Search-based Refactoring: Towards Semantics Preservation

Ali Ouni\textsuperscript{1,2}, Marouane Kessentini\textsuperscript{2}, Houari Sahraoui\textsuperscript{1},
And Mohamed Salah Hamdi\textsuperscript{3}

\textsuperscript{1}DIRO, Université de Montréal, Canada
\textsuperscript{2}CS, Missouri University of Science and Technology, USA
\textsuperscript{3}IT Department, ABMMC, Qatar

28th IEEE International Conference on Software Maintenance

Riva del Garda, September 23-30, 2012
Motivating example

Employee
- ID
- Name
- FamilyName
- Natinality
- DateOfBirth
- Sex
- ...
- ...
- getPhoneNumber()
- calculateLocalTax()
- getAge()
- calculateSalary()
- setMaritalStatus()
- getCurrentNatinality()
- ...
- ...

Car
- IdNumber
- TowingCapacity
- OwnerName
- ...
- getHistoryReport()
- getTowingCapacity()
- setInsuranceNum()
- ...

Position
- PositionId
- Grade
- CompanyName
- ...
- getPosition()
- setGrade()
- ...

defect: Blob
Outline

- Problem statement
- Approach: Multi-Objective Refactoring
- Evaluation
- Conclusion
Problem statement

- **Design defect** introduced during the initial design or during evolution
  - Anomalies, anti-patterns, bad smells…
  - Design situations that adversely affect the development of a software
  - Examples: *Blob, spaghetti code, functional decomposition,* ...
Problem statement

• **Refactoring** to correct them and to improve code quality
  - The process of improving a code after it has been written by changing the internal structure of the code without changing the external behavior *(Fowler et al., ‘99)*
  - Examples: *Move method, extract class, move attribute, ...*

• Refactoring implementation may produce semantic errors
• Automate the refactoring task
• Existing approaches
  – See the refactoring as a single-objective problem
  – Improve the internal structure
  – The semantics is not a major concern
  – Produce semantic errors / incoherencies
  – Fastidious manual inspection is needed

How to find the best compromise between quality improvement and semantics preservation
Approach Overview

Source code with defects

Detection rules

Refactoring operations

Search-based Refactoring (NSGA-II)

Call Graph

Proposed refactorings
Multi-Objective Refactoring

• See the refactoring task as a multi-objective optimization problem
  – Improve software quality
    • Defects correction
  – Preserve domain semantics / coherence
    • Program analysis: call graph, derive metrics, ...
    • Information retrieval: cosine similarity

Meta Heuristic Search Using Multi-Objective Optimization (NSGA-II)
NSGA-II overview

• NSGA-II: Non-dominated Sorting Genetic Algorithm (K. Deb et al., ’02)
NSGA-II adaptation

- Representation of the **individuals**
- Creation of a **population** of individuals
- Creation of new individuals using genetic operators (**crossover** and **mutation**)
- Definition of **fitness functions**
Representation of individuals

- Individual = Refactoring solution
- Sequence of refactoring operations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RO1</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO2</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO3</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO4</td>
<td>inlineClass</td>
</tr>
<tr>
<td>RO5</td>
<td>extractSuperClass</td>
</tr>
<tr>
<td>RO6</td>
<td>inlineMethod</td>
</tr>
<tr>
<td>RO7</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO8</td>
<td>moveMethod</td>
</tr>
</tbody>
</table>
Population: set of refactoring solutions

- Context
- Problem statement
- Multi-Objective Refactoring
- Validation
- Conclusion

RO1  moveMethod
RO2  pullUpAttribute
RO3  extractClass
RO4  inlineClass
RO5  extractSuperClass
RO6  inlineMethod
RO7  extractClass
RO8  moveMethod
RO9  extractClass
RO10 moveMethod

RO1  moveMethod
RO2  pullUpAttribute
RO3  extractClass
RO4  inlineClass
RO5  extractSuperClass
RO6  inlineMethod
RO7  extractClass
RO8  moveMethod
RO9  extractSuperClass

