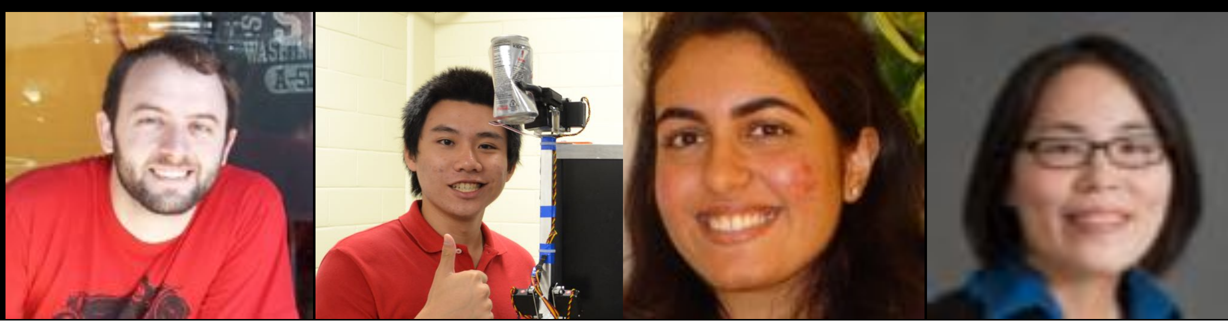
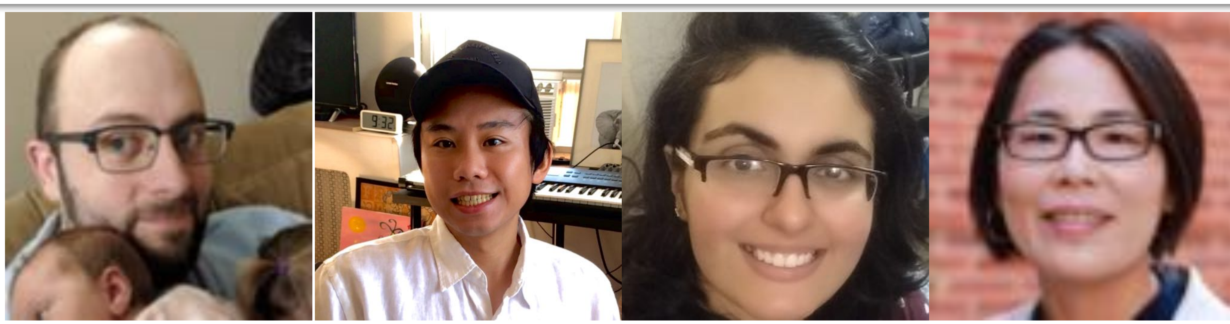
36th IEEE International Conference on Software Maintenance and Evolution (ICSME 2020) Adelaide, Australia

Most Influential Paper from ICSM 2010

"Template-based Reconstruction of Complex Refactoring"

by Kyle Prete, Napol Rachatasumrit, Nikita Sudan, and Miryung Kim





UCLA

What was 2010 like?

Earthquake, Port-au-Prince, Haiti

Air Travel Disruption from Volcanic Eruption, Iceland

ICSE 2010, South Africa, Cape Town

SOUTH AFRICA 2010
FIFA WORLD CUP

2nd year Assistant Professor: Miryung Kim

Deadlines sketched on the white board

Boxes of papers on book shelves

1st year graduate student: Kyle Prete

A fresh graduate from Vanderbilt U in 2010

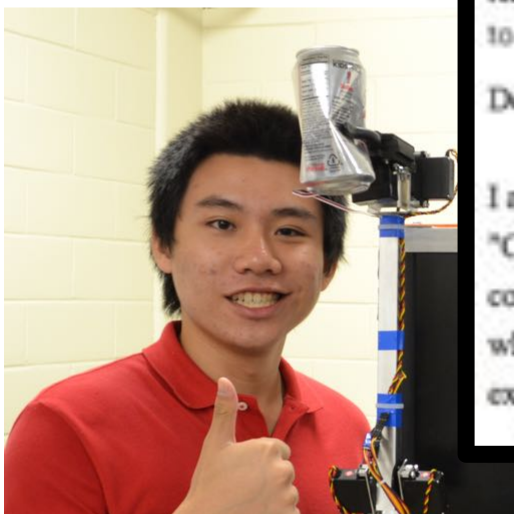
RA offer   Inbox ×

Miryung Kim <miryung@ece.utexas.edu>
to Kyle, Stephanie

Dear Kyle,

I have talked with your references and I am very excited about doing research with you! I am pleased to see your motivation and determination to pursue a Ph.D degree in software engineering and I believe our interests are very aligned.

