Identification and Selection of Refactorings for Improving Maintainability of Object-Oriented Software

Ah-Rim Han

Dept. of Computer Science
KAIST
2013. 5. 8
Introduction

Main Approach

- Refactoring Candidate Identification
  - Extracting with Dynamic Information Based Rules
  - RER-aware Grouping Entities into Maximal Independent Sets (MISs)
- Refactoring Selection
  - Selecting Multiple Elementary Refactorings

Evaluation

Related Work

Conclusion and Future Work
Introduction
Software Changes and Need of Refactoring

- Object-oriented software undergoes continuous changes with various maintenance activities
  - Ex) addition of new functionalities and correction of bugs

- Since the changes often take place without consideration of the design rationale due to time constraints
  - The design quality of the software may degrade overtime

“Refactoring can serve to restructure the design of object-oriented software without altering its external behavior to improve maintainability” [Fowler’1999]

→ In this thesis, by refactoring, we aim to make software for accommodating changes more easily
Systematic Refactoring Identification Process

- Activities for systematic refactoring identification process

![Diagram showing the systematic refactoring identification process with steps: Source code → Identification → Candidate Refactorings → Accessment → Selection → Selected Refactorings. The Accessment step includes two fitness functions each with a table showing the fitness values for different refactoring candidates.]
Motivation and Research Goal (1/2)

- Refactoring identification using only static information (captured by static source code analysis)
  - Refactorings candidates may be suggested on the pieces of code
    - Never used and never changes having occurred

⇒ When establishing refactoring candidate extraction rules, we use dynamic information
  - Motivated by the previous study [Han’2010] that the data capturing how the system is utilized (i.e., dynamic information) is an important factor for estimating changes
  - Investing efforts on the refactorings involving such codes may effectively reduce maintenance cost
Motivation and Research Goal (2/2)

- Determining refactoring sequences to be applied
  - The best refactoring selection in a greedy way
    - *Inefficient* to select just one best refactoring for the iteration of refactoring identification process

→ For each iteration of refactoring identification process, we select the group of elementary refactorings (multiple refactorings) that can be applied at a same time
  - When grouping elementary refactorings, we consider refactorings’ effect relevance (RER) on maintainability
Thesis Overview

I. Refactoring Candidate Identification
   - Extracting with Dynamic Information based Rules
   - RER-aware Grouping into Maximal Independent Sets

II. Refactoring Selection
   - Refactorings' Effect Evaluation Framework
   - Assessing the Effect of Refactorings
   - Selecting Multiple Refactorings

Selected Multiple Refactorings

Apply Selected Refactorings

Stop

No Improvement

RER: Refactoring Effect Relevance
What Have Been Improved from Proposal

I. Refactoring Candidate Identification
   - Extracting with Dynamic Information based Rules
   - RER-aware Grouping into Maximal Independent Sets

II. Refactoring Selection
   - Refactorings' Effect Evaluation Framework
   - Assessing the Effect of Refactorings
   - Selecting Multiple Refactorings

Selected Multiple Refactorings

Apply Selected Refactorings

No Improvement

Stop

Newly developed

 nhỏ

© KAIST SE LAB 2013
Refactoring Candidate Identification:
Extracting with Dynamic Information Based Rules

Extracting with Dynamic Information Based Rules

Overview

- Operational profile or User scenarios
- Object-oriented source code

- Dynamic dependency
- Static dependency

- Profiled model

- Refactoring candidate extraction rules

- Extracting refactoring candidates

- Collapse Class Hierarchy and Move Method refactorings
Design Problems and Resolving Refactoring

❖ Change Preventing Related Design Problems [Fowler’1999]
   ▪ Many classes are modified when making a single change to a system (e.g., Shotgun Surgery)
   ▪ A single class is modified by many different types of changes (e.g., Divergent Change)

❖ Resolving Refactorings
   ▪ Refactorings should be applied in a way that reduces dependencies of entities (i.e., methods and classes)
     • Collapse Class Hierarchy and Move Method refactorings
Use of Dynamic Dependency

