Applied Static Analysis 2016

Recommendation Systems in Software Engineering

Sebastian Proksch

Dr. Michael Eichberg (Organizer), Johannes Lerch,

Ben Hermann, Karim Ali Ph. D.

Blick ins Buch

Alle 3 Bilder anzeigen

Design Patterns. Elements of Reusable Object-Oriented Software. (Englisch)

Gebundene Ausgabe – 31. Oktober 1994

von Erich Gamma (Autor), Richard Helm (Autor), & 2 mehr

★★★★½ 94 Kundenrezensionen

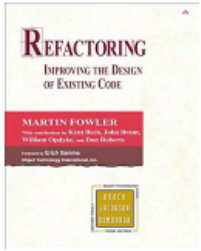
Lieferung Freitag, 1. Juli: Bestellen Sie innerhalb 20 Stunden und 43 Minuten per **Premiumversand** an der Kasse. Siehe Details.

72 neu ab EUR 25,53 | 12 gebraucht ab EUR 15,54

These texts cover the design of object-oriented software and examine how to investigate requirements, create solutions and then translate designs into code, showing developers how to make practical use of the most significant recent developments. A summary of UML notation is included.

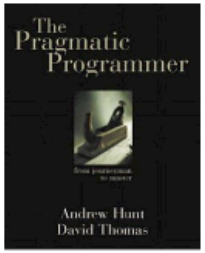
Kunden, die diesen Artikel gekauft haben, kauften auch

Seite 1 von 20

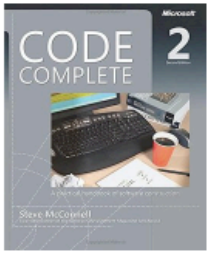


Refactoring: Improving the Design of Existing Code (Object Technology Series)
› Martin Fowler
★★★★½ 58
Gebundene Ausgabe
EUR 47,95 Prime

Clean Code: A Handbook of Agile Software Craftsmanship (Robert…
› Robert C. Martin
★★★★½ 59
Taschenbuch
EUR 35,95 Prime

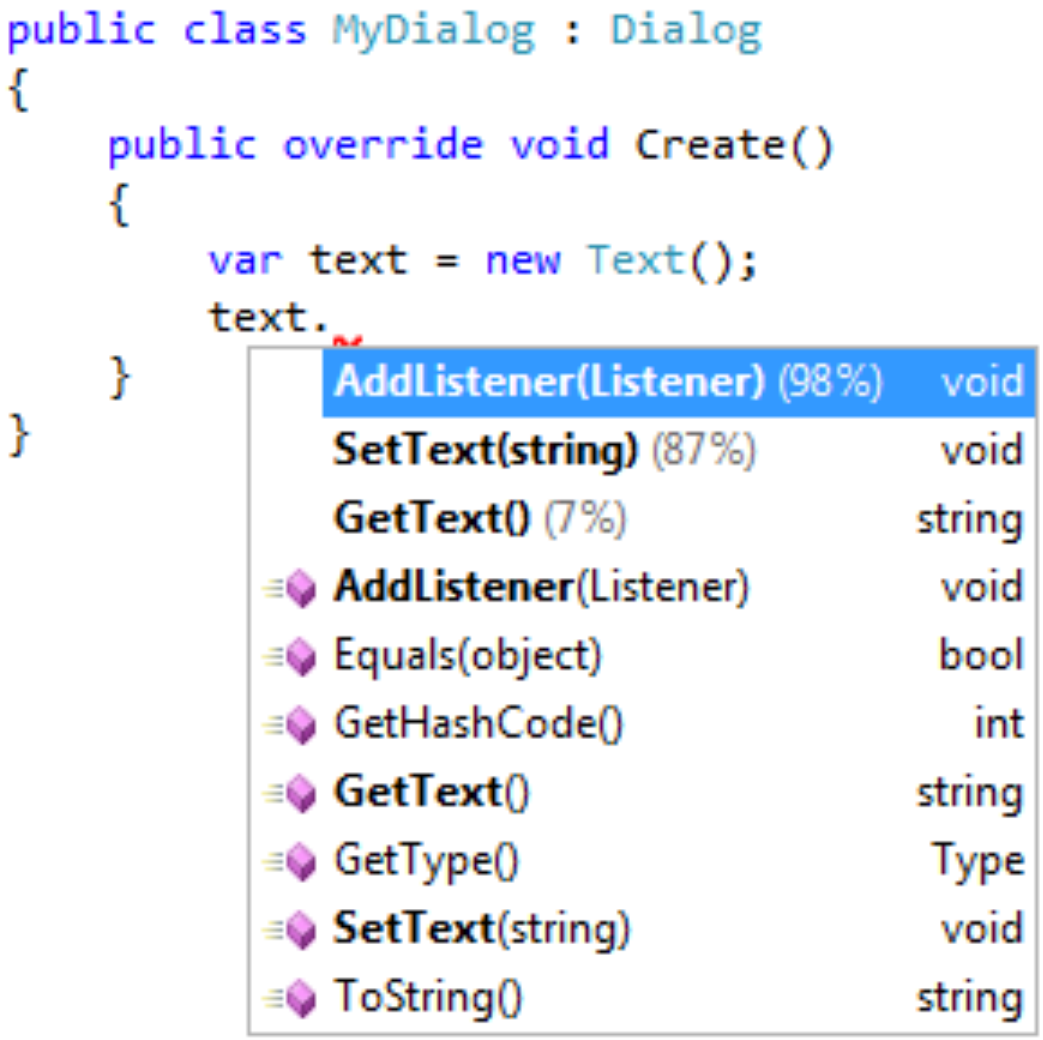


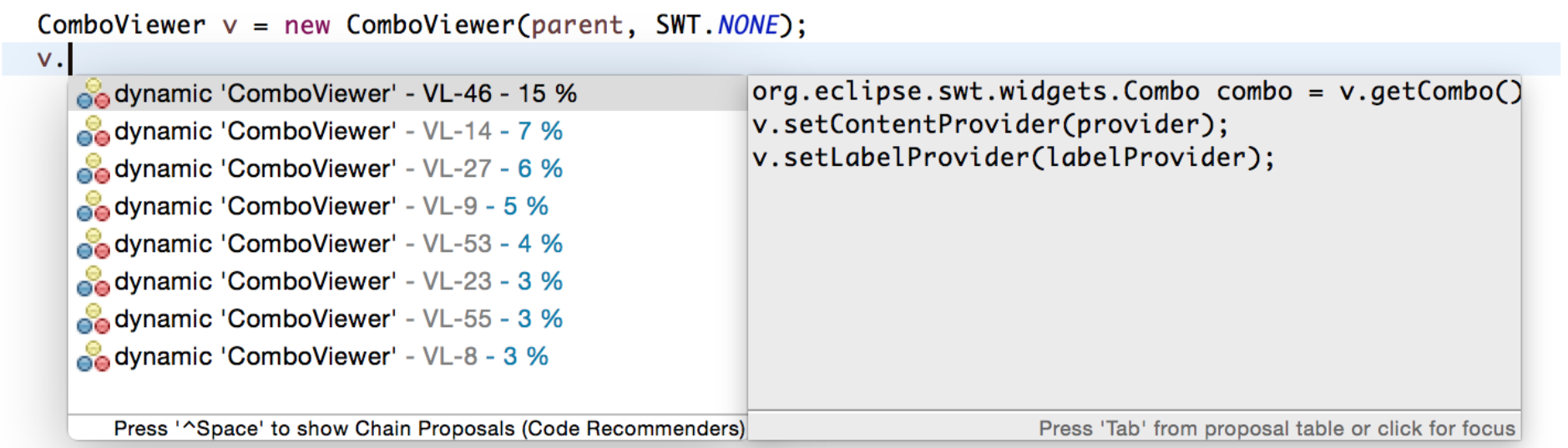
The Pragmatic Programmer. From Journeyman to Master
› Andrew Hunt
★★★★½ 36
Taschenbuch
EUR 35,95 Prime