2nd Year undergraduate: Napol Ra



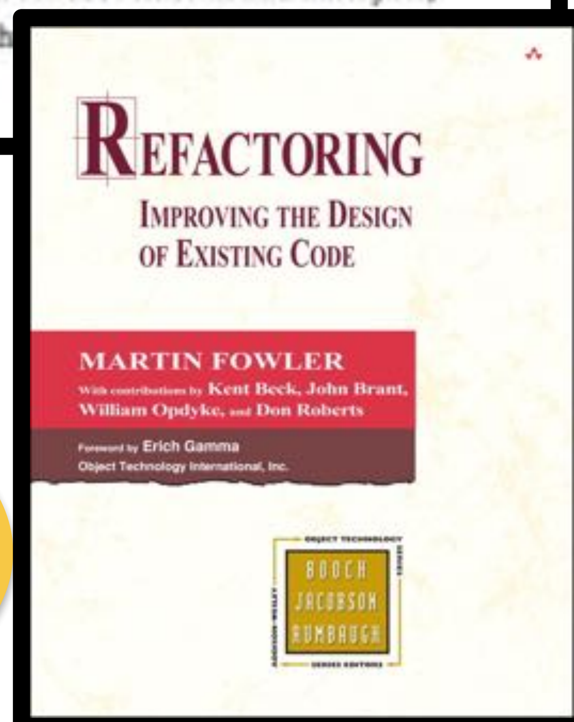
A sophomore in Math and ECE in 2010

Research Assistant Opportunity Inbox

Napol Rachatasumrit the_decz@hotmail.com via ec... Thu, Feb 4, 2010, 7:21 AM
to miryung

Dear Dr. Kim

I am writing to apply for a research assistant opportunities. According to the email, I found that "Coping with Evolution in Software Reuse" is the most interesting topic. I always face with the complication in reusing existing library, especially when I worked in robot clubs in freshmen year, where there were many subgroups working separately. I believe th
experience and a foundation for neat programming style.



Do you know how to program?
Do you know Java?

Could you please read this book and let me know your thoughts?

1st Year graduate Student: Nikita Sudan

A fresh graduate from U Maryland in 2010

applying to UTexas, Dr. Vibha Sazawal's student

Project-TemplateRefactoring ×

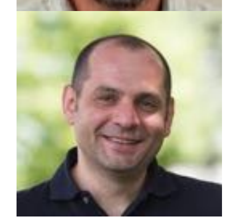
Nikita Sudan <nsudan@gmail.com> Wed, Jan 14, 2009, 11:15 PM

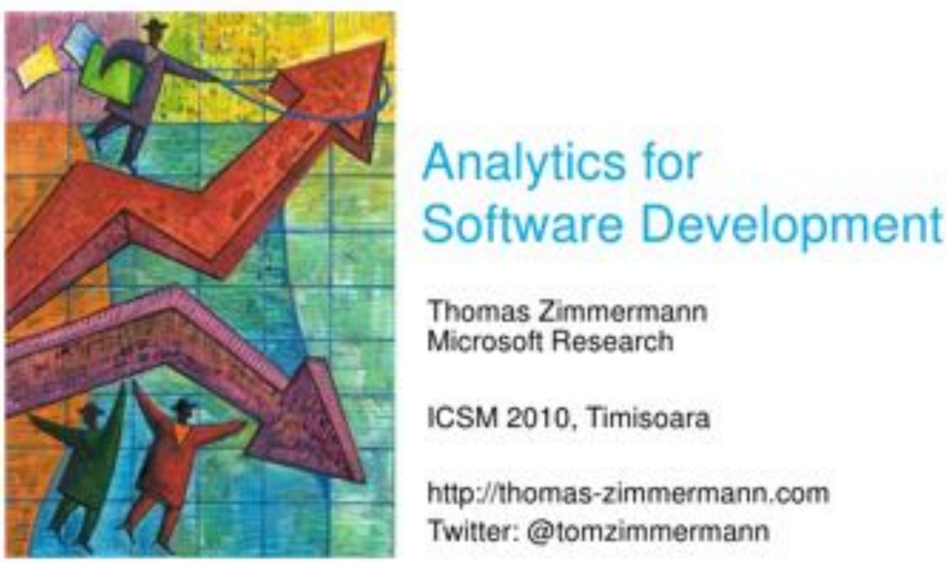
to miryung

Dear Dr. Kim,

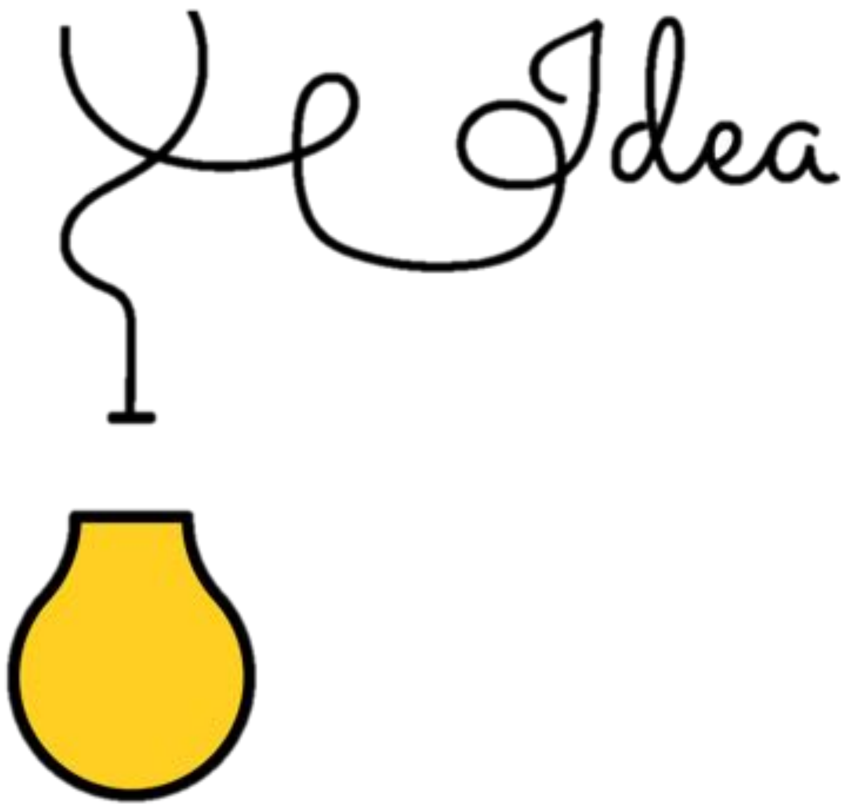
Hope you are doing well. I am a senior Computer Engineering and Economics (double) major studying at the University of Maryland, College Park. I have been doing research under Dr. Vibha Sazawal, Assistant Professor, Department of Computer Science, UMD on the topic of Modeling Software Evolution using Game Theory. We recently submitted a paper to the International Conference on Software Processes, 2009.

