A Novel Client-Driven Perspective on Class Hierarchy Understanding and Quality Assessment

Petru Florin Mihancea

LOOSE Research Group

Politehnica University

Timișoara, Romania

Context

> 50 % of software costs generated by maintenance

Class hierarchies

Keys for easy to maintain software (via **polymorphism**)

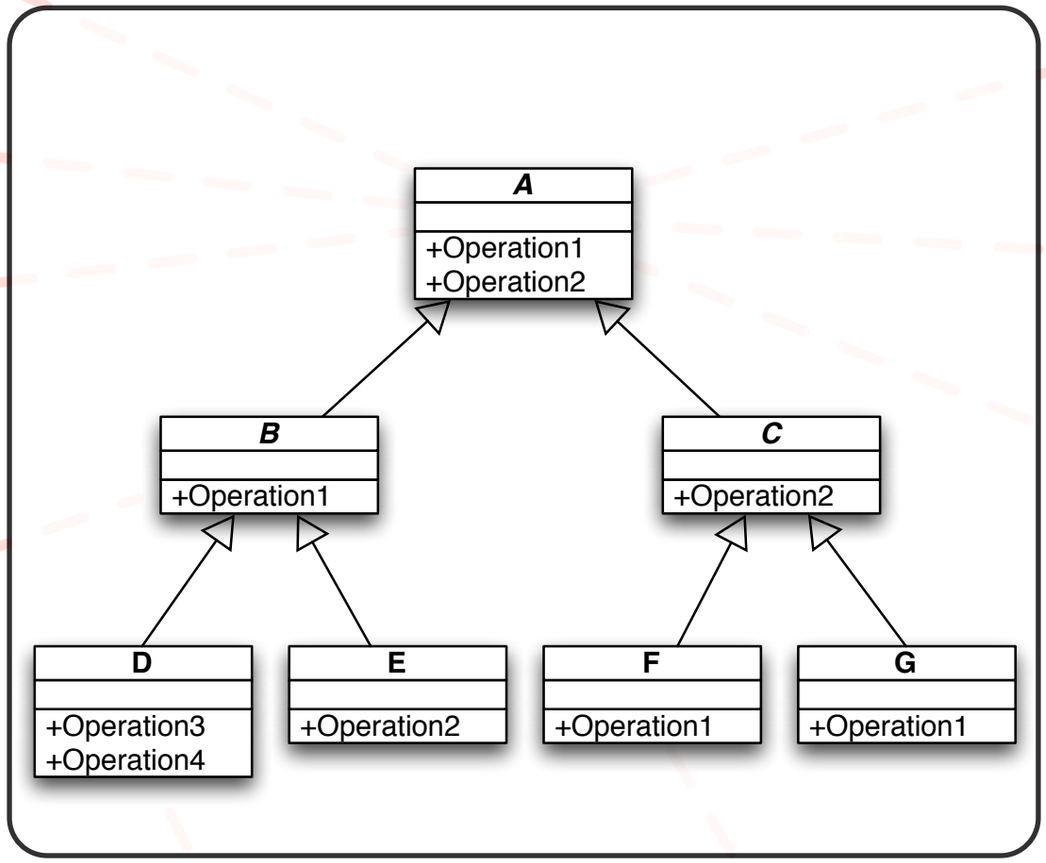
Also affected by the software **aging** phenomenon

Class hierarchy **geriatrics**

Support to understand them and assess their quality

Current Limitation

Do not capture how hierarchies are actually used via polymorphism in a system !



Client 9

Client 8

Client 7

Client 4

nt 3

A model [class hierarchy] cannot be meaningfully validated in **isolation** […] it should be validated in terms of its **clients**

