone results, for improved software. Jiang et al. [5] first introduced the concept of clone mining as the process of revealing interesting clones and clone patterns from large volumes of clone candidates. These clone candidates are in turn the result of clone detection tools.

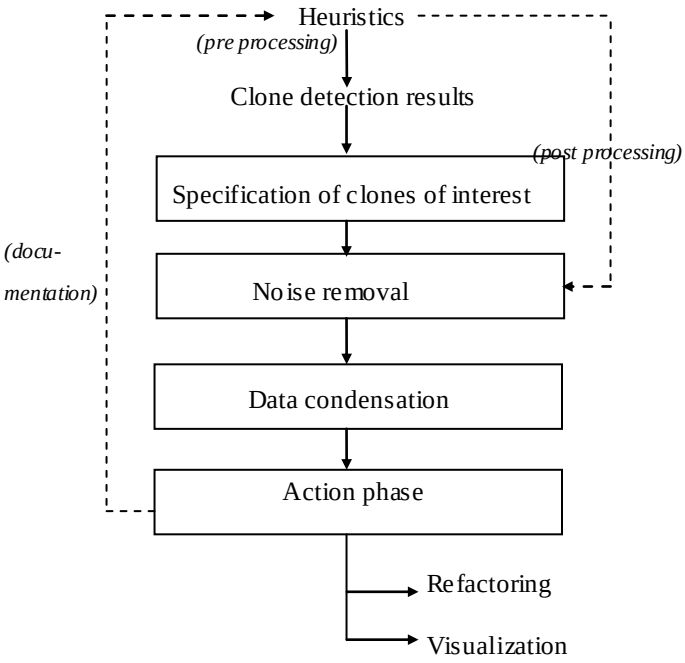

Fig. 1 Phases in High Level Clone Mining Process

Specification of clones of interest refers to stating the domain of our problem and defining the aim of our high level clone analysis. For example, we may be interested in studying the frequency with which each clone class occurs. It is only according to this specification that the clone mining (or extraction) shall be executed.

Post processing after application of heuristics and detection of clones includes, calculating metrics for clones and/or storing clones in clone repository etc. The results of post processing step are fed as input to noise removal phase.

In noise removal, cloning data (or candidates) obtained from clone detection tools and augmented with manual analysis, is filtered. That is, since no technique has perfection some clones might miss being detected while some non clones may be accidentally detected as clones [9]. The heuristics used in pre processing have a role to play here too. For example, if a tool uses 'string matching' algorithm with predefined threshold for detection of large granularity clone (like an entire Java class being duplicated); then noise can be removed on the basis of 'text based filtering'.

Data condensation is concerned with rendering useful clone information more visible. It is intuitive that this can be done through abstraction of information, reducing unnecessary information and emphasizing useful information. Data needs to be condensed in volume, by merging clone sets having similar files or abstracting clone relations to a higher level i.e. from method to file and even directory level.

Finally during action phase of clone mining process, clones are:

- Refactored :

This is an important reengineering activity to put inferences from high level clone analysis into use. Here, one essentially factors out program into modules or 'extracts' and 'pulls' method in a manner such that semantics are preserved while harmful cloning extent is reduced, hence making the program more maintainable.

- Visualized:

Many generic representations of clones are available, namely- clusters, trees and graphs, dot plot (scatter diagram), matrix etc. Visualization is an important aspect of high level clone mining and should be generic and easily comprehendible.

- Documented:

Many useful clone information thus far obtained can be used to enrich the knowledge base or repository, such as identification of some mental template. It is useful to document it and store for future applications.

III. TRAVEL ITINERARY PLANNER- A CASE STUDY

We illustrate few phases of our proposed framework for mining of high level clones. For this purpose, we use a Java web application for travel itinerary. It follows the Model-View-Architecture pattern [10] of development. The overall organization of this project is shown in Fig. 2.

