Template-based Reconstruction of Complex Refactorings

Kyle Prete, Napol Rachatasumrit, Nikita Sudan, Miryung Kim

Electrical and Computer Engineering
The University of Texas at Austin
Problem: Refactoring Reconstruction

Existing refactoring reconstruction techniques cannot easily identify complex refactorings, which consist of a set of atomic refactorings.
Solution: Ref-Finder

- Ref-Finder expresses each refactoring type in terms of *template logic rules*.
- It uses *a logic programming engine* to infer concrete refactoring instances.
- It covers 63 of the 72 refactoring types in Fowler’s catalog, showing *the most comprehensive coverage*. 
Outline

• Motivation and a survey of existing techniques

• A template-based reconstruction approach

• Evaluation

• Conclusions and future work
Motivation

- Inferred refactorings can help *developers understand* other developers’ modifications.
- To *adapt* broken *client applications*.
- To *empirically study* refactorings when the documentation about past refactorings is unavailable.
A Survey of Refactoring Reconstruction Techniques

1. Demeyer et al.
2. Malpohl
3. Van Rysselberghe and Demeyer
4. Antoniol et al.
5. S. Kim et al.
6. Xing and Stroulia’s UMLdiff and change-fact queries
7. Zou and Godfrey
8. Dig et al.’s Refactoring Crawler
9. Weißgerber and Diehl
10. Fluri et al.’s Change Distiller
11. Dagenais and Robillard
12. M. Kim et al.
<table>
<thead>
<tr>
<th>Action</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Method</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Extract Subclass</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move Class</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Move Field</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Move Interface</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Move Method</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rename Method</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replace Package</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Replace Class</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replace Return</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replace Input Signature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Add Parameter</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Extract Superclass</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Field</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Up Method</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Field</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Down Method</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Parameter</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hide Method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Unhide Method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refactoring Type</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Extract Subsystem</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>✔</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
</tr>
<tr>
<td>Inline Subsystem</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>✔</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
</tr>
<tr>
<td>Form Template Method</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>✔</td>
<td>◊</td>
<td>✔</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
</tr>
<tr>
<td>Replace Inheritance with Delegation</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Replace Delegation with Inheritance</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Inline Class</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Convert Anonymous Class into Inner Class</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Introduce Factory Method</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Introduce Parameter Object</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Encapsulate Field</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Preserve Whole Object</td>
<td>◊</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>◊</td>
<td>◊</td>
<td>◊</td>
</tr>
</tbody>
</table>

The remaining **40 refactoring types in Fowler’s catalog are not handled** by any of existing techniques.
Challenges of Complex Refactoring Reconstruction

- Must find pre-requisite refactorings to identify composite refactorings
- Require information about changes within method bodies
- Require the knowledge of changes to the control structure of a program
Outline

• Motivation and a survey of existing techniques

• A template-based reconstruction approach

• Evaluation

• Conclusions and future work
Approach Overview

- Step 1. Encode each refactoring type as a template logic rule
- Step 2. Extract change-facts from two input program versions
- Step 3. Refactoring identification via logic queries
  - Ref-Finder orders pre-requisite refactorings before composite refactorings
## Predicates

<table>
<thead>
<tr>
<th><strong>LSdiff Predicates</strong></th>
<th><strong>Extended Predicates</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>type</td>
</tr>
<tr>
<td>method</td>
<td>field</td>
</tr>
<tr>
<td>return</td>
<td>fieldoftype</td>
</tr>
<tr>
<td>typeintype</td>
<td>accesses</td>
</tr>
<tr>
<td>calls</td>
<td>subtype</td>
</tr>
<tr>
<td>inheritedfield</td>
<td>getter</td>
</tr>
<tr>
<td>inheritedmethod</td>
<td>addedparameter</td>
</tr>
<tr>
<td>methodbody</td>
<td></td>
</tr>
<tr>
<td>cast</td>
<td></td>
</tr>
<tr>
<td>throws</td>
<td></td>
</tr>
<tr>
<td>methodmodifiers</td>
<td></td>
</tr>
<tr>
<td>fieldmodifiers</td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td></td>
</tr>
<tr>
<td>similarbody(σ)*</td>
<td></td>
</tr>
<tr>
<td>conditional</td>
<td></td>
</tr>
<tr>
<td>trycatch</td>
<td></td>
</tr>
<tr>
<td>variabledelection</td>
<td></td>
</tr>
<tr>
<td>fieldmodifiers</td>
<td></td>
</tr>
<tr>
<td>deletedparameter</td>
<td></td>
</tr>
</tbody>
</table>
Fact-Level Differences

