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.

	1	2	3	4	5	6	7	8	9	10	11	12
Extract Method	✔	✔	◇	◇	◇	✔	✔	✔	✔	✔	✔	◇
Extract Subclass	✔					✔						
Move Class	✔		✔	✔	✔	✔	✔	✔	✔	✔	◇	✔
Move Field	✔		✔	✔	✔	✔	✔	✔	✔	✔	◇	✔
Move Interface	✔		✔	✔	✔	✔	✔	✔	✔	✔	◇	✔
Move Method	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rename Method	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Replace Package	✔	◇	✔	◇	✔	✔	✔	✔	✔	✔	◇	✔
Replace Class	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	◇	✔
Replace Return		◇	◇	◇	◇	◇	✔	◇	✔	✔	◇	✔
Replace Input Signature	✔	◇	◇	◇	◇	✔	✔	◇	✔	✔	✔	◇
Add Parameter	✔	◇		◇	✔	✔	✔	✔	◇	✔	◇	✔
Extract Superclass	✔					✔						
Pull Up Field	✔					✔						
Pull Up Method	✔					✔						
Push Down Field	✔					✔						
Push Down Method	✔					✔						
Remove Parameter	✔	◇			✔	✔	✔	✔	✔	◇	◇	✔
Hide Method	◇	◇				✔	✔	✔		◇	◇	
Unhide Method	◇	◇				✔	✔	✔		◇	◇	

	1	2	3	4	5	6	7	8	9	10	11	12
Extract Subsystem	◇					✔	◇	◇		◇	◇	◇
Inline Subsystem	◇			◇		✔	◇	◇		◇	◇	◇
Form Template Method		◇	◇	◇		✔		✔		◇	◇	
Replace Inheritance with Delegation						✔						
Replace Delegation with Inheritance						✔						
Inline Class						✔					✔	✔
Convert Anonymous Class into Inner Class						✔						
Introduce Factory Method						✔						
Introduce Parameter Object						✔						
Encapsulate Field						✔						
Preserve Whole Object	◇					✔				◇		◇

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

LSdiff Predicates		Extended Predicates	
package	type	methodbody	conditional
method	field	cast	trycatch
return	fieldoftype	throws	variabledeclation
typeintype	accesses	methodmodifiers	fieldmodifiers
calls	subtype	parameter	**similarbody(σ)***
inheritedfield		getter	setter
inheritedmethod		addedparameter	deletedparameter

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")
```

Fact-Level Differences

Old Program **before_***

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

- **set difference**

New Program **after_***

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

=

Differences (ΔFB) **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.

```
 (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 may refer to pre-requisite refactorings.

```
 (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.

The structural constraints are represented in Boolean logic.

```
 (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.

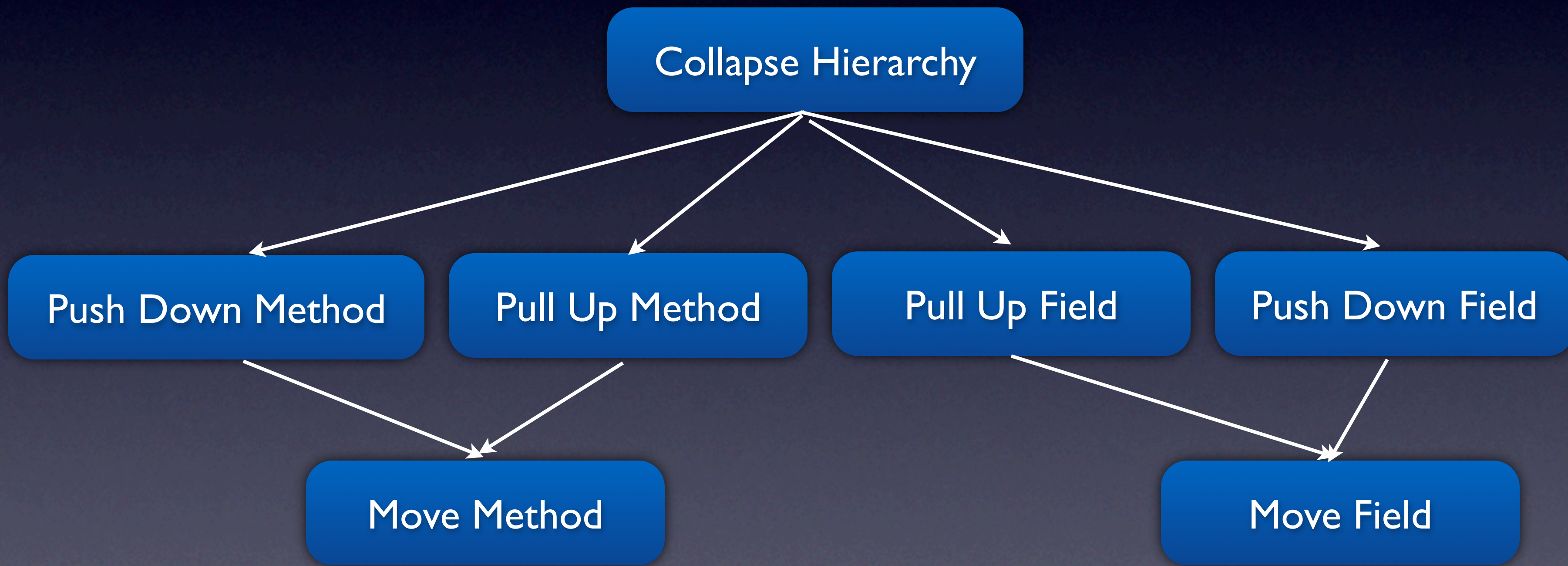
```
 (deleted_subtype(t1,t2)
∧(pull_up_field(f,t2,t1) ∨ pull_up_method(m,t2,t1)))
∨(before_subtype(t1,t2) ∧ deleted_type(t1,n,p)
∧(push_down_field(f,t1,t2) ∨ push_down_method(m,t1,t2))
⇒collapse_hierarchy(t1,t2)
```