ICSM 2010 in Timisoara



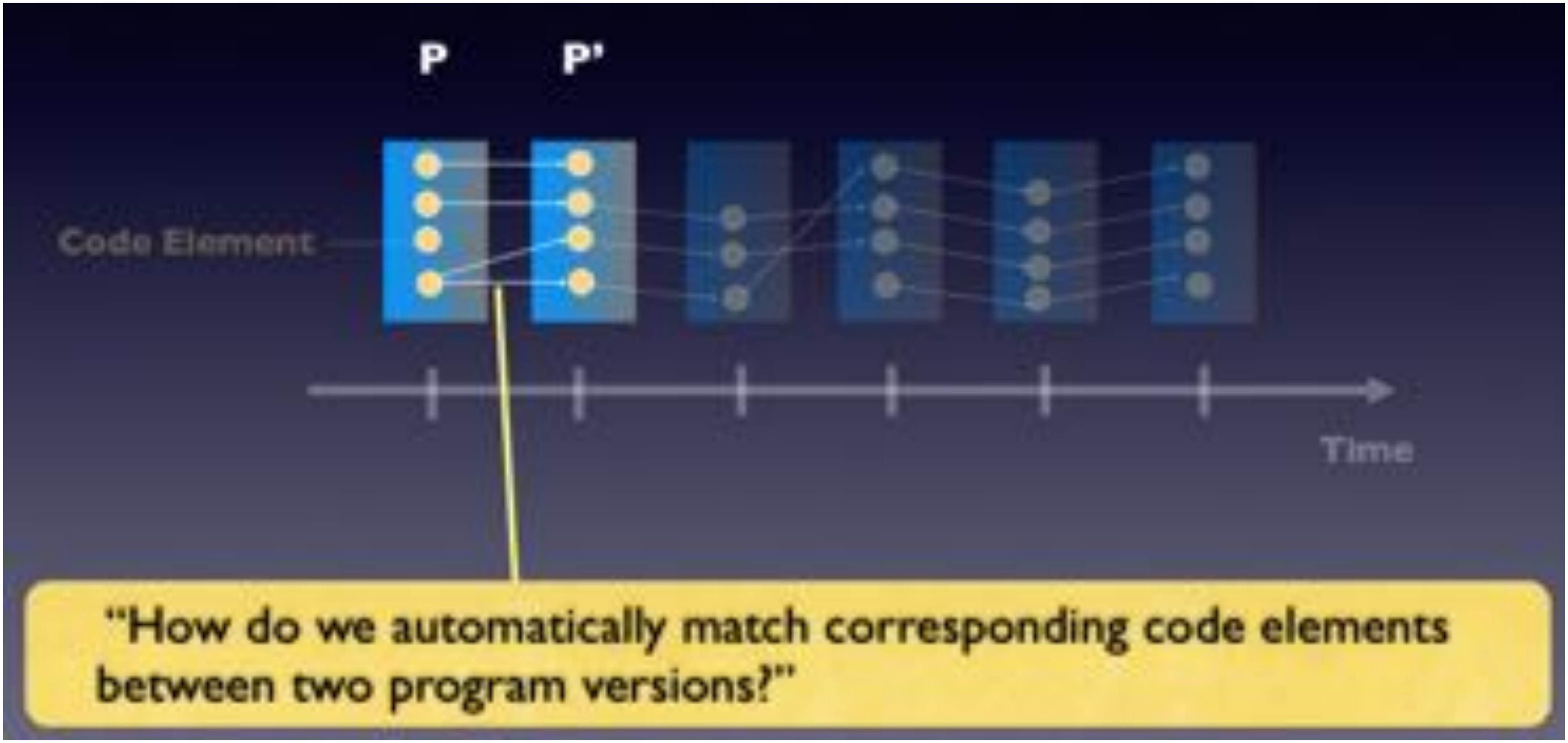


What ideas have motivated and inspired RefFinder?