Old Program  before *

```java
type("Foo", ..)
method("Foo.main","main","Foo")
conditional("date.before(SUMMER_START)...")
methodbody("Foo.main", ...)
```

New Program  after *

```java
method ("Foo.notSummer(Date)", "notSummer", "Foo")
```
Fact-Level Differences

Old Program  
before_*

```
type("Foo",..)
method("Foo.main","main","Foo")
conditional("date.before(SUMMER_START)...
methodbody("Foo.main", ...)
```

New Program  
after_*

```
type("Foo",..)
method("Foo.main","main","Foo")
method ("Foo.notSummer(Date)", "notSummer", "Foo")
```

Differences $\Delta FB$  
added_* / deleted_*

```
added_method("Foo.summerCharge", ...)
added_method("Foo.notSummer", ...)
deleted_conditional("date.before(SUMMER_START)...
```
Rule Syntax

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.
Rule Syntax

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.

A rule’s consequent refers to a target refactoring to be inferred.

(deleted_subtype(t1,t2) ∧ (pull_up_field(f,t2,t1) ∨ pull_up_method(m,t2,t1))) ∨ (before_subtype(t1,t2) ∧ deleted_type(t1,n,p) ∧ (push_down_field(f,t1,t2) ∨ push_down_method(m,t1,t2))) ⇒ collapse_hierarchy(t1,t2)
Rule Syntax

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.

A rule’s antecedent refers to the structural constraints before and after the target refactoring.

\[
\text{(deleted subtype}(t1,t2) \\
\lor (\text{pull up field}(f,t2,t1) \lor \text{pull up method}(m,t2,t1))) \\
\lor (\text{before subtype}(t1,t2) \land \text{deleted type}(t1,n,p) \\
\lor (\text{push down field}(f,t1,t2) \lor \text{push down method}(m,t1,t2))) \\
\Rightarrow \text{collapse hierarchy}(t1,t2)
\]
Rule Syntax

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.

A rule’s antecedent may refer to pre-requisite refactorings.

\[
\begin{align*}
(deleted\_subtype(t_1,t_2) & \land (pull\_up\_field(f,t_2,t_1) \lor pull\_up\_method(m,t_2,t_1))) \\
& \lor (before\_subtype(t_1,t_2) \land deleted\_type(t_1,n,p) \\
& \land (push\_down\_field(f,t_1,t_2) \lor push\_down\_method(m,t_1,t_2)))
\end{align*}
\]

\[\Rightarrow collapse\_hierarchy(t_1,t_2)\]
Rule Syntax

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.

The structural constraints are represented in Boolean logic.

\[
\text{(deleted_subtype(t1,t2) \land (pull_up_field(f,t2,t1) \lor pull_up_method(m,t2,t1)))} \\
\lor (\text{before_subtype(t1,t2) \land deleted_type(t1,n,p)}) \\
\land (\text{push_down_field(f,t1,t2) \lor push_down_method(m,t1,t2)}) \\
\Rightarrow \text{collapse_hierarchy(t1,t2)}
\]
Rule Syntax

Example: `collapse hierarchy` refactoring—a superclass and its subclass are not very different. Merge them together.

\[
\text{(deleted_subtype}(t_1,t_2) \land (\text{pull_up_field}(f,t_2,t_1) \lor \text{pull_up_method}(m,t_2,t_1))) \\
\lor (\text{before_subtype}(t_1,t_2) \land \text{deleted_type}(t_1,n,p) \\
\land (\text{push_down_field}(f,t_1,t_2) \lor \text{push_down_method}(m,t_1,t_2))) \\
\Rightarrow \text{collapse_hierarchy}(t_1,t_2)
\]
Encoding Fowler’s Refactorings

- We encoded 63 types but excluded a few because
  - they are too ambiguous,
  - require accurate alias analysis, or
  - require clone detection at an arbitrary granularity.

- Catalog of Template Refactoring Rules, Kyle Prete, Napol Rachatasumrit, Miryung Kim, Technical Report, UT Austin
Refactoring Inference Order

Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.
Refactoring Inference Order

*Example:* **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.
Example: **collapse hierarchy** refactoring—a superclass and its subclass are not very different. Merge them together.
Refactoring Inference Order

Example: *collapse hierarchy* refactoring—a superclass and its subclass are not very different. Merge them together.
**Collapse Hierarchy Inference**

To find a *move field* refactoring

```
deleted_field(f1, f, t1)
∧ added_field(f2, f, t2)
∧ deleted_access(f1, m1)
∧ added_access(f2, m1)
⇒ move_field(f, t1, t2)
```

**Fact-base**

```
before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
```
To find a move field refactoring

\[\text{move\_field}(f, t_1, t_2) \quad \Rightarrow \quad \text{move\_field}(f, t_1, t_2)\]

**Fact-base**

before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
Collapse Hierarchy Inference

Invoke a **move-field** query

∃ f1, ∃ f, ∃ t1, ∃ t2, ∃ f2, ∃ m1,
deleted_field(f1, f, t1)
∧ added_field(f2, f, t2)
∧ deleted_access(f1, m1)
∧ added_access(f2, m1)?

