Detection of Chained Clone and Its Application

Norihiro Yoshida
NAIST / Osaka University, Japan

Overview of my presentation

- Introduction of chained clone detection
  - Basically, it is proposed for refactoring support
- Discussion on other application of chained clone detection
  - We would like to try to apply chained clone detection into supporting other software maintenance activity.

Refactoring

- Refactoring[1] is a way to deal with code clone problem.
- Refactoring is a technique for restructuring an existing code
  - Alter software’s internal structure without changing its external behavior
  - Improve the maintainability of software
  - Number one in the stink parade is duplicate code

Difficulty of Refactoring

- It is difficult to identify refactoring opportunities in large scale source code.
  - Where are code fragments that should be merged into one method?
  - How should they be merged into one method?
    - Extract Method or Pull Up Method Refactoring?

Token-based clone detection for refactoring support

(1/2)

- In many cases, Type2 clone refactoring is easier than Type3 one.
  - Type2 clone set is consist of continuous token sequences
  - it is easy to merge it into one module.
  - Type3 clone refactoring is comprised of more complicated steps
    - It needs to solve syntax differences between code fragments.
- Scalability of detection
  - Token-based clone detection tool is more scalable than syntax-based or semantic-based tools

Token-based clone detection for refactoring support

(2/2)

- Basically, a set of type2 clones DO NOT have semantic similarity.
  - However, target clones for Extract Method or Pull-up Method should be semantic unit.
  - In this context, semantic clone detection is more suitable for refactoring support.
- Most token-based clone detection tools (e.g., CCFinder) DO NOT perform inter-procedural analysis.
  - One functionality is sometimes implemented by a chain of methods.
Proposed tool: Chained clone detection tool

- Detection of clone sets connected by callee-caller relations
- Scalable detection by analyzing only code fragments in CCFinder's output
  - Call-caller relations are inferred by static analysis

Scalable detection by analyzing only code fragments in CCFinder's output

Chained Clone

- For 2 given chained methods CM1 and CM2, we transform them into chained method graphs G1 and G2.
- For G1 and G2, if the following three conditions are satisfied, we call the pair of CM1 and CM2 as a chained clone.
  1. G1 and G2 are isomorphic.
  2. Each pair of the corresponding nodes between G1 and G2, holds a clone relation.

Chained Clone Set

- An equivalence class of chained clones

Applicable Refactorings for Chained Clones

- The following refactoring\[1\] can be applied to merge chained clones.
  - Pull Up Method Refactoring
  - Extract Method Refactoring
  - Extract Super Class Refactoring

Depending on the hierarchy relationship among Java classes having chained clones, we provide appropriate refactoring for each chained clone.

- All chained clones in a chained clone set in single class
  - Extract Method Refactoring is appropriate
- All chained clones in a chained clone set in multiple classes that have common parent classes
  - Pull Up Method Refactoring is appropriate

Research Goal

- Define a set of clone sets having callee-caller relations as a chained clone
- Suggest applicable refactoring pattern for each chained clone based on chained clone categorization

Chained Clone

- It is a better refactoring opportunity than each type2 clone set.
- It is easy to merge each type2 clone set into one module.

Definition of chained clone(1)

- Chained Method
  - A set of methods that hold callee-caller relations
- Chained Method Graph
  - A node represents a method
  - An edge represents a callee-caller relation

Definition of chained clone(2)

- Chained Clone
  - A pair of nodes filled with colored same color is a code clone

Applicable Refactorings for Chained Clones

- All chained clones in a chained clone set in single class
  - Extract Method Refactoring is appropriate
- All chained clones in a chained clone set in multiple classes that have common parent classes
  - Pull Up Method Refactoring is appropriate

Typical Chained Clones Case 1: Extract Method Refactoring

- All the methods in a chained clone that are contained in a single class.

Before Refactoring

After Refactoring

Case Study Overview

- Objective
  - How many chained clone sets exist in actual Java programs?
  - Is it possible to classify chained clone sets and to apply suggested refactorings to them?

- Target software
  - Open source software
    - ANTLR 2.7.4 (47,000 LOC, 285 Classes)
    - JBoss 3.2.6 (640,000 LOC, 3364 Classes)
    - J2EE Application Server
  - Commercial software
    - X (70,000 LOC, 399 Classes)
    - Y (81,000 LOC, 290 Classes)

- We used CCFinder to detect code clones[1].

---

Case Study

Detected chained clone sets (Open source software)

<table>
<thead>
<tr>
<th>Category</th>
<th># of chained clone sets</th>
<th># of methods max</th>
<th># of methods min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext. Met.</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Ext. Sup.</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In category 21, the max of the number of methods in very large language (Java, C#, C++) is similar functionalities for each component class.

The number of chained clone sets in category 31 is large.

ANTLR contains several products. As a result, it has code clones among them.

Case Study

Detected chained clone sets (Commercial software)

<table>
<thead>
<tr>
<th>Category</th>
<th># of chained clone sets</th>
<th># of methods max</th>
<th># of methods min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext. Met.</td>
<td>0</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Ext. Sup.</td>
<td>0</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In only category 21, chained clone sets were detected.

X Software has code clones among several classes which inherit the same component class.

The number of chained clone sets in category 31 is large.

Two packages have similar utility classes.

---

Case Study

Refactoring for Category 31 (ANTLR)

We applied suggested refactorings to chained clone sets in ANTLR.

- Extract Super Class
  - GeneralCharFormatter
    - CSharp CharFormatter
      - escapeChar
    - Java CharFormatter
      - escapeChar

Before Refactoring

After Refactoring

---

Other applications of chained clone detection

Automated defect detection by checking the consistency of chained clones

- clone set A
  - Method a1
    - call
  - Method a2
  - Method a3
  - Method b1
  - call
  - Method b2
  - Method b3
  - Method c1
  - Method c2
  - Method d

Why not cloned? (Defect?)
Other applications of chained clone detection

- Precise and scalable calculation of clone ratio between methods or classes
  - Take into account whether callee methods are cloned

Summary

- We focus on refactoring for chained clones that consist of sets of the methods with callee-caller relations
  - Define chained clone
  - Method to classify chained clones according to their applicable refactoring
  - OSS and Industrial case studies

Future Works

- Apply our proposed method to other Java programs
- Other applications of chained clone detection