Toward Identifying Inter-project Clone Sets for Building Useful Libraries

Yoshiki Higo, Kensuke Tanaka, Shinji Kusumoto
Graduate School of Information Science and Technorogy, Osaka University
Suita, Osaka, Japan
{higo,k-tanaka,kusumoto}@ist.osaka-u.ac.jp

ABSTRACT

The present paper discusses how clone sets can be generated from an very large amount of source code. The knowledge of clone sets can help to manage software asset. For example, we can figure out the state of the asset easier, or we can build more useful libraries based on the knowledge.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable libraries*

General Terms

Measurement, Management

Keywords

Code clone, Reengineering for libraries

1. INTRODUCTION

Recently, code clones have attracted a great deal of attention in software engineering, and various studies of code clone related research have been performed. Most of them focus on code clones in a sole software system: for example, effective detection methodologies, identifying candidates for potential bugs or refactoring. The authors think that the knowledge of inter-project clone sets can help to manage software asset. For example, if we can extract common functionalities included in multiple software systems developed in the past, they will be strong candidates for useful libraries. The present paper discusses how we can identify frequent functionalities in a very large amount of source code.

Simone et al. detected code clones from over 7,000 systems with 80 PC workstations [1]. The detection result was visualized as a scatterplot, which made it possible to grasp which systems share code clones. However, they detected clone pairs, not clone sets. Consequently, their analysis cannot reveal how frequently the cloned functionalities occur, or how many systems include a specific functionality. The authors think that the knowledge of clone sets is important to manage software asset efficiently and effectively.

IWSC'10, May 8, 2010, Cape Town, South Africa.

2. TERMS

This subsection explains two terms used in the present paper. The first one is **Clone Pair**, which is a pair of code fragments that are identical or similar to each other. The second one is **Clone Set**. Arbitrary pair of code fragments in a clone set forms a clone pair. That is, there is an equivalence relation (reflexive, transitive, and symmetric relation) between code fragments in a clone set.

3. CLONE SET GENERATION

3.1 Why knowledge of clone sets is required?

The knowledge of clone sets can be an assistance for software asset management (and reverse engineering). For example, in a specific department of a company, if the developers can know how frequently duplicated logics are reimplemented in the past or how many systems share a specific functionality, they will build useful libraries based on the knowledge. Such libraries can avoid unavailing reimplementations, and reduce development cost and period.

Also, detecting inter-project clone sets can reveal copy-and-pasted code created by each developer in the past development, and the team (or the department) can share the code as the organization knowledge. Even if it is impossible to create libraries based on the knowledge, it can be used as a useful *template*. Especially, the knowledge is pretty effective to inexperienced fresh men because they can learn the sophisticated stereotyped code created by skilled people.

3.2 Methods to build clone sets

This research focuses on code clone detection from a very large amount of source code, so that it is impossible to detect code clones from all the target at one time. Consequently, the target is split into multiple units, which are sets of source files of a fixed size. Code clones are detected from each unit, and the results are merged after all the detection finished.

However, it is not easy to build clone sets from multiple detection results. Figure 1 shows a simple example. In this example, two detections are performed; the first one is between software A and B; the second one is software B and C. the former identified code fragments f_A and f_B as code clones, and the latter identified f'_B and f_C as code clones.

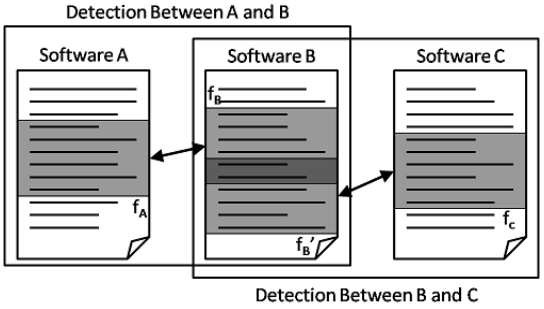


Figure 1: Merging Results of Multiple Detections

Code fragments f_B and f'_B are partly overlapped. If the two code fragments are judged as identical, Code fragments f_A, f_B, and f_C form a single clone set. However, is it no problem to judge as identical? If the number of detections is only 2 like this example, it is appropriate to introduce minimum overlapping threshold for jugding as identical. However, splitting a very large amount of source code generates a large number of units. Consequently, if we use the threshold, it is possible that a clone set indudes two or more code fragments that are not similar to one another at all.