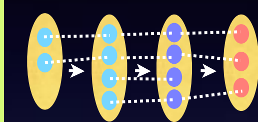


Dagstuhl: Multiversion Program Analysis in 2005

Dagstuhl: Multiversion Program Analysis in 2005



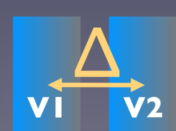
Miryung's PhD @ University of Washington



Analyses of Software Evolution

- Evolution of Code Clones

High-level changes are often systematic at a code level



Automatic Inference of High-Level Change Descriptions

- Rule-based Change Representations
- Rule Learning Algorithms

Miryung's PhD: Discovering Systematic Changes as Rules

Changed Code		
File Name	Status	Lines
DummyRegistry	New	20 lines
AbsRegistry	New	133 lines
JRMPRegistry	Modified	123 lines
JeremieRegistry	Modified	52 lines
JacORBCosNaming	Modified	133 lines
IIOPCosNaming	Modified	50 lines
CmiRegistry	Modified	39 lines
NameService	Modified	197 lines
NameServiceManager	Modified	15 lines
Total Change: 9 files, 723 lines		

```
- public class CmiRegistry implements
NameService {
+ public class CmiRegistry extends
AbsRegistry implements NameService {
-     private int port = ...
-     private String host = null
-     public void setPort (int p) {
-        if (TraceCarol. isDebug()) { ...
-        }
-      }
-      public int getPort() {
-        return port;
-       }
```

Each rule represents ***systematic changes*** by relating groups of change facts. These rules are automatically inferred using ***inductive logic programming***.

```
∀m ∀t past_method(m, "setHost", t) ∧
past_subtype("Service", t)
⇒ deleted_calls(m, "SQL.exec")
[except t="NameSvc" m="NameSvc.setHost"]
```

Inspiration for RefFinder (1) Logical Queries for Code Search

Type-Oriented Logic Meta Programming

Kris De Volder

CodeQuest: Querying Source Code with DataLog

Elnar Hajiyev[1], Mathieu Verbaere[1], Oege de Moor[1] and Kris de Volder[2]

[1] Programming Tools Group
University of Oxford
United Kingdom

[2] Software Practices Lab
University of British Columbia
Vancouver, Canada

Navigating and Querying Code Without Getting Lost

Doug Janzen and Kris De Volder
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver BC Canada V6T 1Z4

Maintaining software through intentional source-code views

Kim Mens
Département INGI
Univ. catholique de Louvain
Louvain-la-Neuve, Belgium
Kim.Mens@info.ucl.ac.be

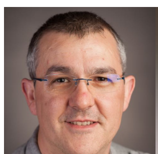
Tom Mens*
Programming Technology Lab
Vrije Universiteit Brussel
Brussels, Belgium
Tom.Mens@vub.ac.be

Michel Wermelinger†
Departamento de Informática
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
mw@di.fct.unl.pt

Inspiration for RefFinder (2) Fine Grained Diff & Change Types

Detecting Merging and Splitting using Origin Analysis


UMLDiff: An Algorithm for Object-Oriented Design Differencing



Automated Detection of Refactorings in Evolving Components

Danny Dig, Can Comertoglu, Darko Marinov, and Ralph Johnson

Department of Computer Science
University of Illinois at Urbana-Champaign



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 11, NOVEMBER 2007 725

Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction


SpyWare: A Change-Aware Development Toolset

Romain Robbes Michele Lanza

Inspiration for RefFinder (3): Need for Domain Knowledge

Statistical Relational Structure Learning

Inductive Logic Programming

Genetic Programming

Heuristic Search

Infer too many "uninteresting" change rules
⇒ must encode inductive bias explicitly

Excerpts from Original ICSM 2010 Talk

Motivation: Refactoring-Aware Code Review

- Developers can benefit from refactoring information when they investigate ***complex non-local edits*** during peer code reviews.
- **Problem:** How can we automatically identify ***the locations and types of refactoring*** from two program versions?

Challenges: 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

Approach: Logic Query-based Refactoring Reconstruction

- 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", ...)
```

− **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.*

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.*

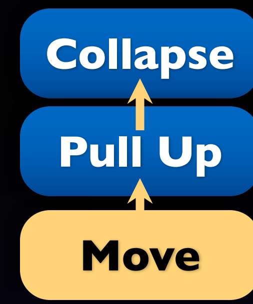
A rule's antecedent may refer to pre-requisite refactorings.

```
(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)
```

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.

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")
```

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

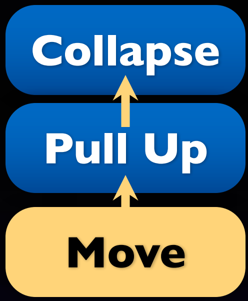
Invoke a ***move-field*** query

```
∃ 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"
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