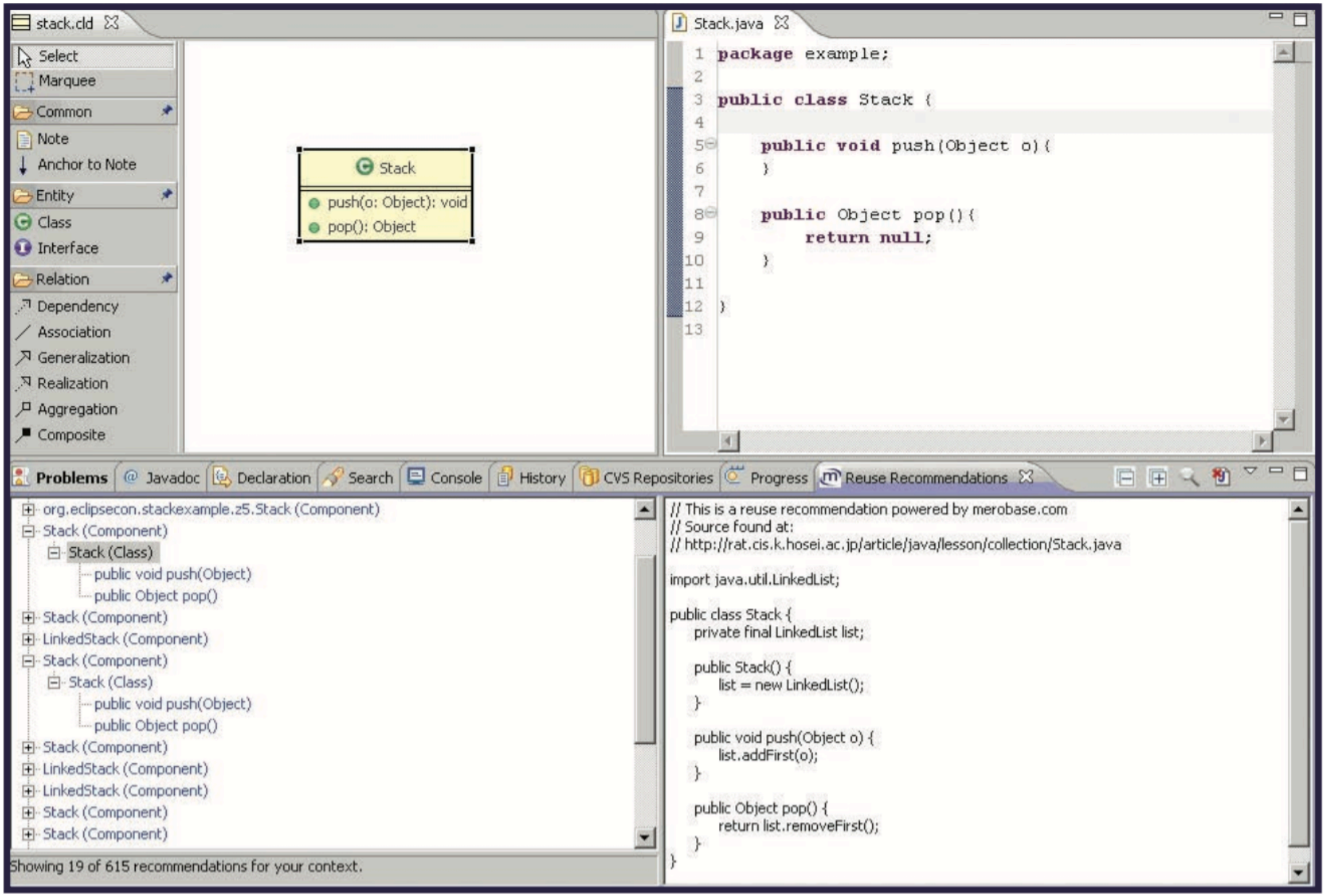


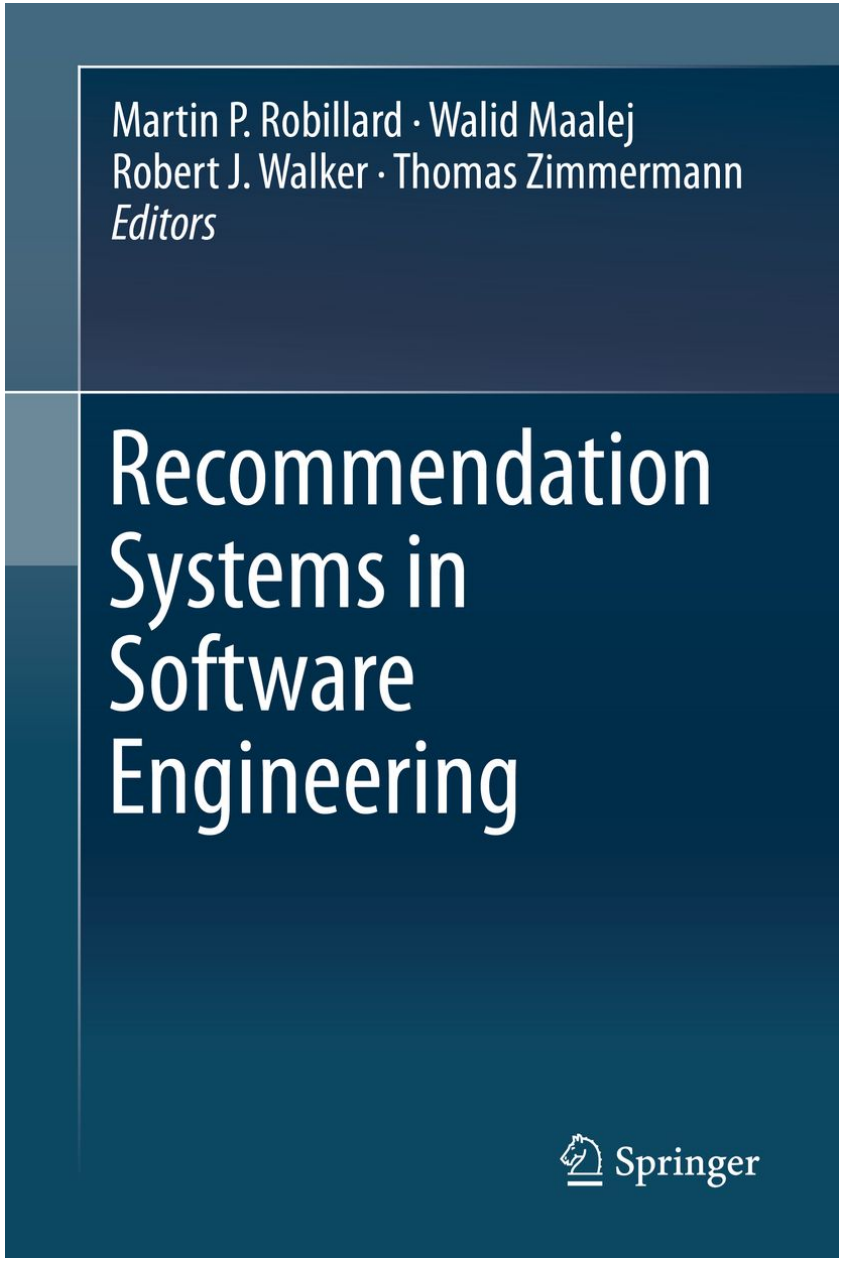
Code Complete: A Practical Handbook of Software Construction: A Practical…
› Steve McConnell
★★★★☆ 10
Taschenbuch
EUR 27,21 Prime

Head First Design Patterns
› Eric Freeman
★★★★★ 50
Taschenbuch
EUR 43,95 Prime


public class MyDialog : Dialog
{
public override void Create()
{
var text = new Text();
text.
}
}
AddListener(Listener) (98%) void
SetText(string) (87%) void
GetText() (7%) string
AddListener(Listener) void
Equals(object) bool
GetHashCode() int
GetText() string
GetType() Type
SetText(string) void
ToString() string


ComboViewer v = new ComboViewer(parent, SWT.NONE);
v.
dynamic 'ComboViewer' - VL-46 - 15 %
dynamic 'ComboViewer' - VL-14 - 7 %
dynamic 'ComboViewer' - VL-27 - 6 %
dynamic 'ComboViewer' - VL-9 - 5 %
dynamic 'ComboViewer' - VL-53 - 4 %
dynamic 'ComboViewer' - VL-23 - 3 %
dynamic 'ComboViewer' - VL-55 - 3 %
dynamic 'ComboViewer' - VL-8 - 3 %
org.eclipse.swt.widgets.Combo combo = v.getCombo()
v.setContentProvider(provider);
v.setLabelProvider(labelProvider);
Press '^Space' to show Chain Proposals (Code Recommenders)
Press 'Tab' from proposal table or click for focus


stack.cld
Select
Marquee
Common
Note
Anchor to Note
Entity
Class
Interface
Relation
Dependency
Association
Generalization
Realization
Aggregation
Composite
Stack
push(o: Object): void
pop(): Object
Stack.java
1 package example;
2
3 public class Stack {
4
5 public void push(Object o){
6 }
7
8 public Object pop(){
9 return null;
10 }
11
12 }
13
Problems
Javadoc
Declaration
Search
Console
History
CVS Repositories
Progress
Reuse Recommendations
org.eclipsecon.stackexample.z5.Stack (Component)
Stack (Component)
Stack (Class)
public void push(Object)
public Object pop()
Stack (Component)
LinkedStack (Component)
Stack (Component)
Stack (Class)
public void push(Object)
public Object pop()
Stack (Component)
LinkedStack (Component)
LinkedStack (Component)
Stack (Component)
Stack (Component)
Showing 19 of 615 recommendations for your context.
// This is a reuse recommendation powered by merobase.com
// Source found at:
// http://rat.cis.k.hosei.ac.jp/article/java/lesson/collection/Stack.java
import java.util.LinkedList;
public class Stack {
private final LinkedList list;
public Stack() {
list = new LinkedList();
}
public void push(Object o) {
list.addFirst(o);
}
public Object pop() {
return list.removeFirst();
}
}


Martin P. Robillard · Walid Maalej
Robert J. Walker · Thomas Zimmermann
Editors
Recommendation Systems in Software Engineering
Springer

How to Build a Recommendation System for Software Engineering

Sebastian Proksch[1], Veronika Bauer[2], and Gail C. Murphy[3]