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*

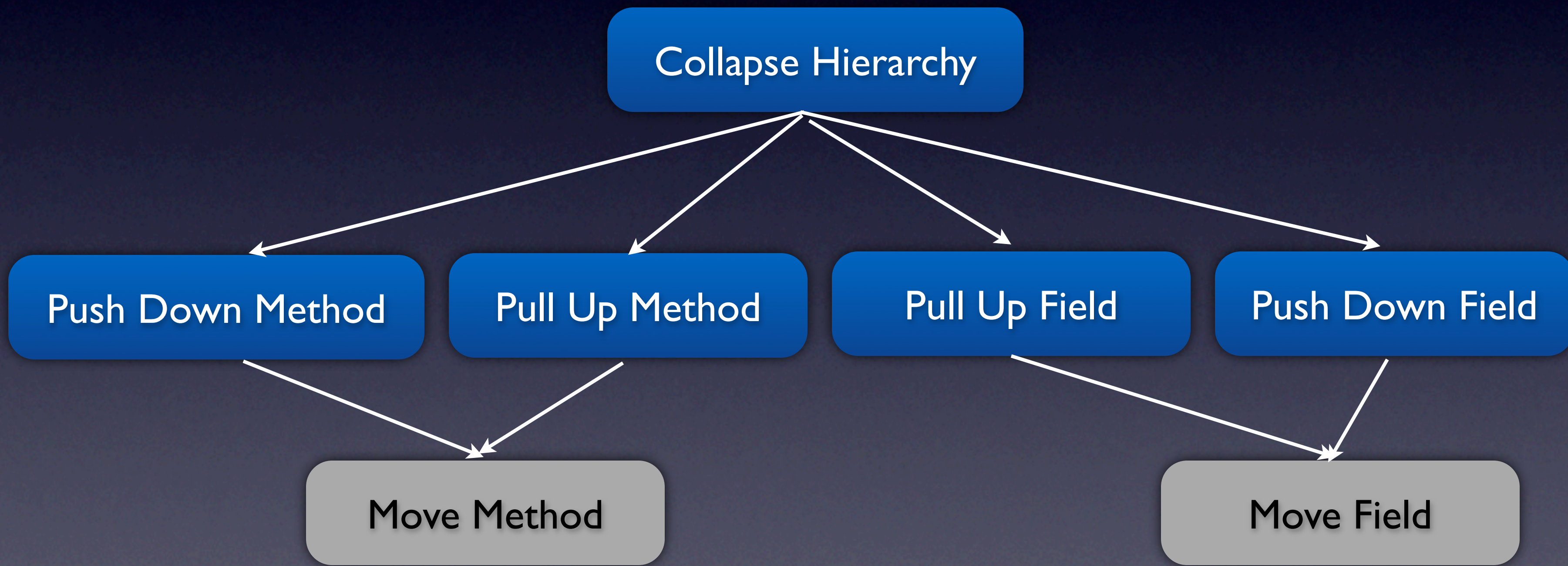
Refactoring Inference Order

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



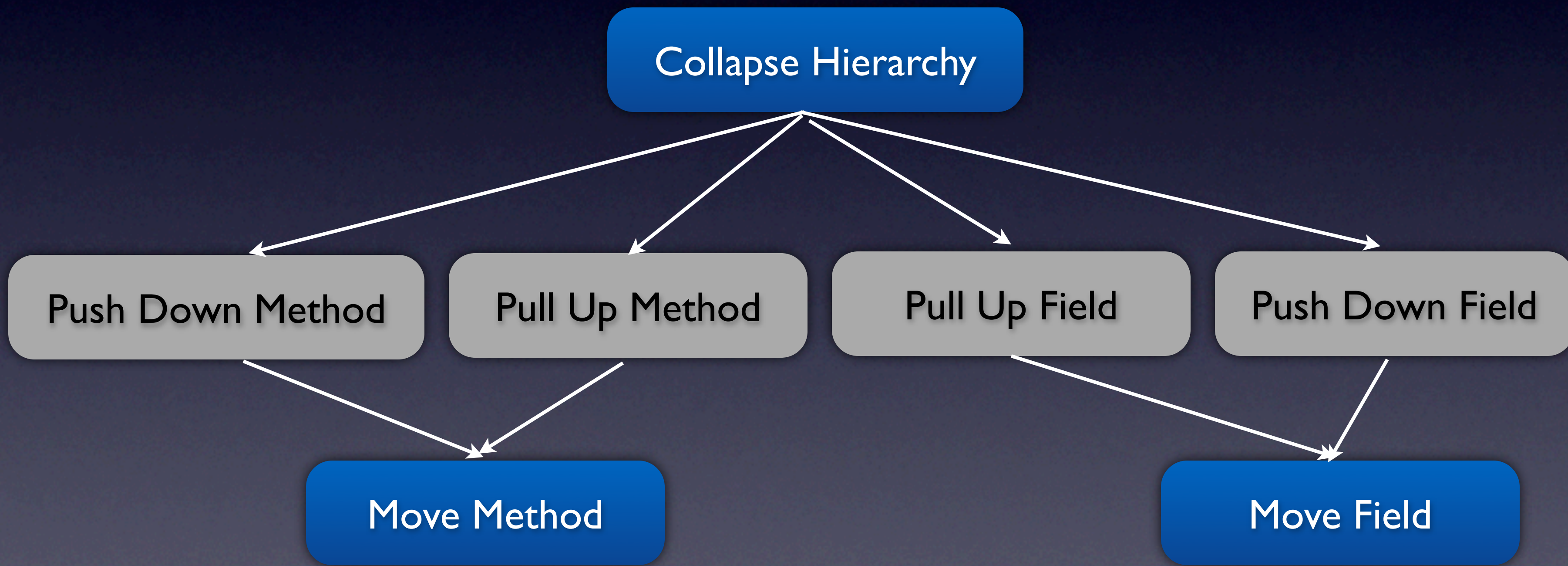
Refactoring Inference Order

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



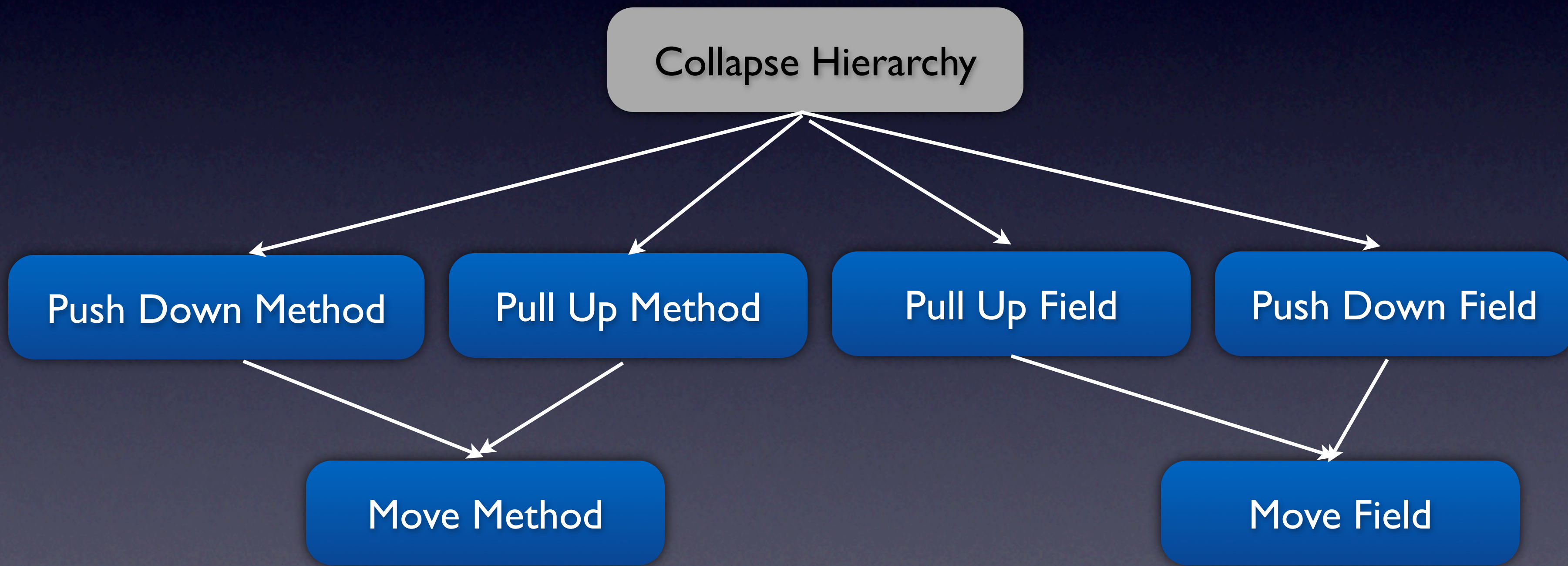
Refactoring Inference Order

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.



Collapse Hierarchy Inference

Collapse
Pull Up
Move

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")
```

Collapse Hierarchy Inference

Collapse

Pull Up

Move