- Dynamic dependency enables to find
  - Entities being really in use
  - Frequency of the relations for those entities

- Dynamic dependencies (DMC)
  - Obtained using dynamic profiling by executing programs
    - Based on dynamic method calls
Refactoring Candidate Extraction Rules

- Rules are defined for reducing dynamic dependencies for identifying refactoring candidates
  - Total of 18 rules (6 types of heuristic design strategies × 3 types of refactorings)
    - When the called methods are implemented in the \( N \) (\( N = 2, 3, 4, 5, 6 \)) different classes (\( N_{\text{Diff}} \))
      - \( \forall (c_i, c_j) \in N_{\text{Diff}_C} \rightarrow \text{Collapse Class Hierarchy} (c_i, c_j) \)
      - \( \forall (m_i, m_j) \in N_{\text{Diff}_M} \rightarrow \text{Move Method} (m_i.\text{class}, m_j) \)
      - \( \forall (m_i, m_j) \in N_{\text{Diff}_M} \rightarrow \text{Move Method} (m_j.\text{class}, m_i) \)
    - When the two methods have many interactions (\( \text{Int} \))
      - \( \forall (c_i, c_j) \in \text{Int}_C \rightarrow \text{Collapse Class Hierarchy} (c_i, c_j) \)
      - \( \forall (m_i, m_j) \in \text{Int}_M \rightarrow \text{Move Method} (m_i.\text{class}, m_j) \)
      - \( \forall (m_i, m_j) \in \text{Int}_M \rightarrow \text{Move Method} (m_j.\text{class}, m_i) \)

- \( c_i (m_i) \) : class (method) entity in a system
- \( x_C (x_M) \) : pairs of classes (methods) extracted as refactoring candidates
Refactoring Candidate Identification:
RER-aware Grouping Entities into Maximal Independent Sets (MISs)
RER-aware Grouping Entities into MISs

Overview

Object Oriented Source Code

Constructing RER-aware Graph

RER-aware Graph

Grouping Entities into Maximal Independent Sets

Maximal Independent Sets of Entities

• RER: Refactoring Effect Relevance
• MIS: Maximal Independent Set
Motivating example

Example: applying Move Method(method m2, class A) and Move Method(method m1, class B)

Expected reduced coupling: -3

Actual reduced coupling: -1

Move Method(method m2, class A) = -2
Move Method(method m1, class B) = -1
Move Method(method m2, class A) = -2
Move Method(method m1, class B) = +1

Delta of coupling for each of Move Method refactoring
G = (V, E) for the corresponding object-oriented program is constructed

- Representing entities (V) and their associations (E)
  - V = \{methods, attributes\}
  - E = \{method_calls (method m1, method m2),
    attribute_assesses_1 (method m1, attribute a1),
    attribute_assesses_2 (method m1, method m2)\}

- Associations:
  1) a method calls the other method (method call)
  2) a method assesses an attribute (attribute_assess_1)
  3) two methods assess the same attribute (attribute_assess_2)
Grouping Entities into MISs

Procedure

Based on G, intermediate groups of entities is obtained by grouping the entities using transitive independent relations

\[(u, v \in V \text{ and } (u, v) \notin E)\]

- Then, remaining entities are assigned on the intermediate groups of entities
  - Until no more entities can be added to any other groups of entities without violating the independence property

- Finally, groups of entities (= MISs) are obtained; and attributes are excluded from MISs
Refactoring Selection
Overview

- MM: Move Method refactoring
- CCH: Collapse Class Hierarchy refactoring
- MIS: Maximal Independent Set of refactorings
- GER: Group of Elementary Refactoring Groups of Elementary Refactorings

Effect of Refactorings

<table>
<thead>
<tr>
<th>GER1</th>
<th>GER2</th>
<th>GER3</th>
<th>GER4</th>
<th>GER5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>-2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Selected Elementary Refactorings

- R1
- R2
- R3

Object-Oriented Source Code

- Creating Link Matrix
- Creating Membership Matrix
- Deriving Delta Table

Refactorings’ Effect Evaluation Framework

- Accessing Effect of Refactorings
- Entities are mapped into Elementary Refactorings
- Transform into Elementary Refactorings