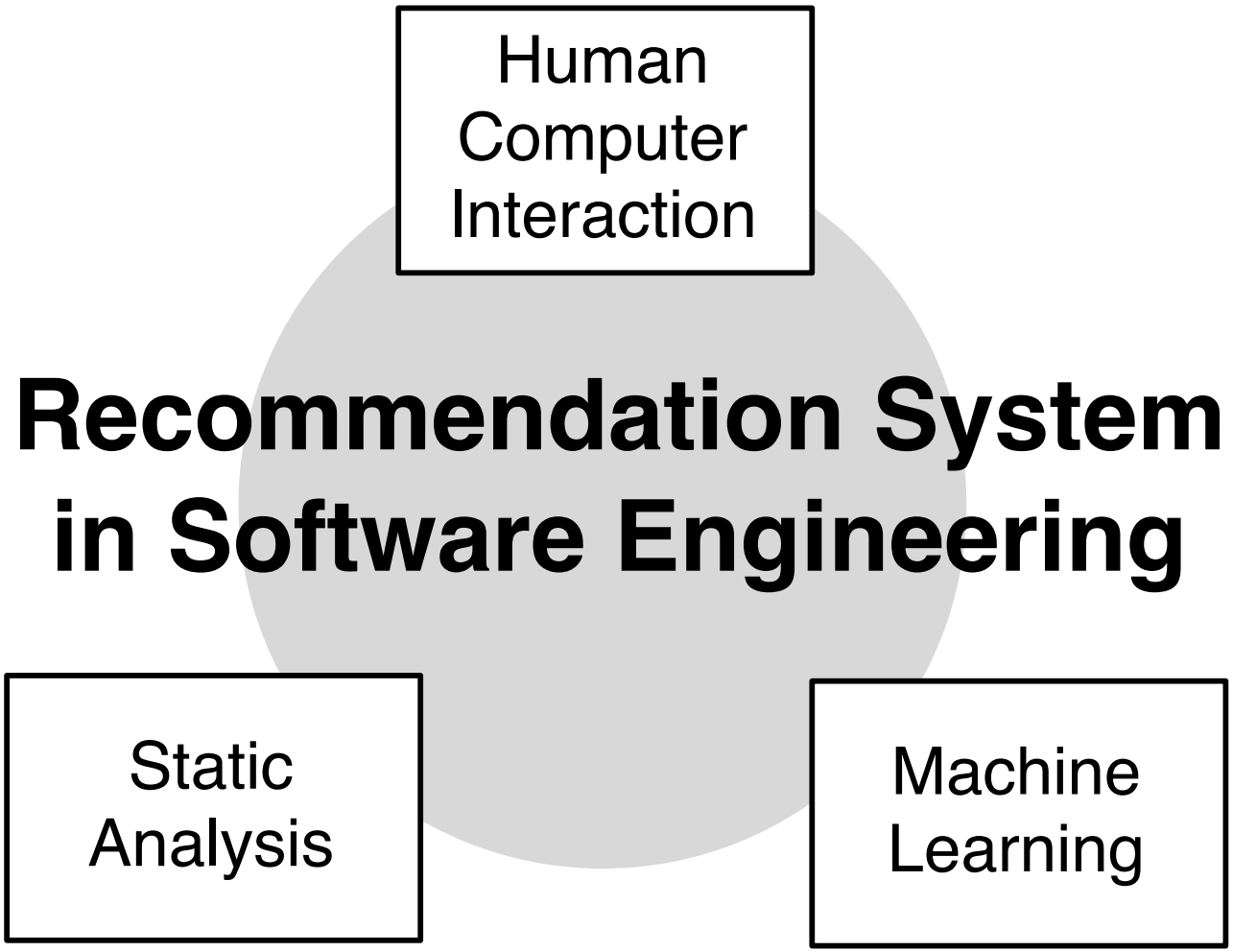
[1] TU Darmstadt, proksch@cs.tu-darmstadt.de
[2] TU München, bauerv@in.tum.de
[3] UBC, Vancouver, murphy@cs.ubc.ca

Abstract. Software developers must interact with large amounts of different types of information and perform many different activities to build a software system. To ease the finding of information and hone workflows, there has been growing interest in building recommenders that are intended to help software developers work more effectively. Building an effective recommender requires a deep understanding of the problem that is the target of a recommender, analysis of different aspects of the approach taken to perform the recommendations and design and evaluation of the mechanisms used to present recommendations to a developer. In this chapter, we outline the different steps that must be taken to develop an effective recommender system to aid software development.

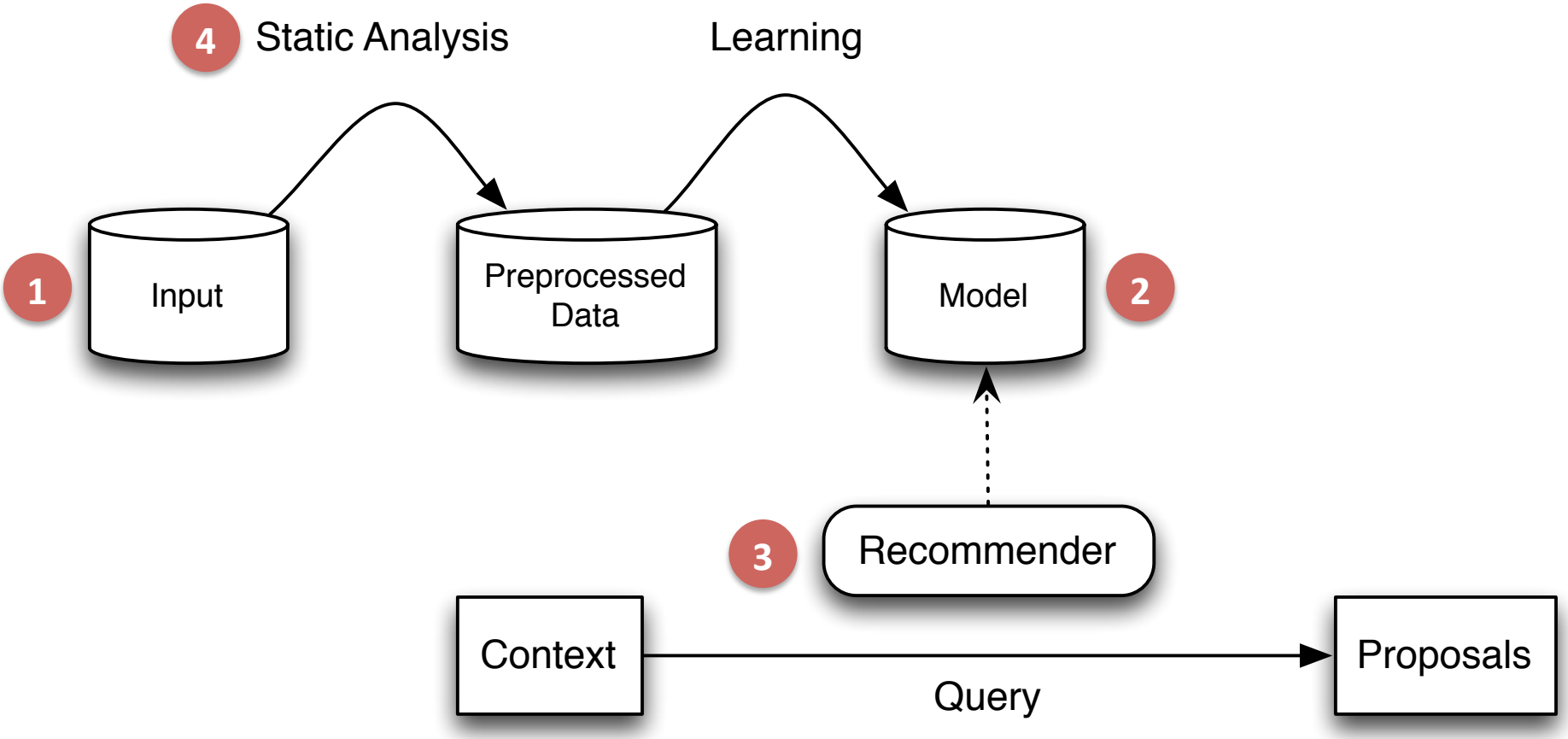
1 Introduction

Software developers perform many different activities when building a software system: writing code, testing code, deploying to the cloud, coordinating via email and meetings, and many more [70]. Each of these activities requires finding and interacting with different kinds of information, using different tools and determining and preparing for the next activity to perform. For example, as part

WHAT DO RSSE HAVE TO DO WITH STATIC ANALYSIS?