Fact-base

before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
Collapse Hierarchy Inference

Create a new **move field** fact

```
f="color",
t1="PieChart",
t2="Chart"
moves_field("color", "PieChart", "Chart")
```

Fact-base

```
beforesubtype("Chart","PieChart")
deletedsubtype("Chart","PieChart")
deletedfield("PieChart.color","color","PieChart")
addedfield("Chart.color","color","Chart")
deletedaccess("PieChart.color","Chart.draw")
addedaccess("Chart.color","Chart.draw")
moves_field("color","PieChart","Chart")
```
To find a **pull up field** refactoring

\[
\text{move\_field}(f, t_1, t_2) \\
\wedge \text{before\_subtype}(t_2, t_1) \\
\Rightarrow \text{pull\_up\_field}(f, t_1, t_2)
\]

**Fact-base**

```plaintext
before\_subtype("Chart","PieChart")
deleted\_subtype("Chart","PieChart")
deleted\_field("PieChart\_color", "color", "PieChart")
added\_field("Chart\_color", "color", "Chart")
deleted\_access("PieChart\_color", "Chart\_draw")
added\_access("Chart\_color", "Chart\_draw")
move\_field("color", "PieChart", "Chart")
```
To find a *pull up field* refactoring:

\[
\text{move_field}(f, t_1, t_2) \\
\land \text{before_subtype}(t_2, t_1) \\
\Rightarrow \text{pull_up_field}(f, t_1, t_2)
\]

Fact-base:

before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
move_field("color", "PieChart", "Chart")
Collapse Hierarchy Inference

Invoke a pull up field query

$\exists f, \exists t1, \exists t2,$
move_field($f, t1, t2$)
$\land$ before_subtype($t2, t1$)?

Fact-base

before_subtype("Chart", "PieChart")
deleted_subtype("Chart", "PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
move_field("color", "PieChart", "Chart")
Create a new **pull up field** fact

```
f="color",
t1="PieChart",
t2="Chart"
pull_up_field("color", "PieChart", "Chart")
```
Collapse Hierarchy Inference

Create a new collapse hierarchy fact

collapse_hierarchy("Chart", "PieChart")

Fact-base

before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
move_field("color", "PieChart", "Chart")
pull_up_field("color", "PieChart", "Chart")
Collapse Hierarchy Inference

Create a new collapse hierarchy fact

Fact-base

before_subtype("Chart","PieChart")
deleted_subtype("Chart","PieChart")
deleted_field("PieChart.color", "color", "PieChart")
added_field("Chart.color", "color", "Chart")
deleted_access("PieChart.color", "Chart.draw")
added_access("Chart.color", "Chart.draw")
move_field("color", "PieChart", "Chart")
pull_up_field("color", "PieChart", "Chart")
collapse_hierarchy("Chart", "PieChart")
Ref-Finder Eclipse Plug-In
Outline

• Motivation and a survey of existing techniques
• A template-based reconstruction approach
  ● **Evaluation**
• Conclusions and future work
## Evaluation: Two Case Studies

1. Code examples from Fowler’s book
2. Open source projects

<table>
<thead>
<tr>
<th></th>
<th>Version Pairs</th>
<th>Factbase Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jEdit</strong></td>
<td>3 releases</td>
<td>110151~121931</td>
</tr>
<tr>
<td><strong>columba</strong></td>
<td>2 revisions</td>
<td>374016~381893</td>
</tr>
<tr>
<td><strong>carol</strong></td>
<td>9 revisions</td>
<td>12869~39353</td>
</tr>
</tbody>
</table>
Evaluation: Criteria

- Precision—how accurate are the identified refactorings?
- Recall—how many known refactorings were detected?
Ref-Finder finds refactorings with 97% precision and 94% recall.