```
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

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

Pull Up

Move

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

Collapse

Pull Up

Move

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”)
```

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	1		
11-20	9	20	0.95	1		extract method
21-30	9	12	1	1		
31-40	10	13	1	0.9	preserve whole objects	
41-50	9	11	1	0.89	replace conditionals with polymorphism	
51-60	10	11	1	0.9	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		

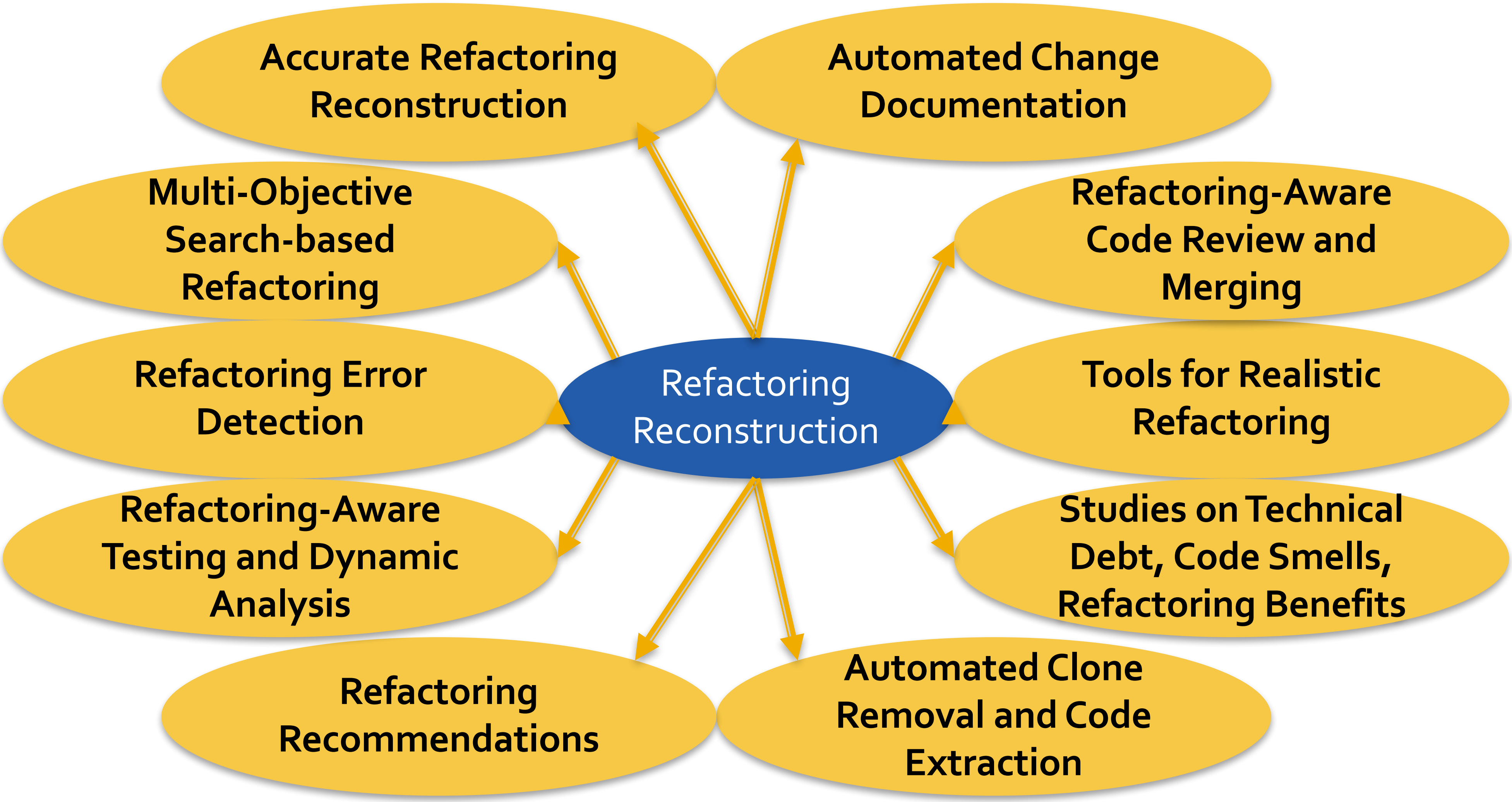
Evaluation: Open Source

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

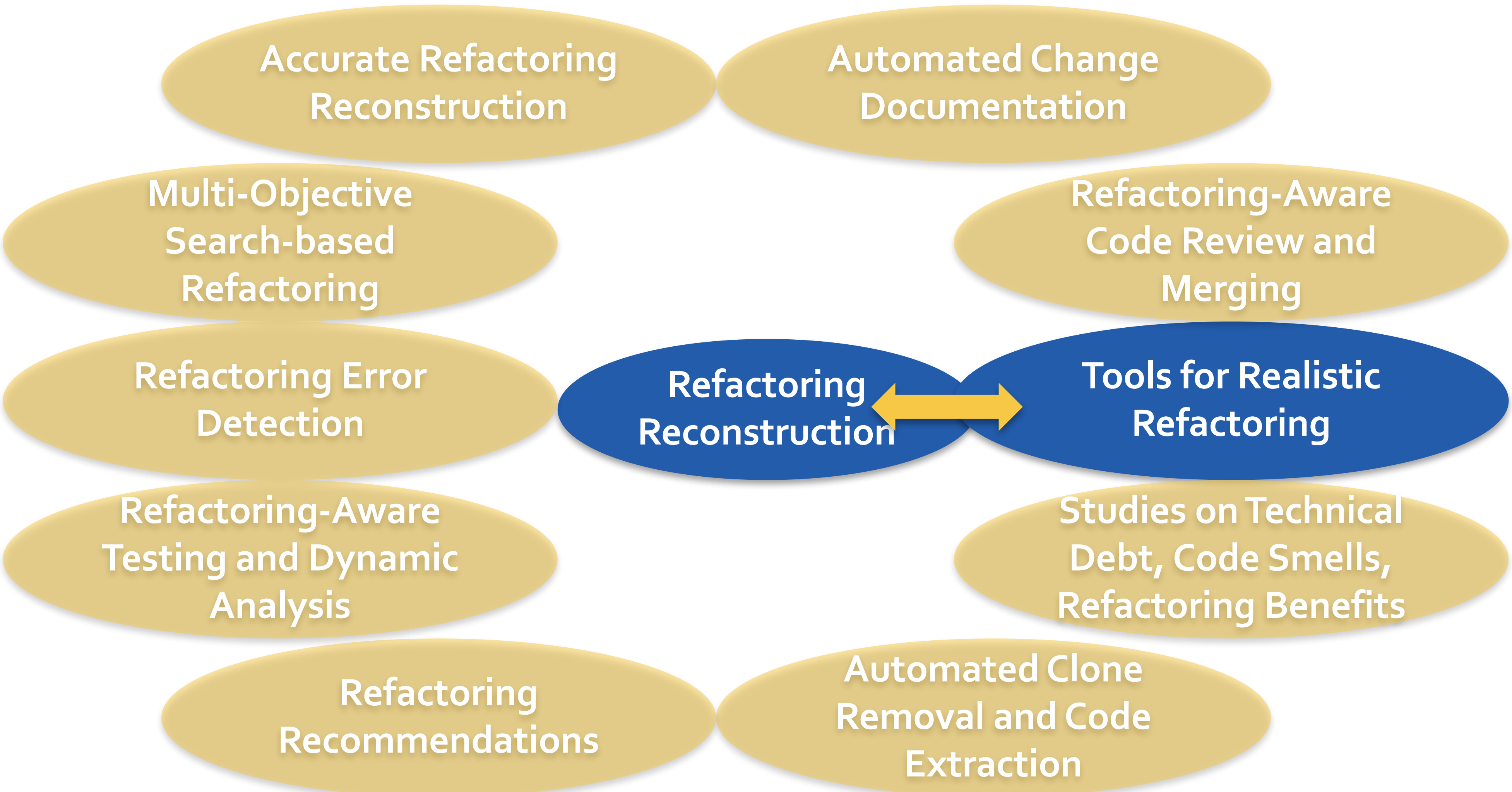
	Versions	# Found	Prec.	Recall
jEdit	*3.0-3.0.1*	*10*	*0.75*	*0.78*
	3.0.1-3.0.2	1	1	1
	3.0.2-3.1	214	0.45	1
Columba	300-352	43	0.52	0.9
	352-449	209	0.91	1
Carol	62-63	12	1	1
	389-421	8	0.63	1
	421-422	147	0.64	0.9
	429-430	48	0.85	1
	430-480	37	0.81	1
	480-481	11	0.91	0.9
	548-576	20	1	1
	576-764	14	0.85	1
Total		774	0.74	0.96

Reflections on the paper

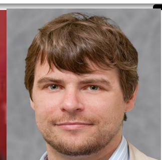
SE community took this work to several directions



Example: Tools for Realistic Refactoring



Friendly Competition Towards The Same Vision


How We Refactor, and How We Know It