Human
Computer
Interaction
Recommendation System
in Software Engineering
Static
Analysis
Machine
Learning

Typical RSSE Pipeline



INPUT

Mining Software Repositories

- Lot's of open source data available
- (Usually) stable source code
- (Usually) Compilable
- Analysis frameworks available, once compiled
- Three-address representation easy to analyze

Source Code Under Development

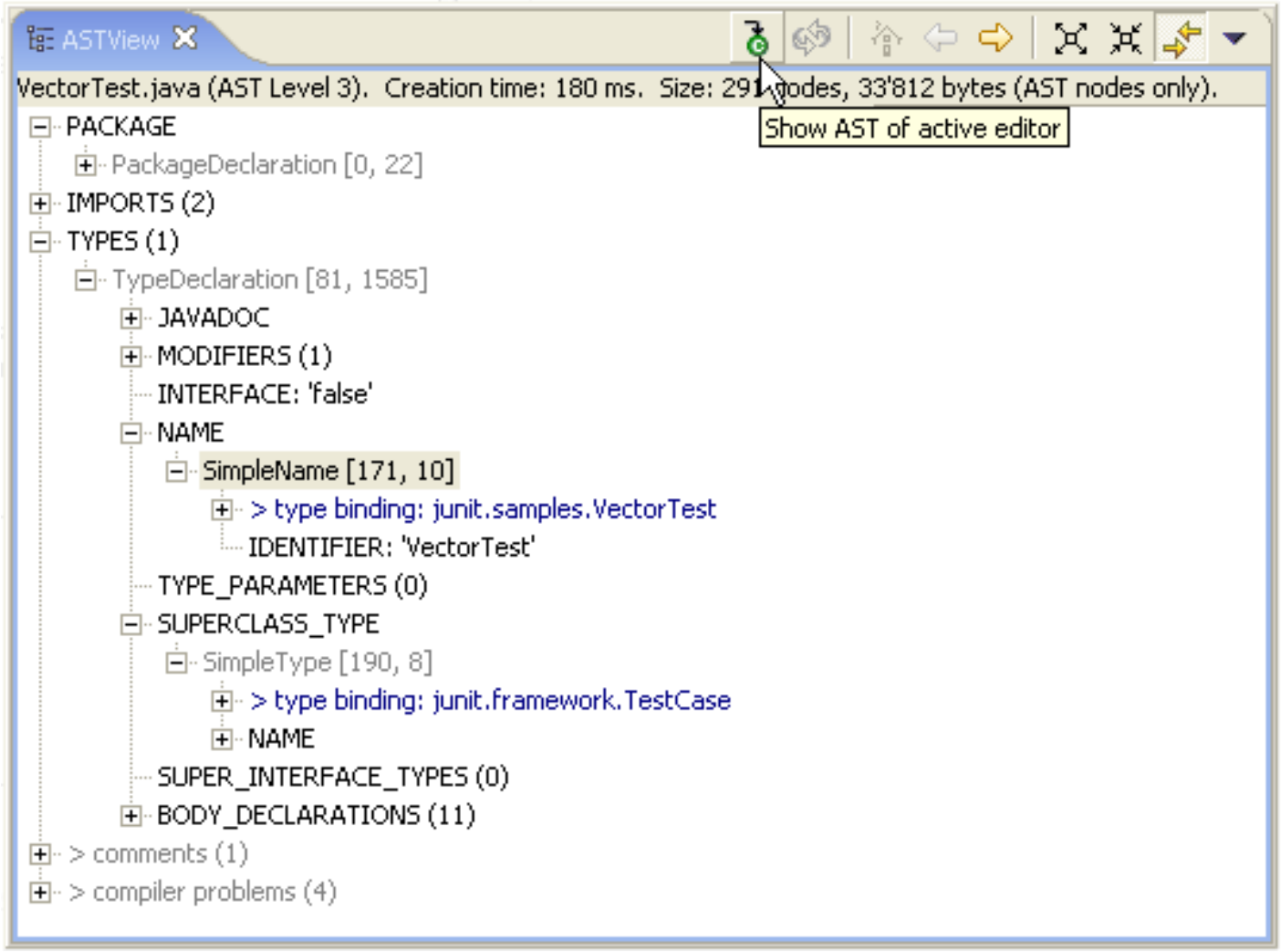
- Problem in practice
- Incomplete code
- Invalid Statements
- Usually all types are resolved, environment is "functional"
- Often, analysis based on AST

Source Code Nested in Plaintext

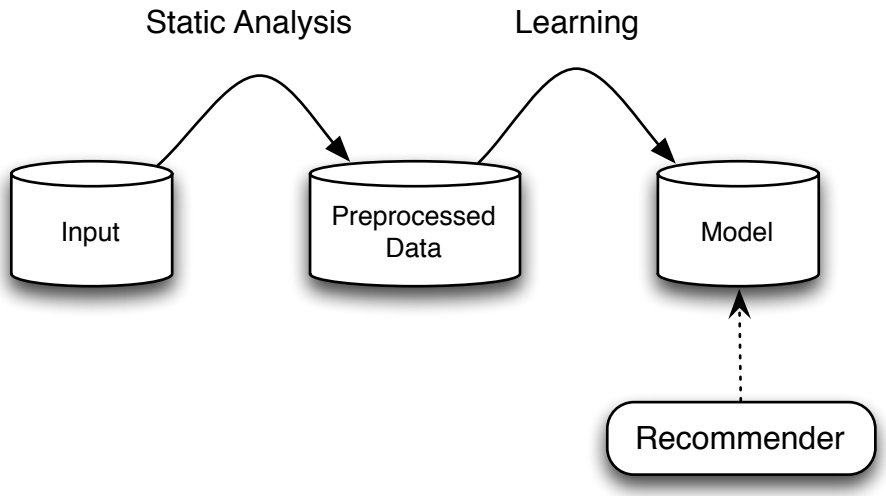
(Stackoverflow, Mailinglists, Tutorials, etc.)

- Types are not resolved
- Incomplete code (e.g., missing imports)
- Invalid Code (e.g., pseudo-code parts)
- Language not always obvious
- …

Once everything is set up…



It is often the first challenge to make the source code analyzable.



(Machine) Learning

MODELS (?)

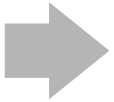
Models represent the knowledge gained in the learning process. The model is used to infer proposals, its contents depend on the recommendation goal. The static analysis depends on the model.

Use Case: Best Matching Neighbor Algorithm

MODELS

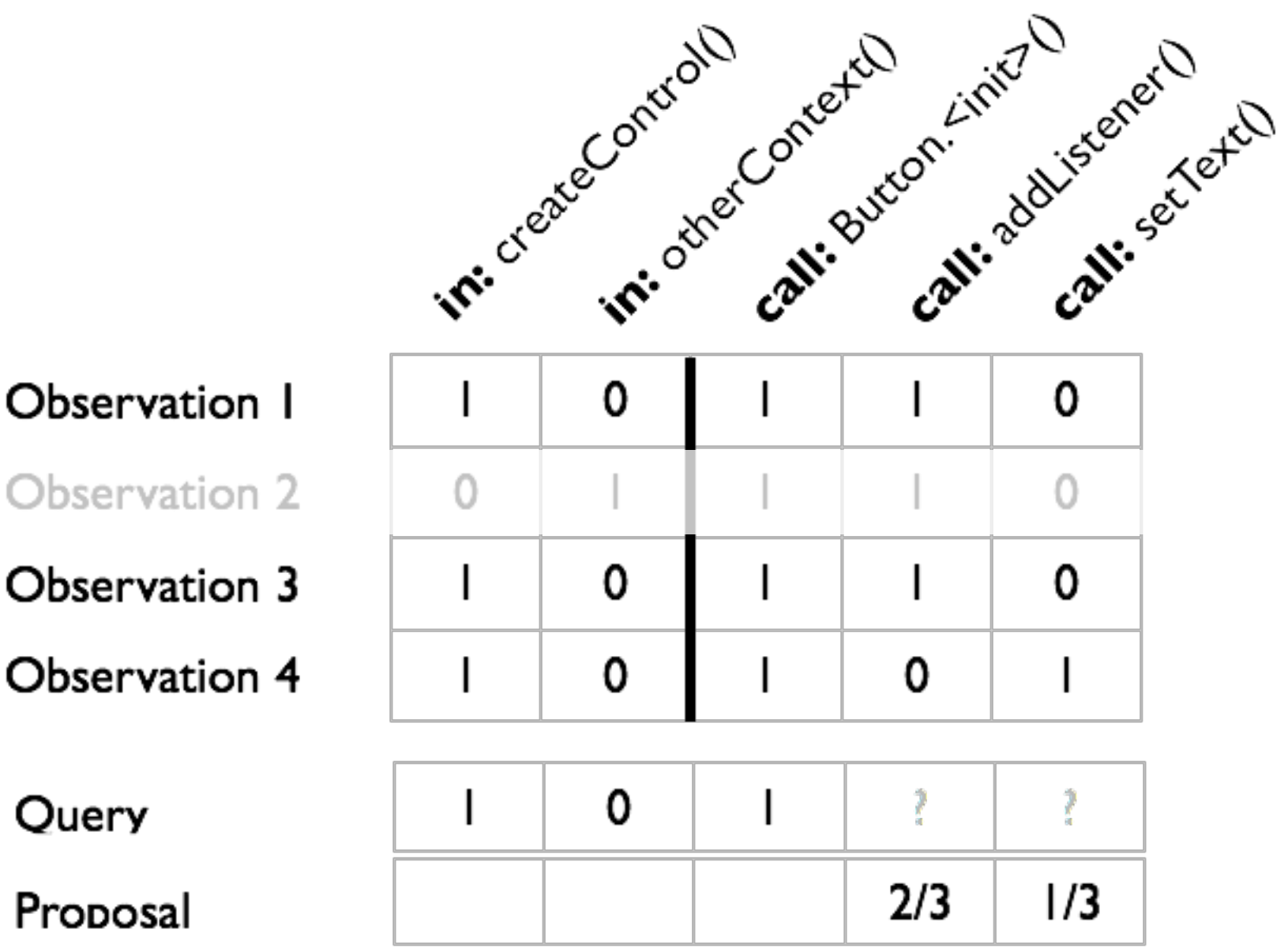
Capture Structural Context

```
public class MyWizardPage extends WizardPage {

    @Override
    public void createControl(Composite arg0) {

        Button b = new Button(arg0, INFORMATION);
        b.addSelectionListener(Helper.createSelectionListener())
        b.setLayoutData(Helper.createLayout());

        b.
    }
```



ctx: WizardPage.createControl(Composite)

call: Button.<init>(...)
call: addSelectionListener(...)
call: setLayoutData(...)