To solve this problem, in this research, structural blocks of programming language are the unit of code clones. In the case of C language, the unit of code clones is *function*, and in the case of Java language, the unit is *method*. After detecting code clones, they are mapped to the unit. If two functions are more similar than the threshold, they are regarded as clone pairs. The present paper proposed two methods to build clone sets from function clone pairs:

Methodology 1: Create an undirected graph from all the clone pairs. Nodes are functions, and edges are clone relation. After creating the graph, local maximum cliques (complete subgraphs) are extracted from it. Extracted cliques are clone sets (see Figure 2(b)).

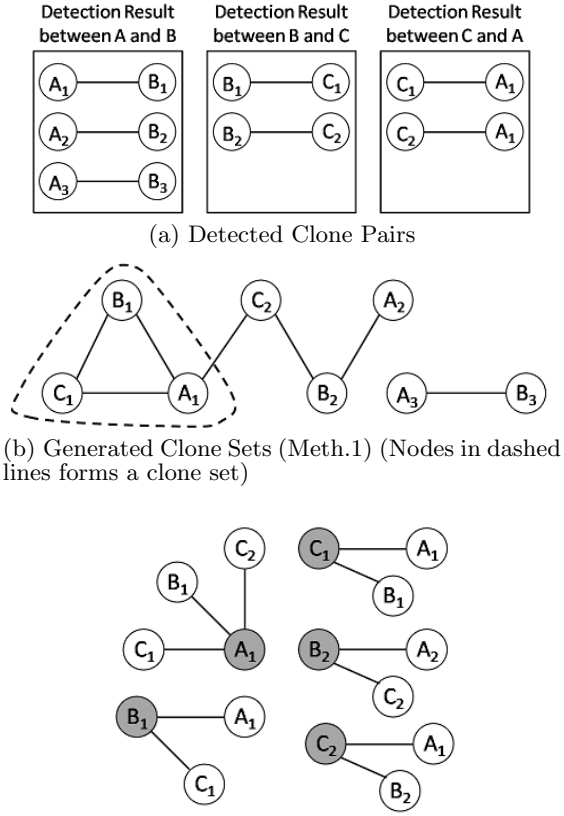
Methodology 2: Create undirected graphs from each clone pair. Created graphs are clone sets (see Figure 2(c)).

In the case of Meth.1, every pair of code fragments included in the extracted cliques is more duplicated than the threshold, so that the cliques are proper as clone sets. However, extracting local maximum cliques requires more cost than *maximum clique problem*, which is known as NP-hard. Consequently, authors think that it is difficult to extract clone sets with Meth.1 within practical timeframe.

On the other hand, in the case of Meth.2, it is possible that some pairs of code fragments in a clone set do not have enough duplication, so that graphs do not satisfy the definition of clone set described in Section 2. However, every function in a clone set have enough duplication with the center node of the clone set (Center nodes have gray color in Figure 2(c)). Consequently, we can think that the center nodes are base functions and, other nodes are derived functions from them.

4. CONCLUSION

The present paper briefly described two methods to build clone sets from a very large amount of source code. In the workshop, we would like to show the results, and discuss how we can use such clone sets data for effective/efficient software development and maintenance.



(a) Detected Clone Pairs

(b) Generated Clone Sets (Meth.1) (Nodes in dashed lines forms a clone set)

(c) Generated Clone Sets (Meth.2) (Every graph is a clone set)

Figure 2: Simple Example of Clone Set Generation (In this example, we extract clone sets that include three or more code fragments)

ACKNOWLEDGMENTS

The present research is being conducted as a part of the Stage Project, the Development of Next Generation IT Infrastructure, supported by the Ministry of Education, Culture, Sports, Science, and Technology of Japan. This study has been supported in part by Grants-in-Aid for Scientific Research (A)(21240002) and (C)(20500033) from the Japan Society for the Promotion of Science.

5. REFERENCES

[1] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Very-Large Scale Code Clone Analysis and Visualization of Open Source Program Using Distributed CCFinder: D-CCFinder. In *Proc. of the 29th International Conference on Software Engineering*, pages 106–115, May 2007.