Selected Elementary Refactorings

© KAIST SE LAB 2013
Delta Table (D)

- Provides the method for evaluating elementary refactoring’s effect on maintainability
  - Each element indicates $\Delta \text{maintainability}$
    - Maintainability variance after the application of the elementary refactoring on the current design configuration
  - Maintainability is assessed by the number of external links
    - This number of external links naturally represents lack of cohesion and, at the same time, coupling
    - As a result, by applying refactorings, we aim to reduce this number for improving maintainability

- Computed by matrix computation (fast)
Refactorings’ Effect Evaluation (2/2)

- Delta Table derivation
  - **Formulation**
    - \( L_{\text{Int}} \times M = P_{\text{Int}}; \ L_{\text{Ext}} \times M = P_{\text{Ext}}; \ \text{Inv}(P_{\text{Int}}) - P_{\text{Ext}} = D \)
  - **Example**

### Membership matrix (M)

**Internal link matrix (L_{\text{Int}})**

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
<th>m7</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**External link matrix (L_{\text{Ext}})**

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
<th>m7</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Inversed internal projection matrix**
  - \( \text{Inv}(P_{\text{Int}}) \)

- **External projection matrix**
  - \( P_{\text{Ext}} \)

- **Delta Table (D)**


© KAIST SE LAB
Evaluation
Research Questions

- **[RQ 1.] Effect of dynamic information**
  - Is the dynamic information helpful in identifying refactorings that effectively improve maintainability?

- **[RQ 2.] Effect of multiple refactorings**
  - Do the multiple refactorings help to improve maintainability and reduce search space exploration?
  - Is the RER an important when grouping entities into MISs?
### Experimental Subjects

- **Characteristics and development history for each subject**

<table>
<thead>
<tr>
<th>Name (Version)</th>
<th>jEdit (jEdit-4.3)</th>
<th>Columba (Columba-1.4)</th>
<th>jGit (jGit-1.1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Text editor</td>
<td>Email clients</td>
<td>Distributed source version control system</td>
</tr>
<tr>
<td>Total # of revisions</td>
<td>19501</td>
<td>458</td>
<td>1616</td>
</tr>
<tr>
<td>Number of developers</td>
<td>25</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Class #</td>
<td>952</td>
<td>1506</td>
<td>689</td>
</tr>
<tr>
<td>Method #</td>
<td>6487</td>
<td>8745</td>
<td>5334</td>
</tr>
<tr>
<td>Attribute #</td>
<td>3523</td>
<td>3967</td>
<td>2989</td>
</tr>
</tbody>
</table>
Experimental design

- To assess the capability of refactorings for maintainability improvement, we use the *change simulation*
  - Extract changes as input for change impact analysis
    - Changed methods that had occurred within the examined revisions of the development history
  - Obtain *propagated changes* by performing change impact analysis

We compare the **reduced number of propagated changes**

- approach using dynamic information only (dynamic)
- approach using static information only (static)
- combination of the two approaches (dynamic + static)
Results

Ex) Columba

<table>
<thead>
<tr>
<th></th>
<th>Dynamic+Static</th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average rate</td>
<td>9.09</td>
<td>7.10</td>
<td>7.67</td>
</tr>
<tr>
<td>Percentage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic+Static</td>
<td>100</td>
<td>78.1</td>
<td>84.4</td>
</tr>
</tbody>
</table>

Percentage of reduction for propagated changes: 75 ~ 76%
Effect of Multiple Refactorings

- Experimental design
  - Effect of multiple refactorings
    - Rule-based_RC + MIS (Our approach)
    - Without MIS (Rule-based_RC only)
    
    Comparing 1) Fitness [Han’2013]; 2) # of iterations and Elapsed time (sec)

- Effect of RER
  - Approach considering RER (Our approach)
  - Approach without considering RER
  
  Comparing 1) Fitness [Han’2013]; 2) deviation between actual and expected maintainability

- Rule-based_RC: Approach of rule-based identification of refactoring candidates
- MIS: Approach of grouping into MISs
Results
  
