Detection of Chained Clone and Its Application

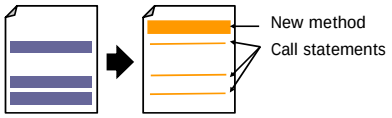
Norihiro Yoshida
NAIST / Osaka University, Japan

Overview of my presentation

- Introduction of chained clone detection
  - N.Yoshida, et al.: "On Refactoring Support Based on Code Clone Dependency Relation", Proc. of METRICS 2005.
  - 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

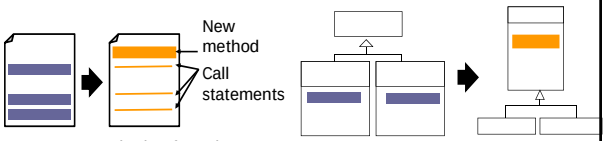
- Refactoring[1] is a way to deal with code clone problem.
- Refactoring is a technique for restructuring an existing code
  - Alter software's internal structure without changing its external behavior
  - Improve the maintainability of software
  - Number one in the stink parade is duplicate code



[1] M. Fowler, Refactoring: improving the design of existing code, Addison Wesley, 1999

Difficulty of Refactoring

- It is difficult to identity 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 Refactroing?



Extract Method Refactoring

Pull Up Method Refactoring

Token-based clone detection for refactoring support (1/2)

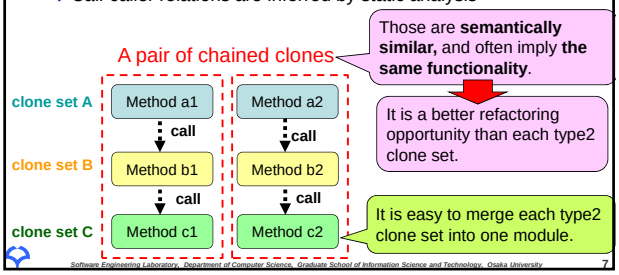
- In many cases, Type2 clone refactoring is easier than Type3 one.
  - Type2 clone set is consist of continuous token sequences
    - it is easy to merge it into one module.
  - Type3 clone refactoring is comprised of more complicated steps
    - It needs to solve syntax differences between code fragments.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

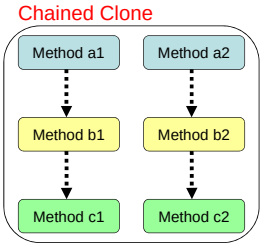
Proposed tool: Chained clone detection tool

- 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



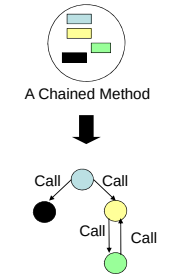
Research Goal

- 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



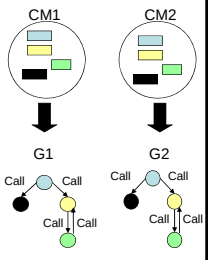
Definition of chained clone(1)

- 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
  - 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



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

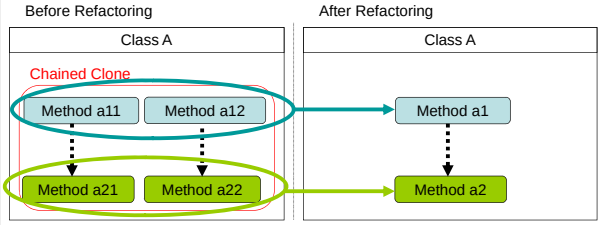
Applicable Refactorings for Chained Clones

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

[1] M. Fowler: Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

Typical Chained Clones
Case 1 : Extract Method Refactoring

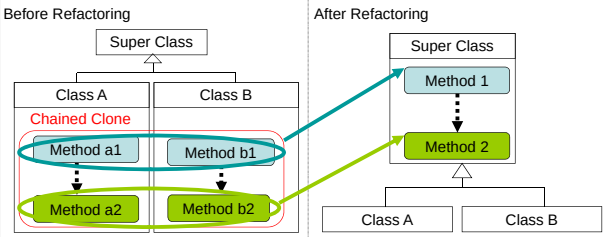
- All the methods in a chained clone that are contained in a single class.