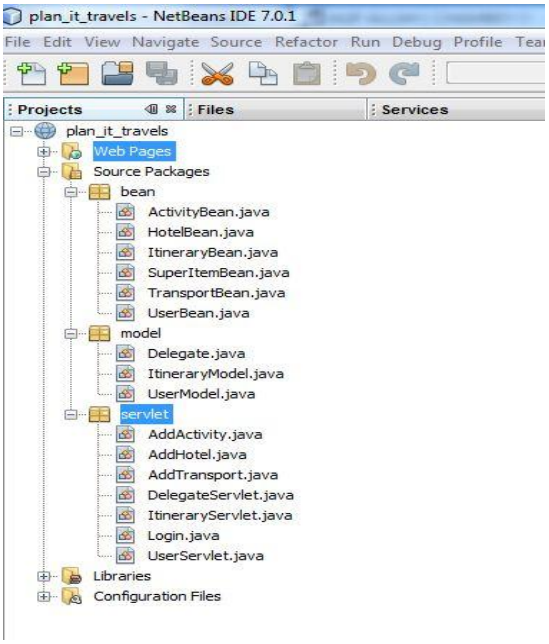

Fig. 2 Travel Itinerary Planner Project

Fig. 3 shows overall clone detection statistics upon employing CloneDr tool.

Clone Detection Statistics	
Statistic	**Value**
File Count	16
Total Source Lines of Code (SLOC)	698
Estimated SLOC before preprocessing	709
Expanded SLOC after preprocessing	709
Total CloneSets	10
Exact-match CloneSets	1
Near-miss CloneSets	9
Number of cloned SLOC	236
SLOC in clones %	33.3%
Estimated removable SLOC	112
Possible SLOC reduction %	15.8%
Possible SLOC reduction in expanded file %	15.8%

Fig. 3 Clone Detection Statistics

File Number	Clone Ratio	Source Lines	File Name
1	100.0%	23	src/java/bean/ActivityBean.java
2	0.0%	41	src/java/bean/HotelBean.java
3	32.4%	34	src/java/bean/ItineraryBean.java
4	40.7%	27	src/java/bean/SuperItemBean.java
5	0.0%	34	src/java/bean/TransportBean.java
6	0.0%	32	src/java/bean/UserBean.java
7	38.3%	47	src/java/model/Delegate.java
8	26.9%	108	src/java/model/ItineraryModel.java
9	21.4%	28	src/java/model/UserModel.java
10	70.3%	37	src/java/servlet/AddActivity.java
11	68.3%	41	src/java/servlet/AddHotel.java
12	68.3%	41	src/java/servlet/AddTransport.java
13	11.8%	93	src/java/servlet/DelegateServlet.jav
14	22.2%	45	src/java/servlet/ItineraryServlet.ja
15	22.7%	44	src/java/servlet/Login.java
16	0.0%	23	src/java/bean/ActivityBean.java

Fig. 4 Files Analyzed

Fig. 4 shows the Java source code files analysed by the tool. Upon application of heuristics (shown in Fig.5) on above project, and with the aid of clone detection tool (CloneDR in our case), detection is made possible.

Detection Parameters	
Value	
Similarity Threshold	95%
Maximum parameter count	6
Minimum Mass (Lines)	6.0
Characters per node	16
Starting height	2

Fig. 5 Heuristics

A. Specification of clones of interest

Our interest here is to identify higher level similarities in the web application.Fig 6 shows clone detection results obtained from CloneDR, consisting of potential clones. The results are incomplete results as they form part of clone candidates capable of being declared as high level clones, only after a series of steps. Potential clone candidates from the above set may be CloneSets 5, 6 and 8. This is because CloneSet 5 has entire compilation unit (a class or .java file) cloned, this represents UML domain model clones [11] with cloned class in UML class diagram. CloneSet 6 and 8 result from same data members in class definition, and can be refactored.