in: createControl()
in: otherContext()
call: Button.<init>()
call: addListener()
call: setText()
Observation 1
1 0 1 1 0
Observation 2
0 1 1 1 0
Observation 3
1 0 1 1 0
Observation 4
1 0 1 0 1
Query
1 0 1 ? ?
Proposal
2/3 1/3

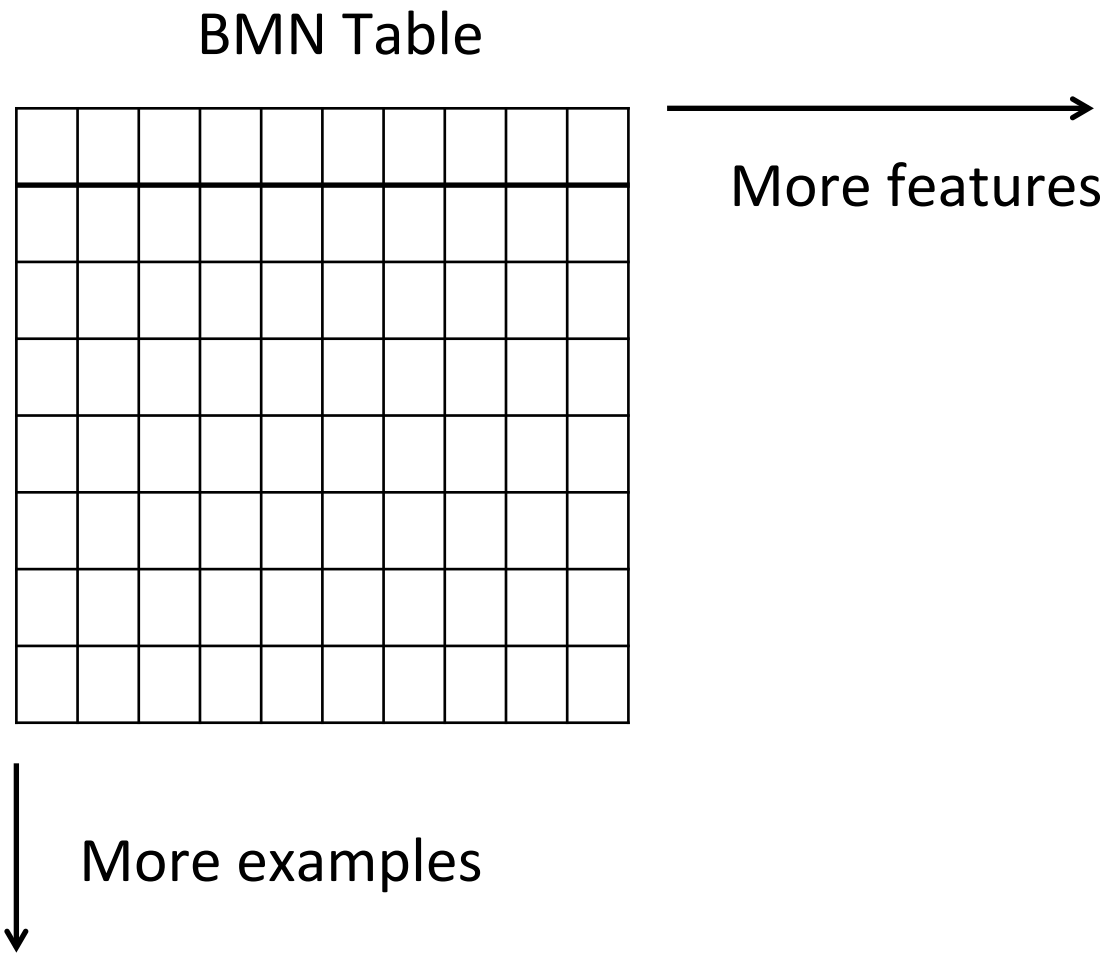
Use Case: Pattern-based Bayesian Network

MODELS

Make it Better, Use More Features

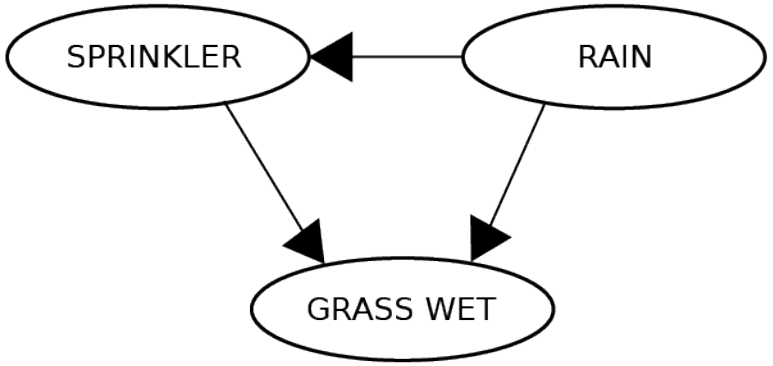
- Already in Use
  - Type
  - Enclosing Method
  - Receiver Callsites
- New Features
  - Enclosing Class
  - Parameter Callsites
  - Definition

Existing Solution Does Not Scale



Bayesian Networks to the Rescue!

RAIN	SPRINKLER: T	SPRINKLER: F
F	0.4	0.6
T	0.01	0.99

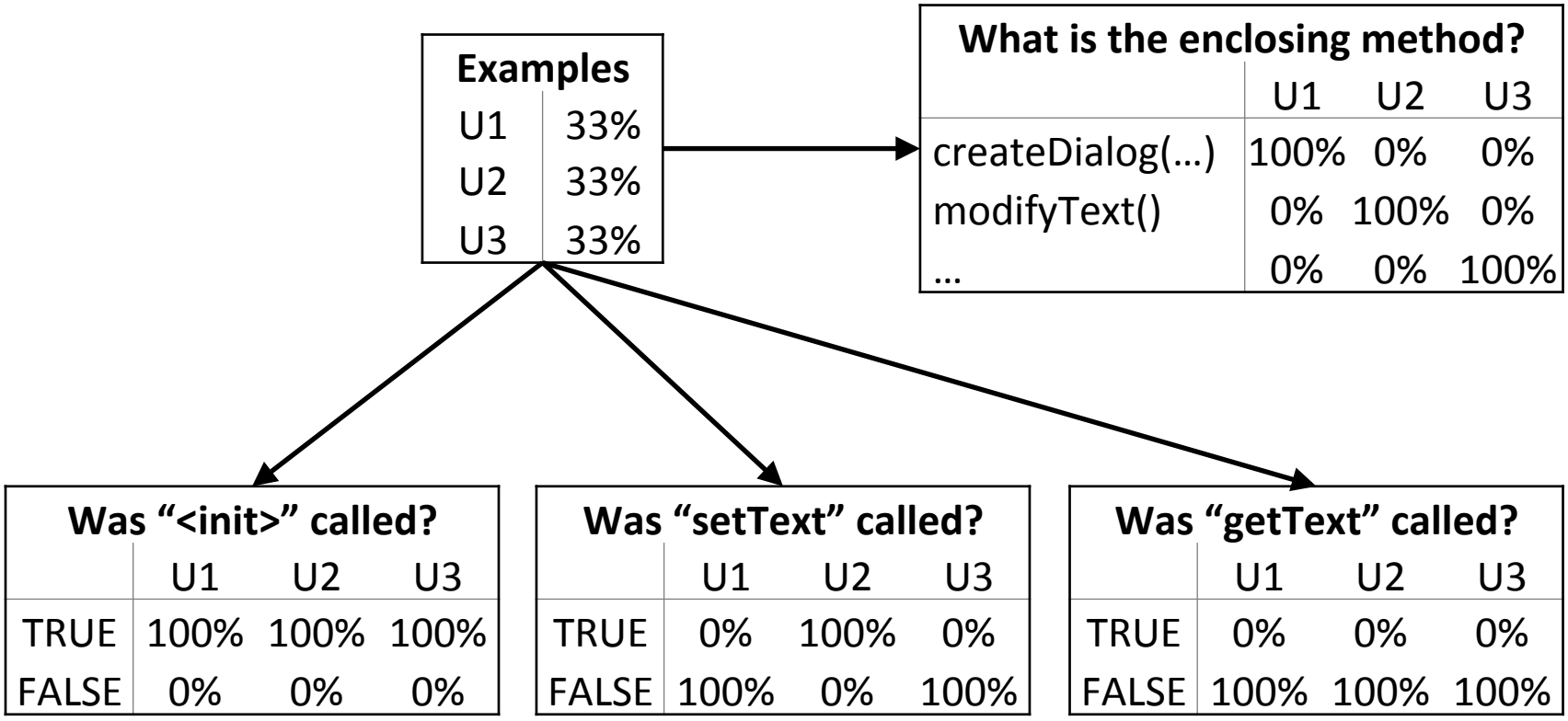


RAIN: T	RAIN: F
0.2	0.8

SPRINKLER	RAIN	GRASS WET: T	GRASS WET: F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Example query: P(RAIN | GRASS WET)?