RO1  moveMethod
RO2  pullUpAttribute
RO3  extractClass
RO4  inlineClass
RO5  extractSuperClass
RO6  inlineMethod
RO7  extractClass
RO8  moveMethod
RO9  extractSuperClass
Population creation

- Specify the controlling parameters

<table>
<thead>
<tr>
<th>Refactorings</th>
<th>Controlling parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>move method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>move field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>pull up field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>pull up method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>push down field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>push down method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>inline class</td>
<td>(sourceClass, targetClass)</td>
</tr>
<tr>
<td>extract class</td>
<td>(sourceClass, newClass)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a₁</th>
<th>a₂</th>
<th>...</th>
<th>aₙ</th>
<th>m₁</th>
<th>m₂</th>
<th>...</th>
<th>mₓ</th>
<th>moyenne</th>
</tr>
</thead>
<tbody>
<tr>
<td>m₁</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.42</td>
</tr>
<tr>
<td>m₂</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mₓ</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Genetic Operators

1. Crossover

<table>
<thead>
<tr>
<th>RO1</th>
<th>moveMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO2</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO3</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO4</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO5</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO6</td>
<td>PushDownMethod</td>
</tr>
<tr>
<td>RO7</td>
<td>inlineClass</td>
</tr>
</tbody>
</table>

Solution A

<table>
<thead>
<tr>
<th>RO1</th>
<th>moveMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO2</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO3</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO4</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO5</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO6</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO7</td>
<td>moveAttribute</td>
</tr>
</tbody>
</table>

Child A

2. Mutation

<table>
<thead>
<tr>
<th>RO1</th>
<th>moveMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO2</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO3</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO4</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO5</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO6</td>
<td>pushDownMethod</td>
</tr>
<tr>
<td>RO7</td>
<td>inlineClass</td>
</tr>
</tbody>
</table>

Solution B

<table>
<thead>
<tr>
<th>RO1</th>
<th>moveMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO2</td>
<td>pullUpAttribute</td>
</tr>
<tr>
<td>RO3</td>
<td>moveAttribute</td>
</tr>
<tr>
<td>RO4</td>
<td>moveMethod</td>
</tr>
<tr>
<td>RO5</td>
<td>extractClass</td>
</tr>
<tr>
<td>RO6</td>
<td>extractSubClass</td>
</tr>
<tr>
<td>RO7</td>
<td>inlineClass</td>
</tr>
</tbody>
</table>
Fitness Functions

1. Quality
   - Maximize the number of corrected defects

2. Semantics
   - Minimize domain-semantics errors
Quality Fitness Function

- For each candidate refactoring solution:

\[
Quality = \frac{\#\text{corrected_defects}}{\#\text{detected_defects\_before\_refactoring}}
\]
Semantics Fitness Function

• **Intuition:**
  – The correctness of proposed refactorings increase when applied to semantically connected elements.
Semantics Fitness Function

- Vocabulary-based similarity
  - Vocabulary similarity

- Dependency-based similarity
  - Shared methods call
  - Shared field access
Semantics Fitness Function

- Vocabulary similarity

V1 = \{ NewProjectWizard, myMainFrame, JFrame, createNewProject, PrjInfos, GanttProj, showNewProjectWizard, wizardNewProjectWizard, mainFrame, project, IGanttProj, ... \}

\[ V1 = \begin{pmatrix} 15 \\ 2 \\ 5 \\ 6 \\ 18 \\ 6 \\ 7 \\ 1 \\ 19 \\ 20 \\ 6 \\ ... \end{pmatrix} \]

\[ V2 = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 1 \\ 9 \\ 10 \\ 0 \\ 0 \\ 4 \\ 26 \\ 2 \\ ... \end{pmatrix} \]

Cosine similarity

\[ \Theta = \arccos \left( \frac{V1 \cdot V2}{\|V1\| \cdot \|V2\|} \right) \]
Semantics Fitness Function

• Shared methods
  – Use call graph
    • Assumption: if two classes share many common method calls then they are likely to be semantically related

\[
\text{sharedMethods}(c1, c2) = \alpha \times \text{sharedCallIn} + \beta \times \text{sharedCallOut}
\]
Semantics Fitness Function