Emerson Murphy-Hill
Portland State University
emerson@cs.pdx.edu

Chris Parnin
Georgia Institute of Technology
chris.parnin@gatech.edu

Andrew P. Black
Portland State University
black@cs.pdx.edu

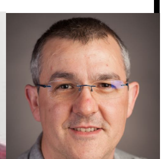


A Field Study of Refactoring Challenges and Benefits

Miryung Kim *
miryung@ece.utexas.edu

Thomas Zimmermann +
tzimmer@microsoft.com

Nachiappan Nagappan +
nachin@microsoft.com


Use, Disuse, and Misuse of Automated Refactorings

Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P. Bailey, Ralph E. Johnson
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{*mvakili2, nchen, snegara2, rajkuma1, bpbailey, rjohnson*}*@illinois.edu*

A Comparative Study of Manual and Automated Refactorings

Stas Negara, Nicholas Chen, Mohsen Vakilian,
Ralph E. Johnson, and Danny Dig

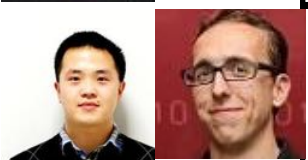
Friendly Competition Towards The Same Vision

WitchDoctor: IDE Support for Real-Time Auto-Completion of Refactorings

Stephen R. Foster
UC San Diego
La Jolla, CA
srfoster@cs.ucsd.edu

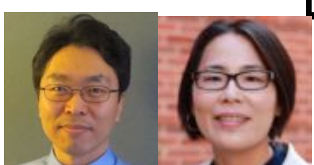
William G. Griswold
UC San Diego
La Jolla, CA
wgg@cs.ucsd.edu

Sorin Lerner
UC San Diego
La Jolla, CA
lerner@cs.ucsd.edu



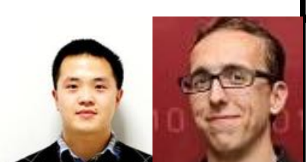
Reconciling Manual and Automatic Refactoring