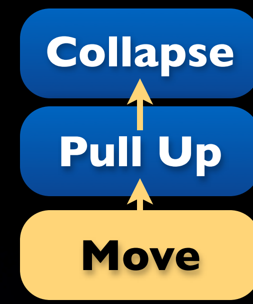
To find a ***move field*** refactoring

```
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")
```

Collapse Hierarchy Inference



Invoke a ***move-field*** query

$\exists$ f1, $\exists$ f, $\exists$ t1, $\exists$ t2, $\exists$ f2, $\exists$ 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

Collapse

Pull Up

Move

Create a new ***move field*** fact

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

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

Collapse
Pull Up
Move

To find a ***pull up field*** refactoring

```
move_field(f, t1, t2)
∧ before_subtype(t2,t1)
⇒ pull_up_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")
move_field("color", "PieChart", "Chart")
```

Collapse Hierarchy Inference

Collapse

Pull Up

Move

To find a ***pull up field*** refactoring

$\boxed{\texttt{move\_field(f, t1, t2)} \wedge \texttt{before\_subtype(t2,t1)}}$
$\Rightarrow$ **pull_up_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")
move_field("color", "PieChart", "Chart")
```

Collapse Hierarchy Inference

Collapse

Pull Up

Move

Invoke a ***pull up field*** query

```
∃ f, ∃ t1, ∃ t2,
move_field(f, t1, t2)
∧ 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")
```