All methods can be merged into two new methods in the class A.
("Extract Method" Refactoring)

Typical Chained Clones
Case 2 : Pull Up Method Refactoring

- All methods in a chained clone belong to classes that have common parent classes.
- All methods of each *chained method are* in the same class respectively.



All methods of each code clone can be merged into a new method in the parent class. (“Pull Up Method” 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)
      - ✓ Compiler-Compiler (Java, C++, C#)
    - JBoss 3.2.6 (640,000 LOC, 3364 Classes)
      - ✓ J2EE Application Server
  - Commercial software
    - X (70,000 LOC, 309 Classes)
    - Y (81,000 LOC, 290 Classes)
- We used CCFinder to detect code clones[1].

[1] T. Kamiya, et. al., CCFinder: A multi-linguistic token-based code clone detection system for large scale source code, *IEEE TSE*, vol.28, no.7, pp.654-670, Jul. 2002.

Case Study
Detected chained clone sets (Open source software)

- ANTLR 2.7.4

Category	# of chained clone sets	# of methods	
		max	min
Ext. Met.	3	4	4
Pul. Met.	6	40	6
Ext. Sup.	1	4	4
Other	0		
Total	10		

In category 21, the max of the number of methods in very large

→ Similar functionalities for each language (Java, C#, C++)

- JBoss 3.2.6

Category	# of chained clone sets	# of methods	
		max	min
Ext. Met.	16	13	4
Pul. Met.	17	8	4
Ext. Sup.	13	29	4
Other	4	44	6
Total	50		

The number of chained clone sets in category 31 is large

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

Case Study
Detected chained clone sets (Commercial software)

- X

Category	# of chained clone sets	# of methods	
		max	min
Ext. Met.	0		
Pul. Met.	9	14	4
Ext. Sup.	0		
Other	0		
Total	9		

In only category 21, chained clone sets were detected

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

- Y

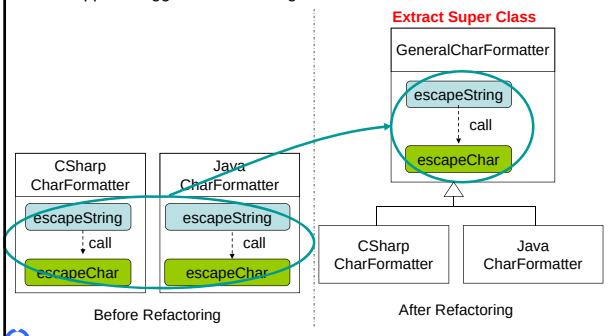
Category	# of chained clone sets	# of methods	
		max	min
Ext. Met.	2	13	13
Pul. Met.	0		
Ext. Sup.	7	26	4
Other	0		
Total	9		

The number of chained clone sets in category 31 is large

→Two packages have similar utility classes

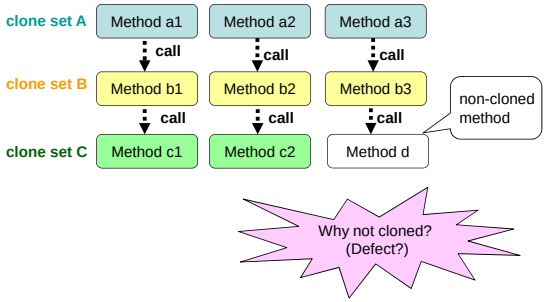
Case Study
Refactoring for Category 31（ANTLR）

- We applied suggested refactorings to chained clone sets in ANTLR.



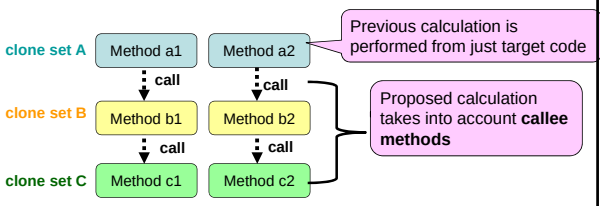
Other applications of chained clone detection

- Automated defect detection by checking the consistency of chained clones



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 refactorings
  - OSS and Industrial case studies
- Future Works
  - Apply our proposed method to other Java programs
  - other applications of chained clone detection