Xi Ge Quinton L. DuBose Emerson Murphy-Hill
Department of Computer Science, North Carolina State University, Raleigh, NC
{xge, qldubose}@ncsu.edu, emerson@csc.ncsu.edu



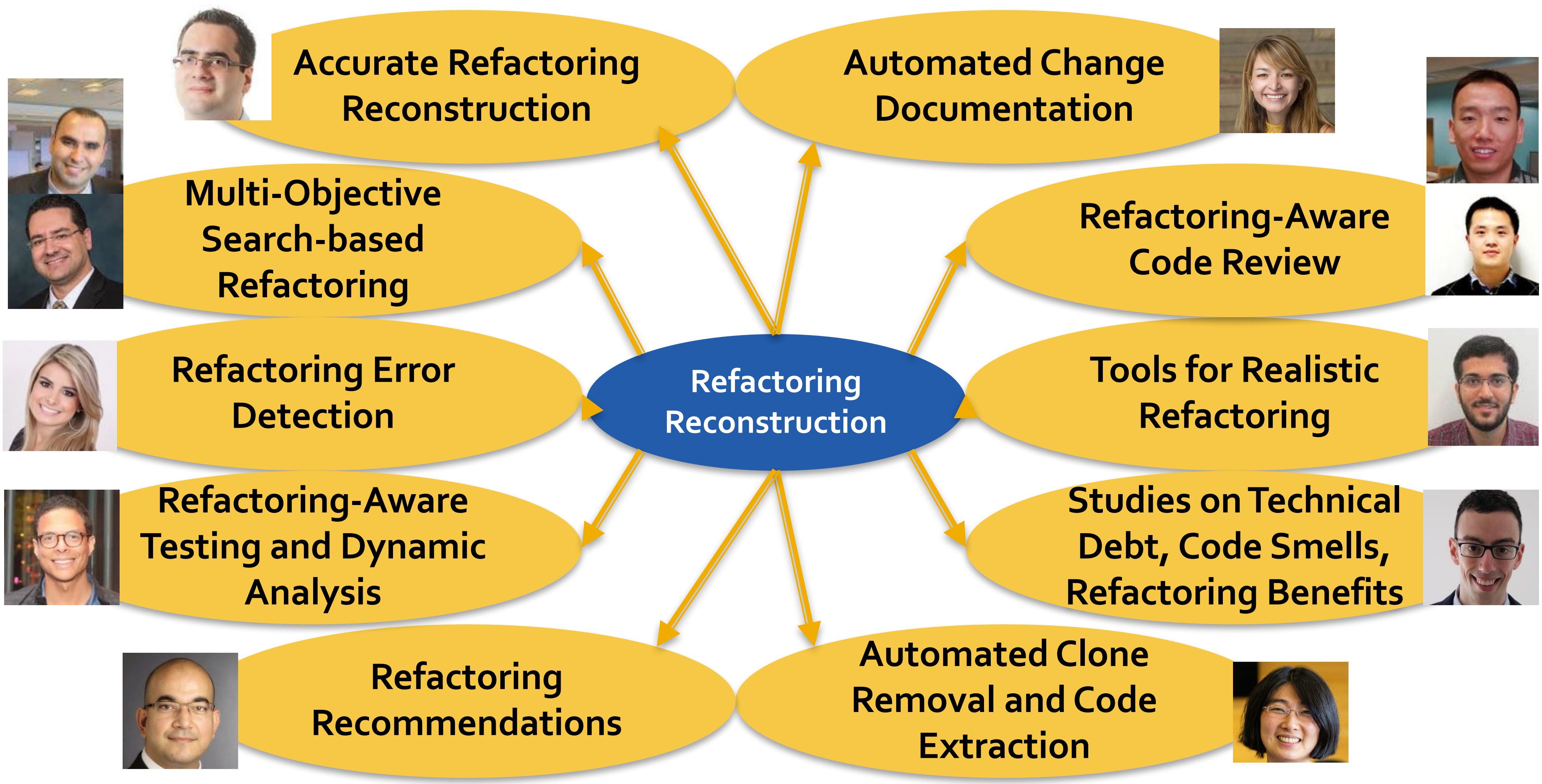
RefDistiller: A Refactoring Aware Code Review Tool for Inspecting Manual Refactoring Edits

Everton L. G. Alves[†‡] Myoungkyu Song[†] Miryung Kim[§]
University of Texas at Austin, USA[†] University of California, Los Angeles, USA[§]
Federal University of Campina Grande, Brazil[‡]
{everton, mksong1117}@utexas.edu, miryung@cs.ucla.edu



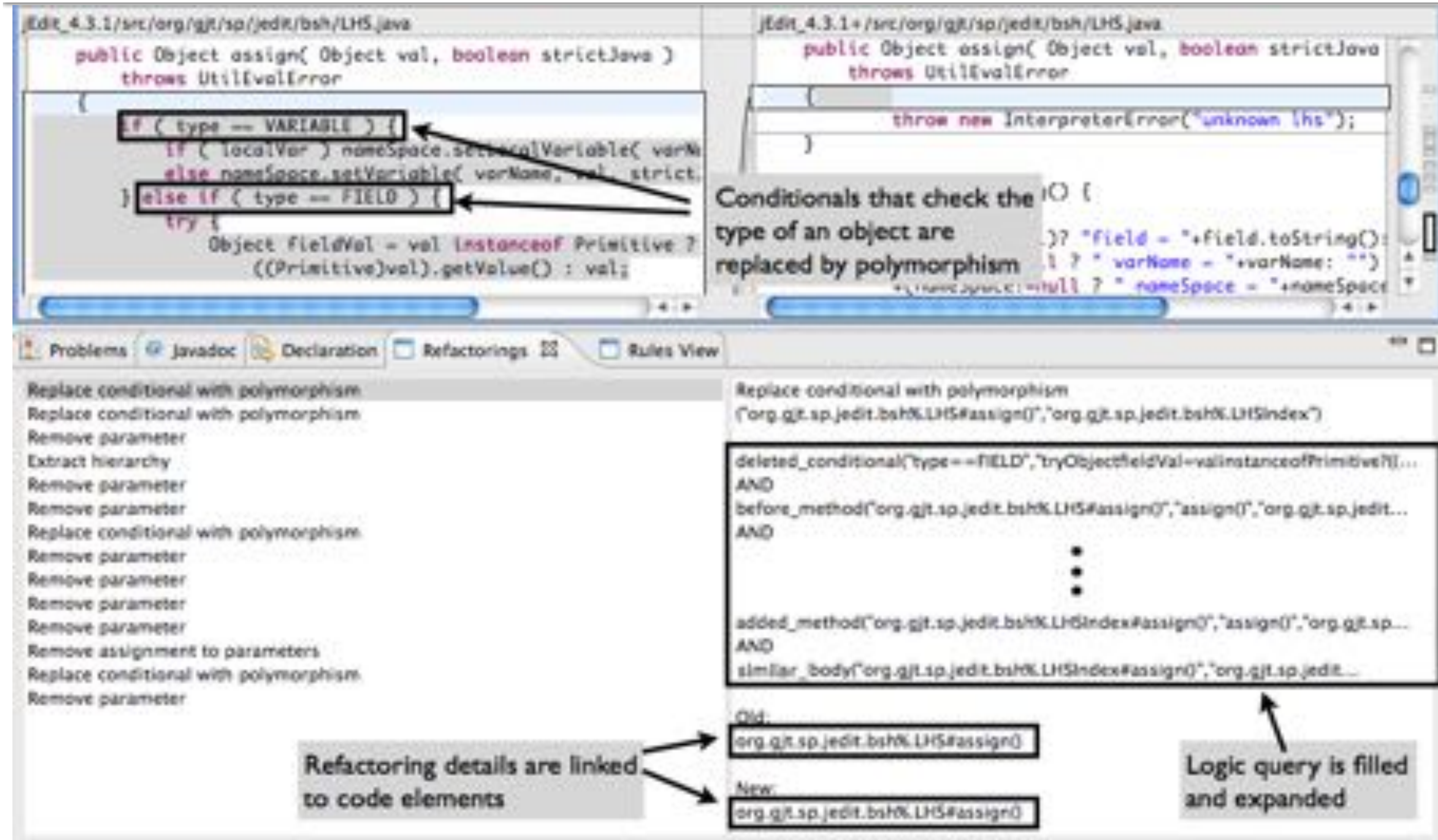
Manual Refactoring Changes with Automated Refactoring Validation

SE community took this work to several directions



RefFinder Tool Release

[ICSM'10, Prete et al. FSE'10 Demo, Kim et al.]



Microsoft SEIF Award

Microsoft

RefFinder: An Extensible Framework for Refactoring Reconstruction

Professor Miryung Kim
The University of Texas at Austin

UT ECE Prof. Miryung Kim Receives 2011 Microsoft Software Engineering Innovation Foundation Award

MSR Visit & Collaboration

Studies on Refactoring Challenges & Benefits

Re-architecting Windows

Refactoring Change Impact Analysis

[Napol's Undergraduate Honors Thesis / ICSM'12 Rachatasumrit and Kim]

- We integrate RefFinder with FaultTracer dynamic change impact analysis [ICSM'12]