• Shared Field access
  – Assumption: two software elements are semantically related if they read or modify the same fields.

\[
\text{sharedFieldsRW}(c_1, c_2) = \frac{\| \text{accessFieldsRW}(c_1) \cap \text{accessFieldsRW}(c_2) \|}{\| \text{accessFieldsRW}(c_1) \cup \text{accessFieldsRW}(c_2) \|}
\]
Semantics Fitness Function

• For one refactoring operation

\[ \text{Sem}(\text{RO}) = \alpha \times \text{vocabulary\_similarity} + \beta \times \text{sharedMethodCall} + \delta \times \text{sharedFields} \]

where \( \alpha + \beta + \delta = 1 \)

• For a refactoring solution

\[
\text{Semantic} = \frac{\sum_{i=0}^{n-1} \text{Sem}(\text{RO}_i)}{n}
\]
Evaluation

- **Two research questions**

  - **RQ1.** To what extent can the proposed approach correct design defects and preserve the semantics when applying refactorings?

  - **RQ2.** To what extent can the semantics preservation improve the results provided by existing work?
Evaluation

• Data: Two large open source Java projects

<table>
<thead>
<tr>
<th>Systems</th>
<th>Number of classes</th>
<th>KLOC</th>
<th>Number of Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>GanttProject v1.10.2</td>
<td>245</td>
<td>31</td>
<td>41</td>
</tr>
<tr>
<td>Xerces-J v2.7.0</td>
<td>991</td>
<td>240</td>
<td>66</td>
</tr>
</tbody>
</table>
Evaluation

- Method: Two metrics
  - defect correction ratio (DCR)
    \[ DCR = \frac{\# \text{corrected defects}}{\# \text{defects before applying refactorin gs}} \]
  - refactoring precision (RP)
    \[ RP = \frac{\# \text{meaningful refactorin gs}}{\# \text{proposed refactorin gs}} \]
# Results & Comparison

<table>
<thead>
<tr>
<th>Systems</th>
<th>Approach</th>
<th>Corrected defects (DCR)</th>
<th>Meaningful refactorings (RP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GanttProject v1.10.2</td>
<td>Multi-objective (NSGA-II)</td>
<td>87% (36</td>
<td>41)</td>
</tr>
<tr>
<td></td>
<td>Single-objective (GA) Kessentini et al. 11</td>
<td>95% (39</td>
<td>41)</td>
</tr>
<tr>
<td></td>
<td>Harman et al. 07</td>
<td>N.A</td>
<td>69% (218</td>
</tr>
<tr>
<td>Xerces-J v2.7.0</td>
<td>Multi-objective (NSGA-II)</td>
<td>78% (51</td>
<td>66)</td>
</tr>
<tr>
<td></td>
<td>Single-objective (GA) Kessentini et al. 11</td>
<td>89% (59</td>
<td>66)</td>
</tr>
<tr>
<td></td>
<td>Harman et al. 07</td>
<td>N.A</td>
<td>63% (262</td>
</tr>
</tbody>
</table>
Approach sensitivity

An example of multiple executions on Xerces-J
Conclusion

• A novel search-based approach for refactoring suggestion
  – Quality vs semantics preservation

• Evaluation
  – Two large open-source systems
  – Our approach succeeded in correcting the majority of defects while preserving the domain-semantics of the original program

• Future Work
  – Extend the semantics approximation
  – Test this approach with other projects
  – Comparative study with other different existing techniques
Thanks for your attention

Questions? Comments? Suggestions?
Selection of best solutions

An example of Pareto optimal solutions for Xerces-J
Dominance Principle

• Problem: How to identify optimal solutions for a given population?
• Pareto optimality
The Blob example

• Definition
  – Procedural-style design leads to one object with numerous responsibilities and most other objects only holding data or executing simple processes.

• Symptoms
  – A Blob is a controller class, abnormally large, with almost no parents and no children. It mainly uses data classes, i.e. very small classes with almost no parents and no children (Brown et al. '98).