Detected CloneSets (Sorted by Total Mass of CloneSet)					
CloneSet Details	**Clone Mass**	**Clones in CloneSet**	**Parameter Count**	**Clone Similarity**	**Syntax Category [Sequence Length]**
CloneSet1	7	5	6	0.963	statement_sequence[6]
CloneSet2	12	3	2	0.989	statement_sequence[7]
CloneSet3	5	3	6	0.950	statement_sequence[5]
CloneSet4	8	2	2	0.965	statement_sequence[5]
CloneSet5	24	2	0	1.000	compilation_unit
CloneSet6	11	2	3	0.959	class_body_declarations[4]
CloneSet7	2	3	4	0.957	statement_sequence[2]
CloneSet8	9	2	2	0.985	class_body_declaration
CloneSet9	10	2	4	0.959	import_statements
CloneSet10	10	2	1	0.990	import_statements

Fig. 6 Clone Detection Results

B. Noise Removal

Since the concept of high level clones is complex, it needs some manual analysis to augment the automated (tool) detection results. In Fig. 7, we may note that clones of SLOC less than 10 are not of much use in high level clone mining. However, clones beyond 10 SLOC (a threshold set by iterative trial and error method), form a class of high level clones namely-collocated simple clones i.e. recurrence of low level code clones causes high level clones. Thus only clone sets of size 10, 11, 12 and 24 may be useful to us in this case.

CloneSets by Clone Size	
Clone Size (SLOC)	**Number of CloneSets**
2	1
5	1
7	1
8	1
9	1
10	2
11	1
12	1
24	1

Fig. 7 Clone Sets by Clone Size

C. Data Condensation

As per Fig. 8 for our project in discussion, we find that each module has its own add, edit and display methods. These methods are similar semantically and even syntactically (besides renaming); thus they form file level clones. A still higher level clone is discovered if we move up the hierarchy, it is the module level structural clone of Hotel and Transport. The reason for terming them clones is obvious; they contain similar methods (even in same order). Thus method level clones become file level clones, which in turn become module level clones.

After this abstraction, another interesting condensation would be to term 'Hotel-Transport-Activity' a clone set rather than viewing Hotel-Transport, Transport-Activity and Activity-Hotel as clone pairs.

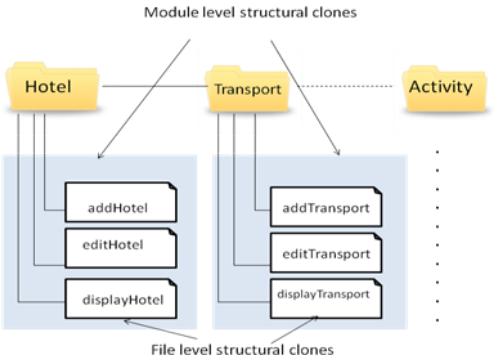

Fig. 8 Raising the Abstraction Level

D. Action

Here, we perform refactoring based on the output of previous step. Later, we visualize the results and give feedback to step 1 of our framework.

Fig. 9 shows output of reverse engineering using StarUML [12] on model package (directory) of our project (refer Fig. 2).

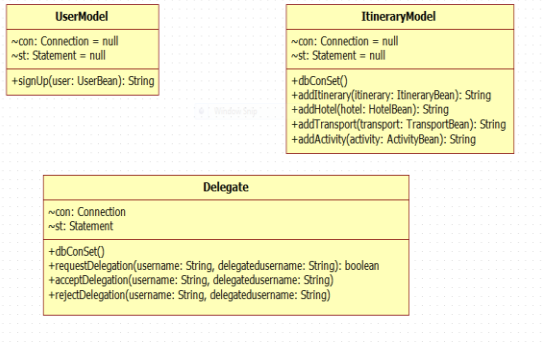

Fig. 9 UML Class diagram

As we know reverse engineering is an important activity in clone detection, we have generated class diagram here to observe some interesting results. Note, only 'model' package (Fig. 2) was reverse engineered. The three classes in the class diagram for 'model' package reveal low level similarity in class attributes namely, con and st. Though, no logical relationships exist among these classes except 'same module' i.e. physical location relationship, yet this result of low level similarity detection is significant to recommend a change in design. Therefore, the absence of high level structural clone is as useful in design decisions and reengineering as is the presence of them. This is a poor design and can be improved by inheriting from a super class (preferably abstract or an interface), the two common attributes. This results in higher level design decision of employing object oriented principle of inheritance for improved design. We reengineer the system by abstracting out lower level similarities i.e. moving bottom to up in the abstraction hierarchy. Good design ultimately leads to reduced maintenance effort and easy extension of system.