Pattern-based Bayesian Network (PBN)



Examples	
U1	33%
U2	33%
U3	33%

What is the enclosing method?	U1	U2	U3
createDialog(…)	100%	0%	0%
modifyText()	0%	100%	0%
…	0%	0%	100%

Was “<init>” called?	U1	U2	U3
TRUE	100%	100%	100%
FALSE	0%	0%	0%

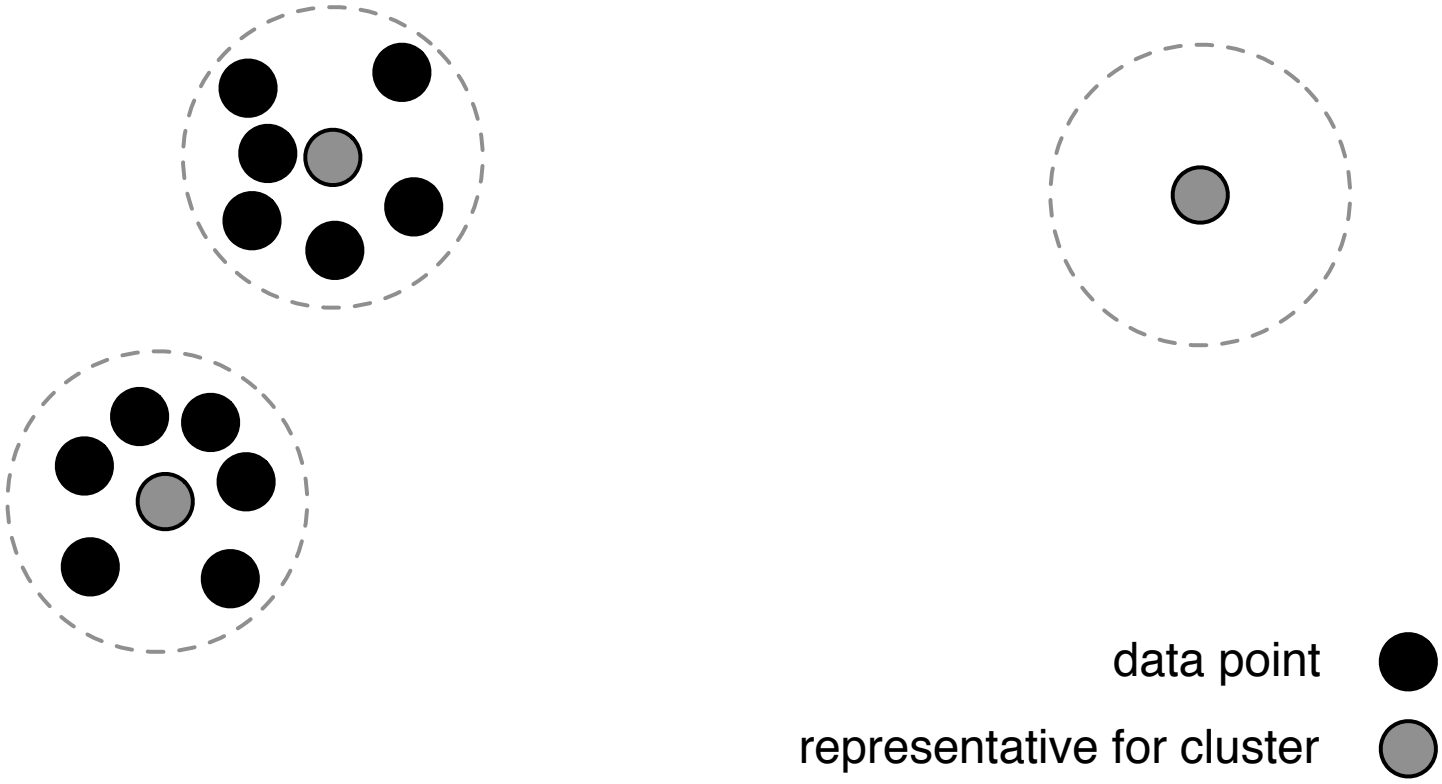
Was “setText” called?	U1	U2	U3
TRUE	0%	100%	0%
FALSE	100%	0%	100%

Was “getText” called?	U1	U2	U3
TRUE	0%	0%	0%
FALSE	100%	100%	100%

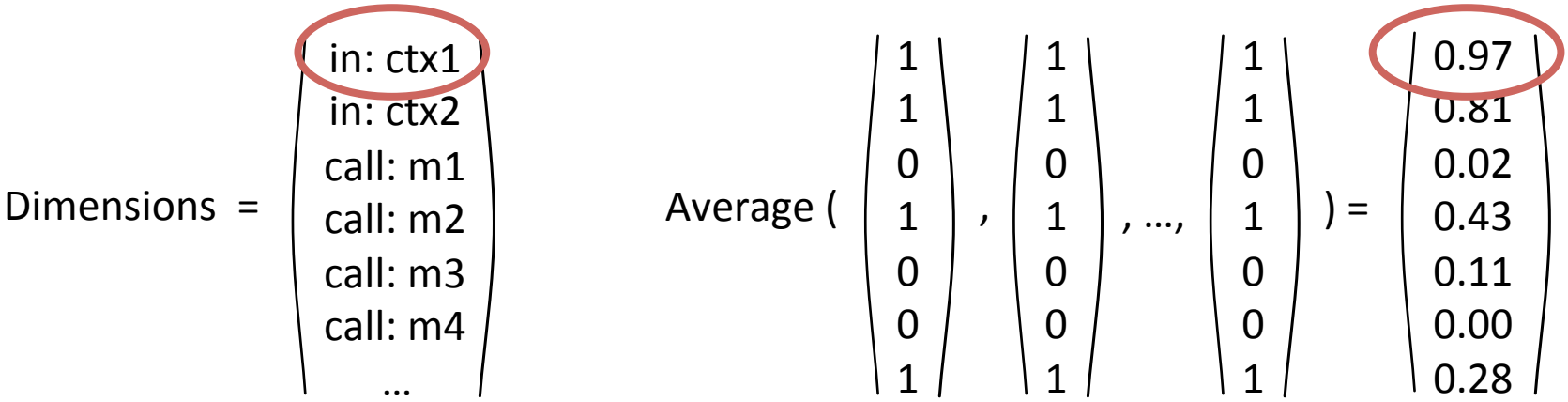
Examples: P(«method» == TRUE)?

P(«method» == TRUE | Context = “createDialog”)?