- While refactoring edits are only 8% of changes, 38% of affected tests are relevant to refactoring and a half of failed affected tests include refactoring edits.

Thankful to My Students

From Right to Left
Baishakhi Ray (PhD 2013 ⇒Assistant Prof @ Columbia) *Detecting Recurring Changes and Errors*
Na Meng (PhD 2014 ⇒ Assistant Prof @ Virginia Tech) *Automating Recurring Changes & Clone Removal*
Tianyi Zhang (PhD 2019, Postdoc @ Harvard) *Leveraging Redundancy for Code Review, Testing, API Usage Mining*
Muhammad Ali Gulzar (PhD 2020 ⇒ Assistant Prof @ Virginia Tech) *Debugging and Testing for Big Data Analytics*
Myoungkyu Song (Postdoc 2015 ⇒ Assistant Prof @ Nebraska, Omaha) *Error Detection in Refactoring Edits*

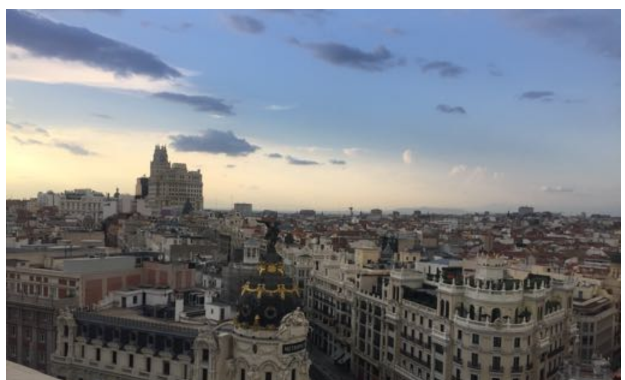
Thankful to ICSME "*Community*"

ICSM 2009 Edmonton
My first PC

ICSM 2011 Williamsburg
My first OC/ ERA co-chair

ICSME 2013 Einhoven

ICSM 2012 Riva del Garda



ICSME 2018 Madrid

ICSME 2019 my first
PC co-chair / my first SC

36th IEEE International Conference on Software Maintenance and Evolution (ICSME 2020) Adelaide, Australia

Most Influential Paper from ICSM 2010

"Template-based Reconstruction of Complex Refactoring"

by Kyle Prete, Napol Rachatasumrit, Nikita Sudan, and Miryung Kim

UCLA