The Use of Development History in Software Refactoring Using a Multi-Objective Evolutionary Algorithm

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

\textsuperscript{1}\textsc{Diro}, Université de Montréal, Canada
\textsuperscript{2}\textsc{Cs}, Missouri University of Science and Technology, USA
\textsuperscript{3}\textsc{It} Department, ABMMC, Qatar

Amsterdam, The Netherlands
July 06-10, 2013
Motivating example

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

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

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

defect: Blob
Outline

• Problem statement
• Approach: Multi-Objective Refactoring
• Evaluation
• Conclusion
Design defects

- 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, ...
Refactoring

• **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
Problem statement

• 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
  – The consistency with **development history** is not considered when searching for new refactorings
Problem statement

• Hypothesis

1. Code elements which undergo changes in the past, at approximately the same time, bear a good probability for being semantically related.

2. Code elements that experienced a huge number of refactoring in the past have a good chance for refactoring in the future.

3. Recorded refactorings applied in the past can be used to propose new ones in similar contexts.
Approach Overview

Source code with defects → Defects detection rules → Multi-Objective Refactoring (NSGA-II) → Refactoring operations → Proposed refactorings → Software version archive (change log, recorded refactorings …)
Multi-Objective Refactoring

- See the refactoring task as a *multi-objective* optimization problem
  - Improve software quality: defects correction
  - Maximize the use of development/maintenance history
  - Preserve semantic coherence

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

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

```plaintext
```

- 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)

<table>
<thead>
<tr>
<th>Ref</th>
<th>Refactorings</th>
<th>Controlling parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>move method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>MF</td>
<td>move field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>PUF</td>
<td>pull up field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>PUM</td>
<td>pull up method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>PDF</td>
<td>push down field</td>
<td>(sourceClass, targetClass, field)</td>
</tr>
<tr>
<td>PDM</td>
<td>push down method</td>
<td>(sourceClass, targetClass, method)</td>
</tr>
<tr>
<td>IC</td>
<td>inline class</td>
<td>(sourceClass, targetClass)</td>
</tr>
<tr>
<td>EC</td>
<td>extract class</td>
<td>(sourceClass, newClass)</td>
</tr>
</tbody>
</table>
Population creation

- **Population**: set of refactoring solutions

```
<table>
<thead>
<tr>
<th>MM</th>
<th>PUF</th>
<th>EC</th>
<th>MF</th>
<th>IC</th>
<th>PDM</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>PUF</td>
<td>EC</td>
<td>MF</td>
<td>IC</td>
<td>PDM</td>
<td>MM</td>
</tr>
<tr>
<td>MM</td>
<td>PUF</td>
<td>EC</td>
<td>MF</td>
<td>IC</td>
<td>PDM</td>
<td>MM</td>
</tr>
<tr>
<td>MM</td>
<td>PUF</td>
<td>EC</td>
<td>MF</td>
<td>IC</td>
<td>PDM</td>
<td>MM</td>
</tr>
<tr>
<td>MM</td>
<td>PUF</td>
<td>EC</td>
<td>MF</td>
<td>IC</td>
<td>PDM</td>
<td>MM</td>
</tr>
</tbody>
</table>
```
Genetic Operators

Crossover

Parent 1

| MM | PUF | EC | MF | IC | PDM |

Child 1

| MM | PUF | EC | EC | MM |

Child 2

| MF | IC | PUM | EC | MM |

Parent 2

| MF | IC | PUM | EC | MM |

K=3

Mutation

Parent

| MM | PUF | EC | MF | IC | PDM |

Child

| MM | IC | EC | MM | IC | PDM |

K=2, j=4
Objective functions

1. Quality
   - Maximize the number of corrected defects (ICPC’11)

2. Semantics
   - Minimize semantic errors (ICSM’12)

3. Development/maintenance history reuse
   - Maximize the use of past maintenance and development history to find refactoring opportunities
Quality Objective Function

For each candidate refactoring solution:

\[
Quality = \frac{\#\text{corrected\_defects}}{\#\text{detected\_defects\_before\_refactoring}}
\]
Semantics Objective Function

• **Minimize semantic errors**
  – Vocabulary-based similarity (cosine similarity)
  – Dependency-based similarity (shared method call, shared fields)

• **Intuition:**
  – The meaningfulness of proposed refactorings increase when applied to semantically connected elements.
History-based Objective Function