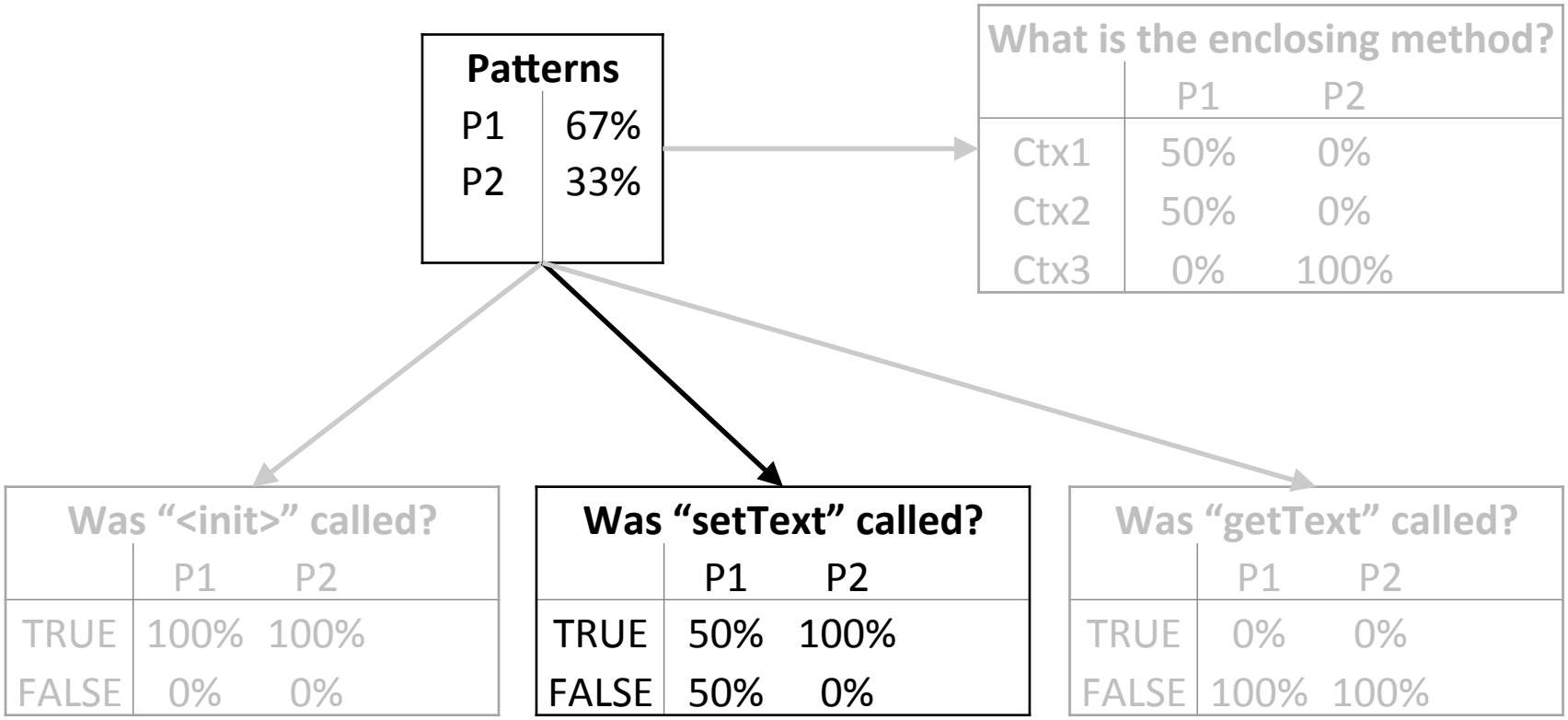
Clustering Data Points



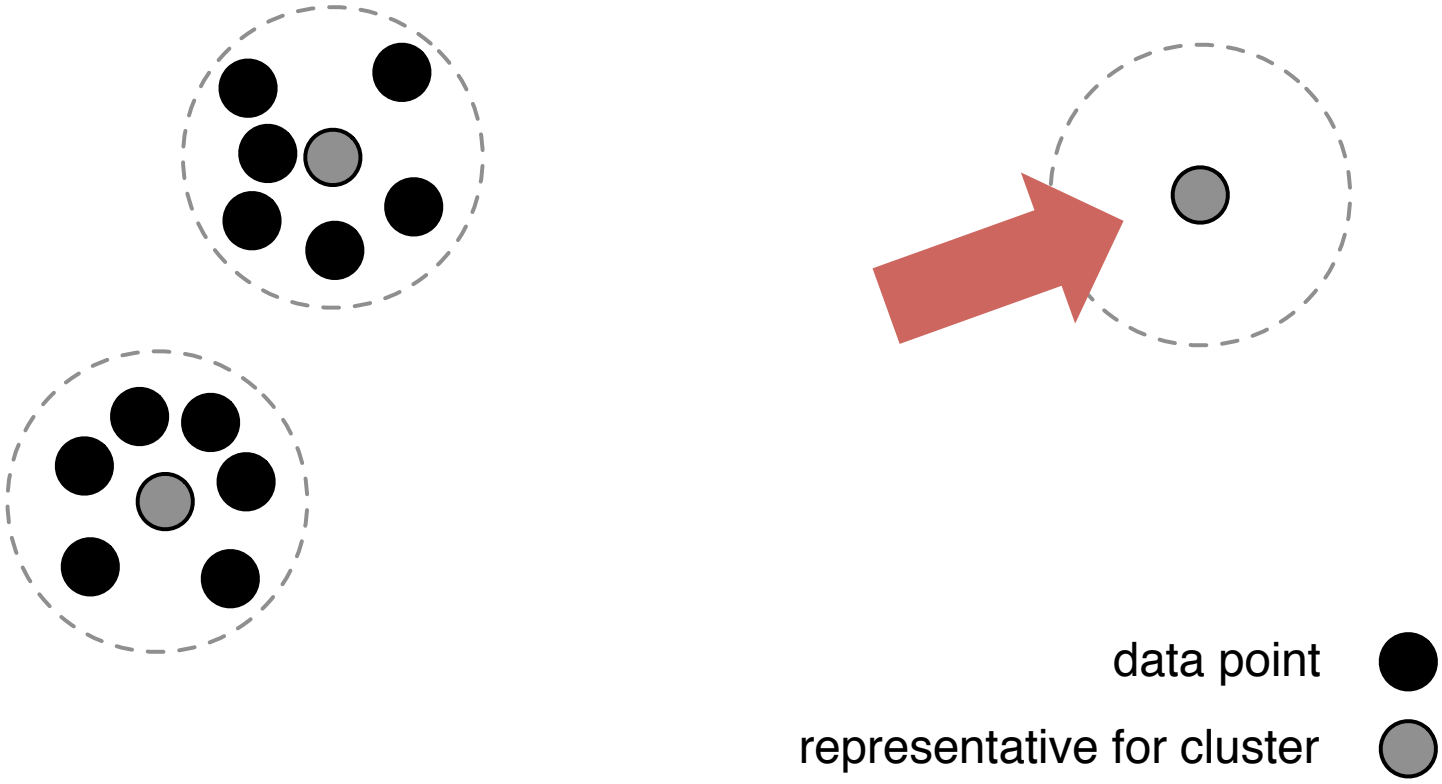
Finding the Cluster Representative



Probabilities Allow Clustering



Detect and Remove Outliers



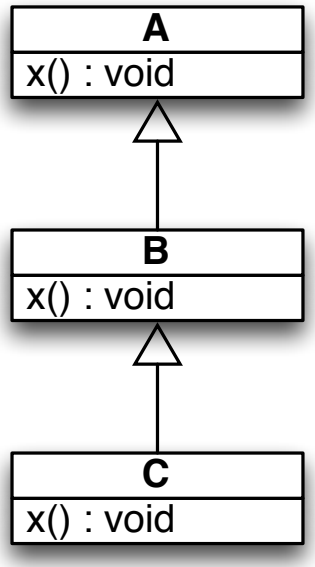
In general, machine learning tries to abstract from examples,
but, very often, the learning boils down to statistics.
RSSE is interested in finding the “common case”.

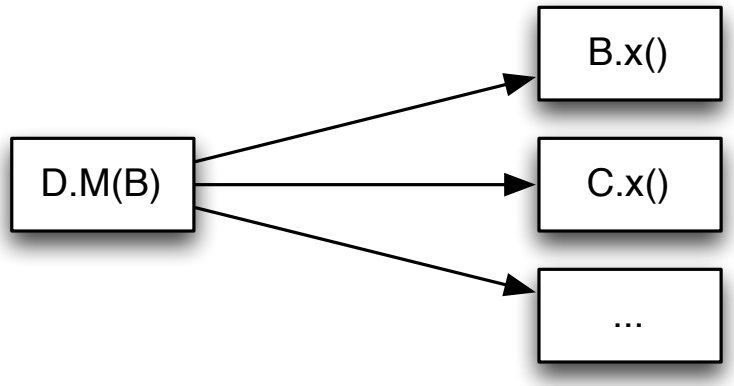
How to describe the context?

STATIC ANALYSES

Call-graph Construction

```
public class D {
        public void M(B b) {
                b.x()
        }
}
```





Excursion: Liskov Substitution Principle

Subtype Requirement:
Let f(x) be a property provable about objects x of type T. Then f(y) should be true for objects y of type S where S is a subtype of T.

LSP Violation?

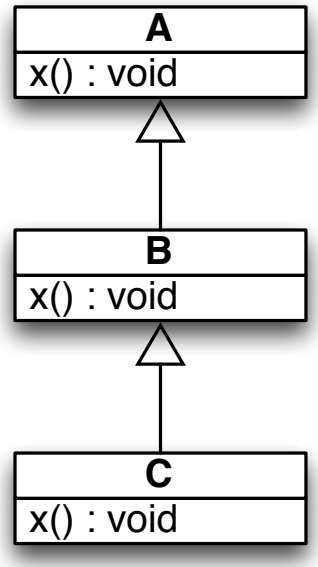
```
public class Rectangle {
        public int getX() { … }
        public void setX(int x) { … }
        public int getY() { … }
        public void setY(int y) { … }

        public int area() {
                return getX() * getY();
        }
}

public class Square extends Rectangle {
        public int getX() { return getY(); }
        public void setX(int x) { setY(x) }
}
```

```
public void testArea(Rectangle r) {
        r.setX(2);
        r.setY(3);
        assert(6, r.area());
}
```

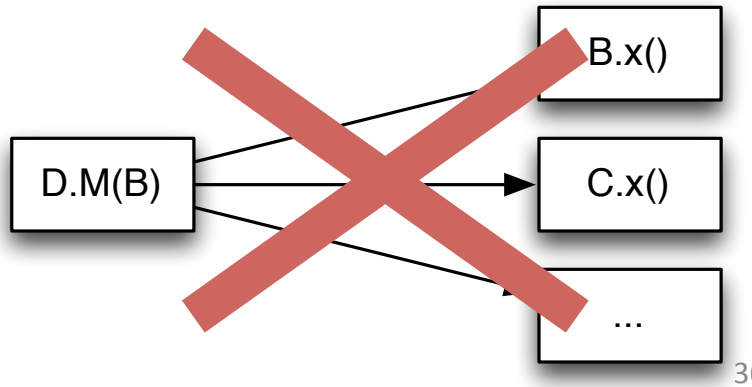
Breaks for Squares!

Call-graph creation (revisited)

```
public class D {
        public void M(B b) {
                b.x()
        }
}
```



A.x() defines the “Contract”



RSSE focuses on reusable information,
(project-) specific information is less important.

Scope of the Static Analysis

- Interprocedural?
- Interprocedural, but intraclass?
- Intraprocedural?