  Summary

- Rule-based_RC: Approach of rule-based identification of refactoring candidates
- MIS: Approach of grouping into MISs

- Rulebased_RCs only: approach without MISs
- Our approach: approach with Rulebased_RCs + MISs
Results

- Ex) jGit

→ In jGit, big refactoring results in local optimum

During the iterative process, it finds the refactoring candidates in the same place
→ Selecting refactorings globally helps to prevent this problem
Effect of RER

Results

Summary

<table>
<thead>
<tr>
<th>Subject</th>
<th>Comparators</th>
<th>Fitness fn.</th>
<th>Accumulated deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>jEdit</td>
<td>Not_RER</td>
<td>0.032379</td>
<td>9246</td>
</tr>
<tr>
<td></td>
<td>Our approach</td>
<td>0.033472</td>
<td>846</td>
</tr>
<tr>
<td>Columba</td>
<td>Not_RER</td>
<td>0.030720</td>
<td>40758</td>
</tr>
<tr>
<td></td>
<td>Our approach</td>
<td>0.037123</td>
<td>481</td>
</tr>
<tr>
<td>jGit</td>
<td>Not_RER</td>
<td>0.023602</td>
<td>13058</td>
</tr>
<tr>
<td></td>
<td>Our approach</td>
<td>0.028192</td>
<td>913</td>
</tr>
</tbody>
</table>

- Not_RER: approach without considering RER
- Our approach: approach considering RER
- Accumulated deviation

\[
\sum_{i=0}^{\text{# of Iteration}} |\text{Expected}_i - \text{Actual}_i|
\]

*Expected*<sub>_i_</sub>: expected maintainability on i-th iteration

*Actual*<sub>_i_</sub>: actual maintainability on i-th iteration
Related Work
Refactoring identification based on static metrics [Tahvildari’2003; Zhao’2006]

- The used metrics are all static
- Neither clear rules for detecting design flaws nor a method of how to apply refactorings
- No quantitative method for evaluating the effect of refactorings
Determining refactoring sequences to be applied by selecting the best refactoring in a greedy way [Tsantalis’2009; Han’2013]
- Inefficient to select just one best refactoring for the iteration of refactoring identification process

Analysis of dependencies or conflicts between refactoring candidates [Mens’2007; Hotta’2012]
- Only considered syntactic dependency
Conclusion and Future Work
Conclusion

- Provide the methods for supporting systematic refactoring identification
  - Develop the method for dynamic information-based identification of refactoring candidates
  - Develop the method for RER-aware grouping entities of MIS and selecting multiple refactorings
Future Work

- We plan to consider more types of refactorings
  - For example, Pull Up Method refactoring and Form Template Method refactoring

- Our framework of refactorings’ effect evaluation
  - Can support to easily extend considering refactorings to other various type of refactorings
    - Because it provides the method of assessment and impact analysis of elementary refactorings
    - The action of big refactoring comprises of elementary refactorings
Thank You.
Reference (1/4)


[Kerievsky'2005] Refactoring to patterns, Pearson Education.

[Jeon'2002(APSEC)] An automated refactoring approach to design pattern-based program transformations in java programs, IEEE Software Engineering Conference in Asia-Pacific.


[Higo'2008(JSME)] A metric-based approach to identifying refactoring opportunities for merging code clones in a java software system, Journal of Software Maintenance and Evolution: Research and Practice.


[Harman'2011(ICSTW)] Refactoring as testability transformation, in: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE.


[Seng'2006(GECCO)] Search-based determination of refactoring opportunities for improving the class structure of object-oriented systems, Proceedings of the 8th annual conference on Genetic and evolutionary computation.


[Tsantalis’2011(CSMR)] Ranking Refactoring Suggestions based on Historical Volatility, Software Maintenance and Reengineering (CSMR), IEEE.


[Liu'2008(IET)] Conflict-aware schedule of software refactorings, Software, IET.

[Mens’2007(SoSyM)] Analysing refactoring dependencies using graph transformation, Software and Systems Modeling.


Reference (4/4)