• **Average of three measures**
  1. Score that characterizes the co-change of elements that will be refactored.
  2. Number of changes applied in the past to the same code elements to modify
  3. Similarity with good refactorings applied in the past to similar code fragments
1. Score that characterizes the co-change of elements that will be refactored.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
History-based Objective Function

2. Number of changes applied in the past to the same code elements to modify

\[ history\_measure2(RO_i) = \sum_{i=1}^{n} t(e) \]

where \( t(e) \) is the number of times that the code element(s) \( e \) was refactored in the past and \( n \) is the size of the list of possible refactoring operations.
History-based Objective Function

3. Similarity with previous refactorings applied to similar code fragments

\[ \text{history\_measure}_3(RO) = \sum_{i=1}^{n} w * s \]

where \( n \) is the number of possible refactorings to use

\( m \) is the number of times that refactoring has been applied in the past

\[ w = \begin{cases} 
2 & \text{if the same refactoring has been applied in the past} \\
1 & \text{if a compatible refactoring has been applied in the past} \\
0 & \text{otherwise} 
\end{cases} \]
Evaluation

• Two research questions

  – RQ1. To what extent the reuse of software development history can improve the results of refactoring suggestion?

  – RQ2. How do the proposed multi-objective approach performs compared to random search, mono-objective approach and other existing work?
Evaluation

- Data: Two large open source Java projects

<table>
<thead>
<tr>
<th>Systems</th>
<th># classes</th>
<th># defects</th>
<th>KLOC</th>
<th># revision commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xerces v2.7.0</td>
<td>991</td>
<td>66</td>
<td>240</td>
<td>7493</td>
</tr>
<tr>
<td>JFreeChart v1.0.9</td>
<td>521</td>
<td>57</td>
<td>170</td>
<td>2009</td>
</tr>
</tbody>
</table>
Evaluation

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

<table>
<thead>
<tr>
<th>Systems</th>
<th>Approach</th>
<th>Algorithm</th>
<th>Correction ratio (DCR)</th>
<th>Meaningful refactorings (RP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFreeChart v1.10.2</td>
<td><strong>Our approach</strong></td>
<td>NSGA-II</td>
<td>86% (49/57)</td>
<td>94% (197/210)</td>
</tr>
<tr>
<td></td>
<td>Ouni et al. CSMR'13</td>
<td>NSGA-II</td>
<td>82% (47/57)</td>
<td>86% (202/234)</td>
</tr>
<tr>
<td></td>
<td>Harman et al. GECCO'07</td>
<td>Pareto opt.</td>
<td>N.A</td>
<td>66% (192/289)</td>
</tr>
<tr>
<td></td>
<td>Kessentini et al. ICPC'11</td>
<td>GA</td>
<td>89% (51/57)</td>
<td>62% (147/236)</td>
</tr>
<tr>
<td></td>
<td>Ouni et al. ICSM'12</td>
<td>NSGA-II</td>
<td>84% (48/57)</td>
<td>77% (157/203)</td>
</tr>
<tr>
<td>Xerces-J v2.7.0</td>
<td><strong>Our approach</strong></td>
<td>NSGA-II</td>
<td>80% (53/66)</td>
<td>96% (282/294)</td>
</tr>
<tr>
<td></td>
<td>Ouni et al. CSMR'13</td>
<td>NSGA-II</td>
<td>79% (52/66)</td>
<td>93% (219/236)</td>
</tr>
<tr>
<td></td>
<td>Harman et al. GECCO'07</td>
<td>Pareto opt.</td>
<td>N.A</td>
<td>63% (251/396)</td>
</tr>
<tr>
<td></td>
<td>Kessentini et al. ICPC'11</td>
<td>GA</td>
<td>88% (58/66)</td>
<td>69% (212/304)</td>
</tr>
<tr>
<td></td>
<td>Ouni et al. ICSM'12</td>
<td>NSGA-II</td>
<td>83% (55/66)</td>
<td>81% (186/228)</td>
</tr>
</tbody>
</table>
Results and comparison

• Comparison
  – NSGA-II
  – GA (mono-objective)
  – Random Search

Defect correction ratio (DCR)

Refactoring precision (RP)
Conclusion

• A novel search-based approach for refactoring suggestion
  – The use of maintenance/development history in software refactoring

• Three objectives to optimize
  – Quality
  – Semantics preservation
  – The use of development history

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

• Future Work
  – Use collected refactorings from different systems and calculates a similarity with not only the refactoring type but also the context (code fragments)
  – Test our approach with other other EA and other projects
Thanks for your attention