IV. BENEFITS OF HIGH LEVEL CLONE MINING

Recognizing higher level similarity can have significant value in the following areas:

Program understanding:

When conceptual level similarity is identified, then it is much clearer to understand the basic concept behind a specific program, which is in line with a well known concept.

Program evolution

Sometimes a hierarchy of similarity exists in software i.e. certain low level clones form clusters and get reflected in design layer too. It is thus necessary to update design as and when the lower level implementation changes and vice versa, otherwise the update anomalies shall propagate the program hierarchy.

Reusable solution

In certain situations, the higher level similarity or structural clone is big enough to form a potential candidate for reusable solution. When similar design patterns are recovered through reverse engineering or similar conception recognition occurs through concept and domain analysis; these software artifacts can be put to use more than once in future.

Reengineering

A situation may be encountered where higher level similarity seems like an unnecessary burden to maintenance, because it does not improve efficiency of the program in any manner. This is poor design and could be improved by suitable refactoring. Alternatively, overtly complex higher level design can also be simplified by reengineering.

V. CONCLUSIONS

Software system maintenance requires thorough program understanding of the system, in order to modify it according to the changing needs. A useful piece of information is higher level similarities in the system, without the knowledge of which, update anomalies may occur. Besides this, storing interesting patterns of cloning at higher abstraction levels can be beneficial for reuse.

In this paper, we propose a four phased framework for extracting high level cloning patterns in software. Our approach reduces the volume of data produced by clone detection tools. In future, we aim to extend our framework for including details of visualization and other sub steps of clone mining process, while including missing steps if any. Also, not all steps could be illustrated due to certain limitations. For e.g, the clone detection tool used here helped in semi automated high level clone analysis but has no facility for clone visualization, for which VisCad[13] may be used separately. Nevertheless, an attempt is made to outline clone mining and a case study of our own project supports it.

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] Buss, E., Mori, R. D., Gentleman, W., Henshaw, J., Johnson, H., Kontogiannis,K., Merlo, E., Muller, H., Paul, J. M. S., Prakash, A., Stanley, M., Tilley, S.,Troster, J., and Wong, K, *Investigating reverse engineering technologies for the CAS program understanding project*. IBM Systems Journal, 33(3):477-500, 1994.

[4] H. A. Basit, S. Jarzabek , *Detecting Higher-level Similarity Patterns in Programs*, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, Lisbon, 2005.

[5] Zhen Ming Jiang, Ahmed E. Hassan, *A Framework for Studying clones in large Software Systems*, in seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007.

[6] Pang-Ning Tan, Michael Steinbach and Vipin Kumar, *Introduction to Data Mining*, Addison –Wesley, 2005.

[7] CloneDR *http://www.semdesigns.com/Products/Clone/*, accessed 2012.

[8] H. A. Basit, Usman Ali and Stan Jarzabek , *Viewing Simple Clones from Structural Clones' perspective*, in *IWSC '11, Honolulu,2011*.

[9] Foyzur Rahman, Christian Bird and Premkumar Devanbu, *Clones: What is that Smell?* 7th IEEE Working Conference on Mining Software Repositories , 2010.

[10] *http://msdn.microsoft.com/enus/library/ff649643.aspx*, 2011.

[11] Harald Storlle, Towards Clone Detection in UML Domain Models, in proceedings of the Fourth European Conference on Software Architecture, ACM Press , 2010.

[12] StarUML URL *http://staruml.tigris.org/* 2011

[13] *http://homepage.usask.ca/~mua237/viscad/viscad.html*