Collapse Hierarchy Inference

Collapse
Pull Up
Move

Create a new
pull up field fact

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

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

Collapse

Pull Up

Move

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")
```

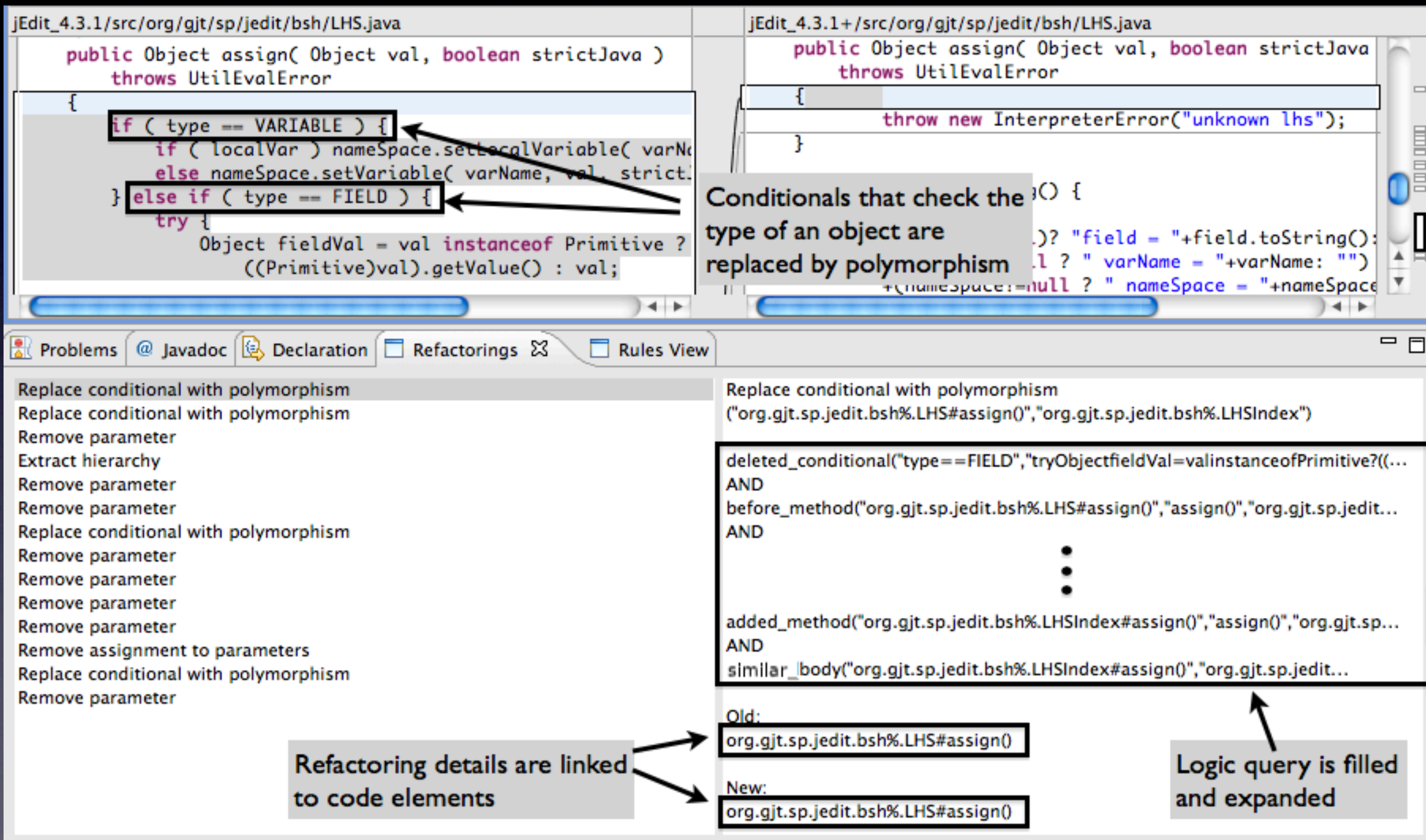
Collapse Hierarchy Inference

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

	Version Pairs	Factbase Size
jEdit	3 releases	110151~121931
columba	2 revisions	374016~381893
carol	9 revisions	12869~39353

Evaluation: Criteria

- Precision—how accurate are the identified refactorings?
- Recall—how many known refactorings were detected?

Evaluation: Fowler's

Ref-Finder finds refactorings with 97% precision and 94% recall.

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

Evaluation: Fowler's

- False positives:
  - *Extract Method*
  - *Replace Magic Number with Constant*
- False negative resulted from not being able to find `similarbody` facts.

Evaluation Method: Open Source Software

- Precision: We randomly sampled at most 50 refactorings per version pair (σ=0.85).
- Recall: We used a threshold (σ =0.65) and manually inspected them until we found 10 correct refactorings. Then we used a stricter threshold (σ=0.85) and compared the results with this set.

Evaluation: Open Source Projects

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

	Versions	# Found	Prec.	Recall
jEdit	*3.0-3.0.1*	*10*	*0.75*	*0.78*
	3.0.1-3.0.2	1	1.00	1.00
	3.0.2-3.1	214	0.45	1.00
Columba	300-352	43	0.52	0.90
	352-449	209	0.91	1.00
Carol	62-63	12	1.00	1.00
	389-421	8	0.63	1.00
	421-422	147	0.64	0.90
	429-430	48	0.85	1.00
	430-480	37	0.81	1.00
	480-481	11	0.91	0.90
	548-576	20	1.00	1.00
	576-764	14	0.85	1.00
Total		774	0.74	0.96

True Positive Example: *Hide Delegate (jEdit 3.0.2-3.1)*

```
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?