Extract Object Usages…

```
public class C implements I {
    private F f;

    @Override
    public int M1(G g) {
        f.a();
        g.x(f);
        M2();
        M3();
    }

    @Override
    public int M2() {
        f.b()
    }

    private int M3() {
        f.c();
    }
}
```

Interesting Features:

Type
Enclosing (Super-) Class
Enclosing Method
Receiver Callsites
Parameter Callsites
Definition

Analyzed Part of the Source Code is (Usually) Heavily Pruned.
Research on RSSE often focuses on class/method level.

Points-to Analysis

- Context-sensitivity
- Field-sensitivity
- Flow-sensitivity
- …

Many different analysis styles exist, precision varies…

Example

```
public class C {
        private F f;
        public void M() {
                if (…) {
                        f = new F();
                        f.init();
                } else {
                        f = Helper.getF()
                }
                f.doSomething();
        }
}
```

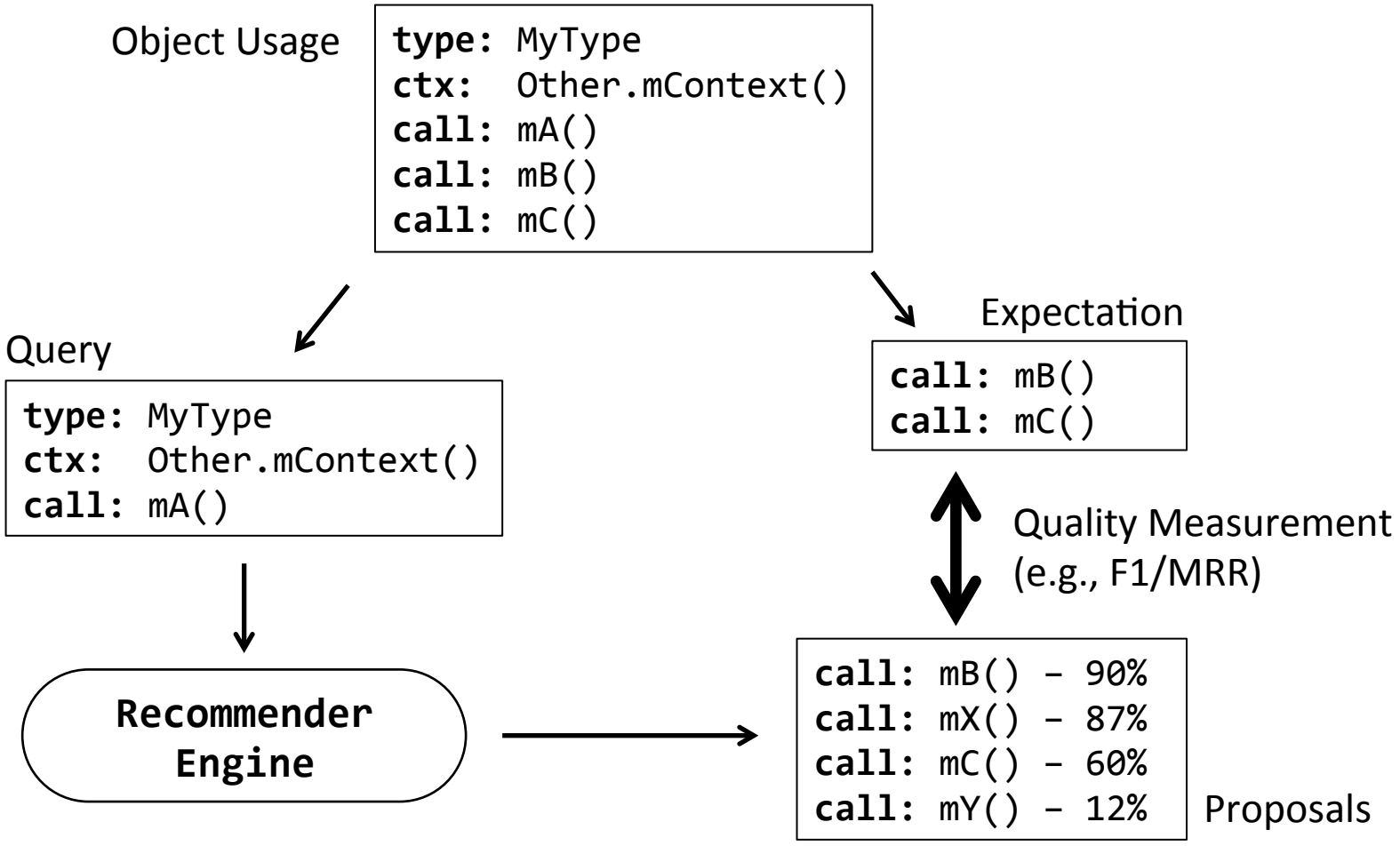
Evaluation of the Differences

- Type-based analysis
- Reference-name-based analysis
- Steensgard-style unification analysis
- Contraint-inclusion analysis

Precision

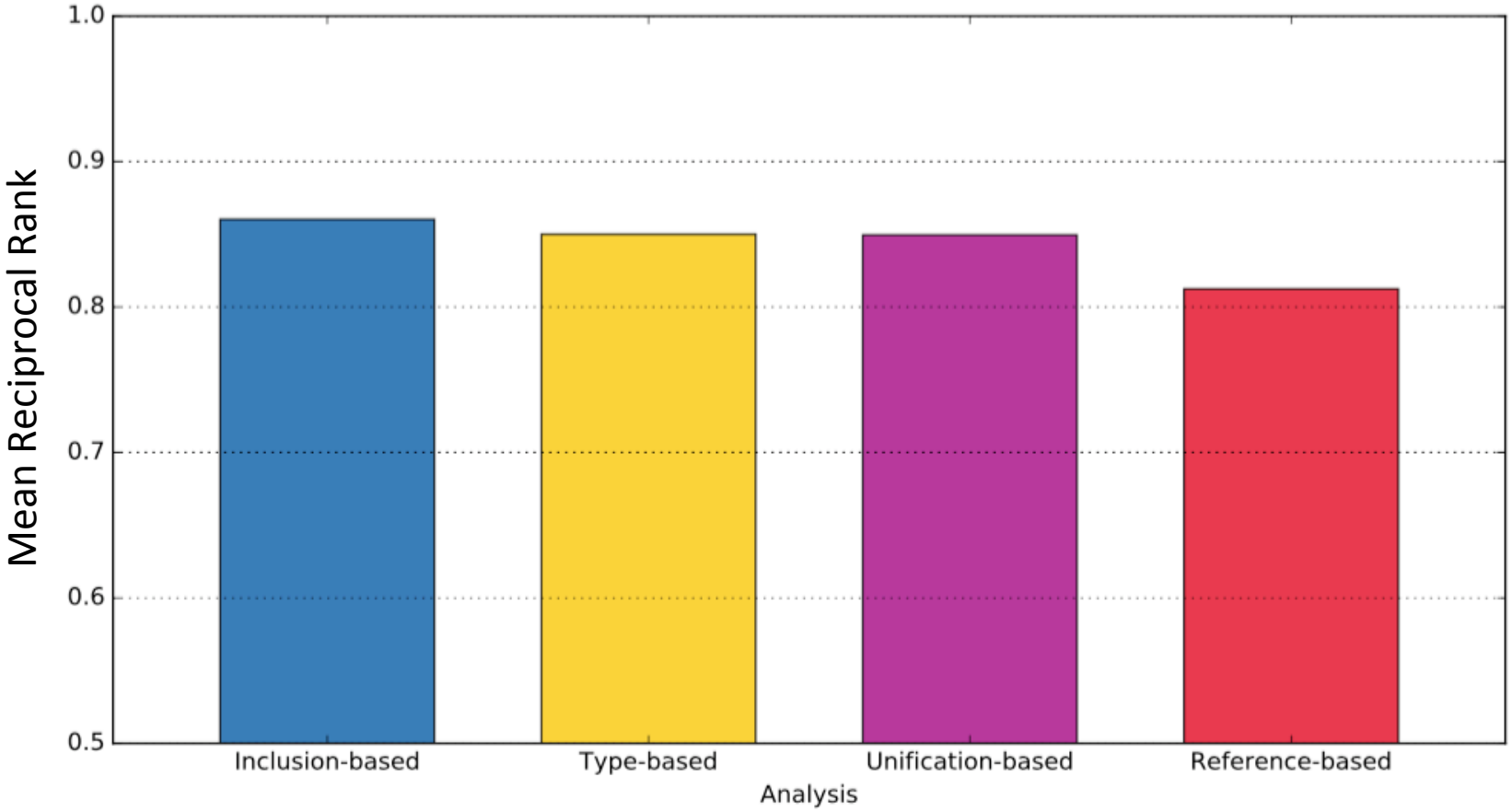
How big is the difference between the analyses?

Artificial Evaluation



MRR Comparison of Diff. P2 Analyses

(in the context of RSSE)



Preliminary Result: Precise Points-to analyses seems to be negligible for RSSE.

TAKE HOME MESSAGE

Take Home Message

- Making the sources analyzable is often first step
- Model and static analysis depend on each other; both strongly depend on the recommendation goal
- Security is interested in the corner case, empirical SE is interested in the common case.
- Accepting under-approximation, the analysis gets easier:
  - Heavily prune…
    - Scope (e.g., intra-class analysis)
    - Call graph (e.g., "first occurrence" of method signature)
  - Strip away (project-) specific information
- RSSE is very specific domain of applied static analyses
- Often no clear recipe exists for static analysis, alternatives need to be explored