<table>
<thead>
<tr>
<th>Types</th>
<th>Expected</th>
<th>Found</th>
<th>Precision</th>
<th>Recall</th>
<th>False negatives</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>8</td>
<td>19</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11-20</td>
<td>9</td>
<td>20</td>
<td>0.95</td>
<td>1.00</td>
<td></td>
<td>extract method</td>
</tr>
<tr>
<td>21-30</td>
<td>9</td>
<td>12</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td>preserve whole objects</td>
</tr>
<tr>
<td>31-40</td>
<td>10</td>
<td>13</td>
<td>1.00</td>
<td>0.90</td>
<td>preserve whole objects</td>
<td></td>
</tr>
<tr>
<td>41-50</td>
<td>9</td>
<td>11</td>
<td>1.00</td>
<td>0.89</td>
<td>replace conditionals with polymorphism</td>
<td></td>
</tr>
<tr>
<td>51-60</td>
<td>10</td>
<td>11</td>
<td>1.00</td>
<td>0.90</td>
<td>replace parameters with explicit methods</td>
<td></td>
</tr>
<tr>
<td>61-72</td>
<td>8</td>
<td>14</td>
<td>0.86</td>
<td>0.88</td>
<td>replace type code with state</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>63</td>
<td>100</td>
<td>0.97</td>
<td>0.94</td>
<td>replace magic number with symbolic constants, extract method</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation: Fowler’s

- False positives:
  - Extract Method
  - Replace Magic Number with Constant
- False negative resulted from not being able to find similar body facts.
Evaluation Method: Open Source Software

- Precision: We randomly sampled at most 50 refactorings per version pair ($\sigma=0.85$).

- Recall: We used a threshold ($\sigma=0.65$) and manually inspected them until we found 10 correct refactorings. Then we used a stricter threshold ($\sigma=0.85$) and compared the results with this set.
## Evaluation: Open Source Projects

Ref-Finder finds refactorings with 74% precision and 96% recall.

<table>
<thead>
<tr>
<th>Versions</th>
<th># Found</th>
<th>Prec.</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jEdit</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0-3.0.1</td>
<td>10</td>
<td>0.75</td>
<td>0.78</td>
</tr>
<tr>
<td>3.0.1-3.0.2</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3.0.2-3.1</td>
<td>214</td>
<td>0.45</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Columba</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300-352</td>
<td>43</td>
<td>0.52</td>
<td>0.90</td>
</tr>
<tr>
<td>352-449</td>
<td>209</td>
<td>0.91</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Carol</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62-63</td>
<td>12</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>389-421</td>
<td>8</td>
<td>0.63</td>
<td>1.00</td>
</tr>
<tr>
<td>421-422</td>
<td>147</td>
<td>0.64</td>
<td>0.90</td>
</tr>
<tr>
<td>429-430</td>
<td>48</td>
<td>0.85</td>
<td>1.00</td>
</tr>
<tr>
<td>430-480</td>
<td>37</td>
<td>0.81</td>
<td>1.00</td>
</tr>
<tr>
<td>480-481</td>
<td>11</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>548-576</td>
<td>20</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>576-764</td>
<td>14</td>
<td>0.85</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>774</td>
<td>0.74</td>
<td>0.96</td>
</tr>
</tbody>
</table>
public class TextUtilities {
    public static int findMatchingBracket(Buffer buffer, int line, int offset, int startLine, int endLine)
        throws BadLocationException{
        ...
        TokenMarker tokenMarker = buffer.getTokenMarker();
        TokenMarker.LineInfo lineInfo = tokenMarker
            .markTokens(buffer, line);
        Token lineTokens = lineInfo.firstToken;
        ...
        Buffer.LineInfo lineInfo = buffer.markTokens(line);
        Token lineTokens = lineInfo.getFirstToken();
        ...
    }

    hide_delegate("TokenMarker", "Buffer", "TextUtilities")
Limitations

- Propagation of incorrect inferred refactorings
- Our rule encoding is subject to bias
- Better clone detection mechanisms and API-level refactoring detection needed
Future Work

• Investigate robustness of Ref-Finder in case of floss refactorings [Murphy-Hill et al. 2009]

• Discover refactorings seeded by IDE’s refactoring features

• Compare reconstructed refactorings with recorded refactorings in IDE [Robbes et al. 2008]
Related Work

• Logic-based program representation
  • source code navigation (e.g., Grok, JQuery, CodeQuest, Intentional View)
  • design pattern detection (e.g., DeMIMA)
  • bad-smell detection (e.g., Tourwé et al.)
  • conformance checking (e.g., Eichberg et al.)
Summary

- Ref-Finder uses a template-logic query based approach
- It supports 63 refactoring types out of 72 in Fowler’s catalog.
- It detects complex refactorings by knitting together pre-requisite atomic refactorings with other structural constraints.
- Its overall precision and recall are 0.79 and 0.95.
Questions?