Robert Martin

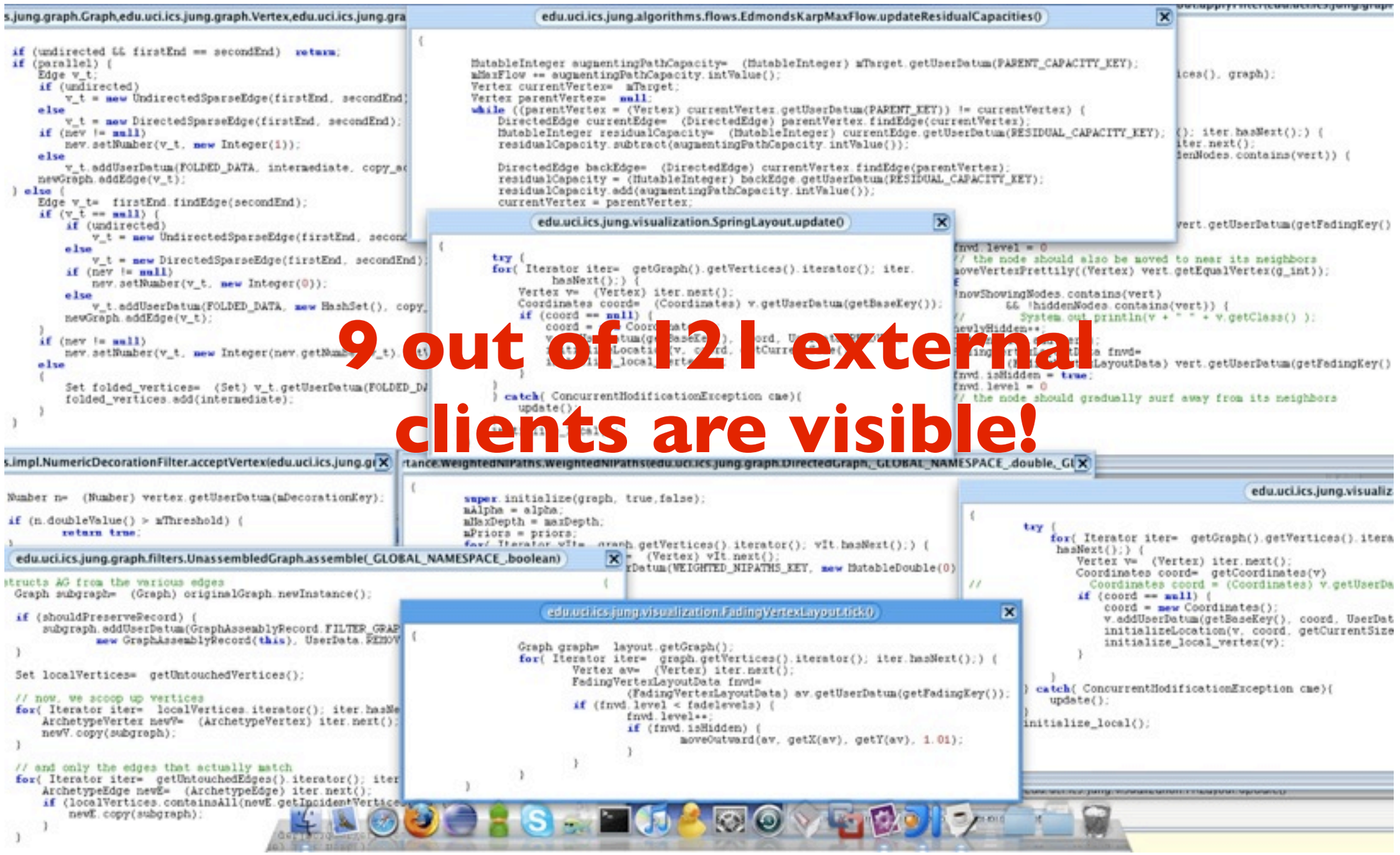
Current Limitation

Issue 1

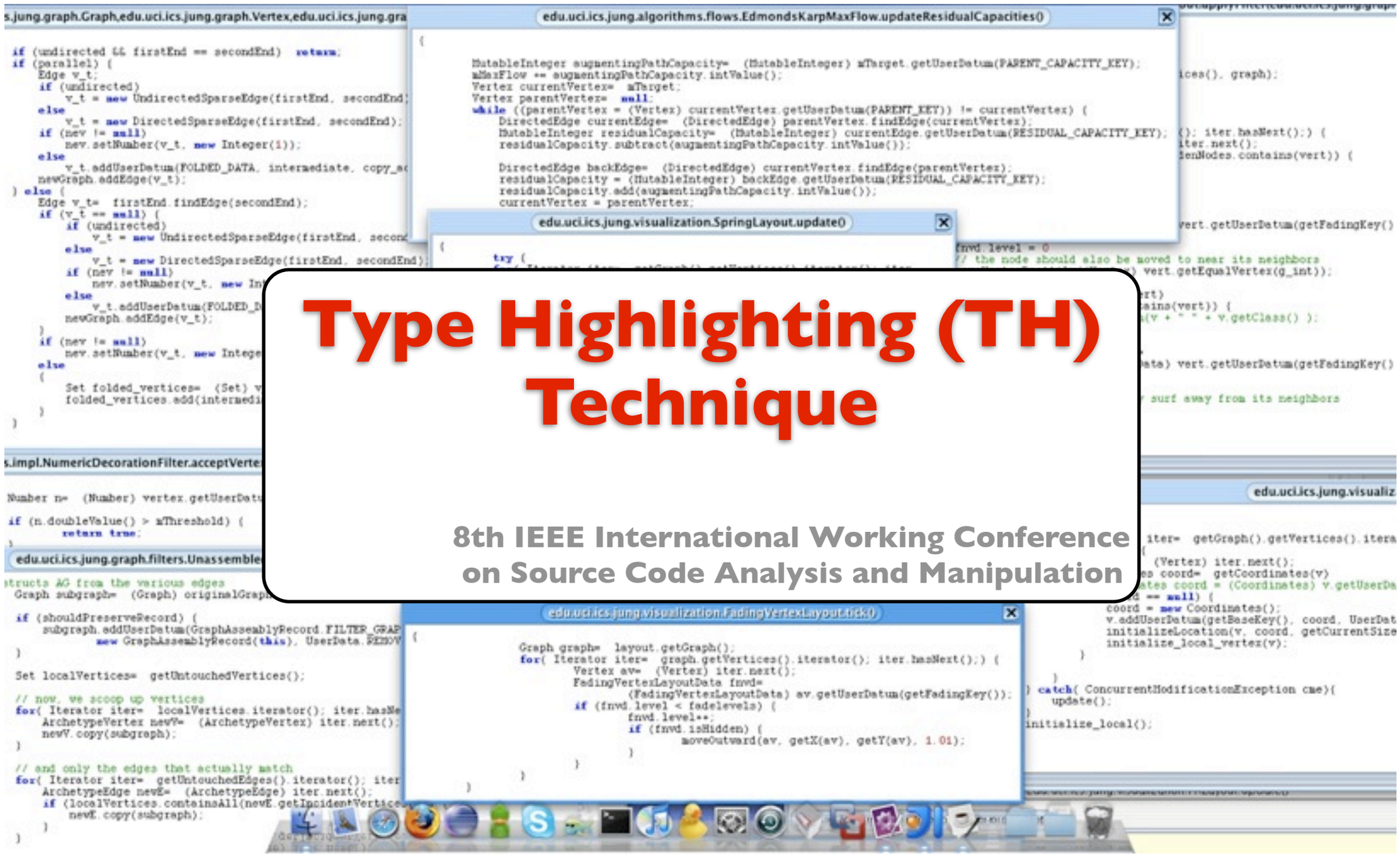
What could be revealed about a hierarchy in this way ?

1. Observe the usage of polymorphism in clients
2. Discover patterns of polymorphism usage
3. Interpret these patterns

Issue 1


edu.uci.ics.jung.algorithms.flows.EdmondsKarpMaxFlow.updateResidualCapacities()
edu.uci.ics.jung.visualization.SpringLayout.update()
edu.uci.ics.jung.graph.filters.UnassembledGraph.assemble(_GLOBAL_NAMESPACE_,boolean)
edu.uci.ics.jung.visualization.FadingVertexLayout.tick()
9 out of 121 external clients are visible!

Issue I



Type Highlighting (TH) Technique

8th IEEE International Working Conference
on Source Code Analysis and Manipulation

TH : Quantifying Polymorphism Usage

1.0

```
void polymorphic(A x) {
  x.p();
}
```

0.25

```
void pPolymorphic(SomeA x) {
  x.p();
}
```

0.0

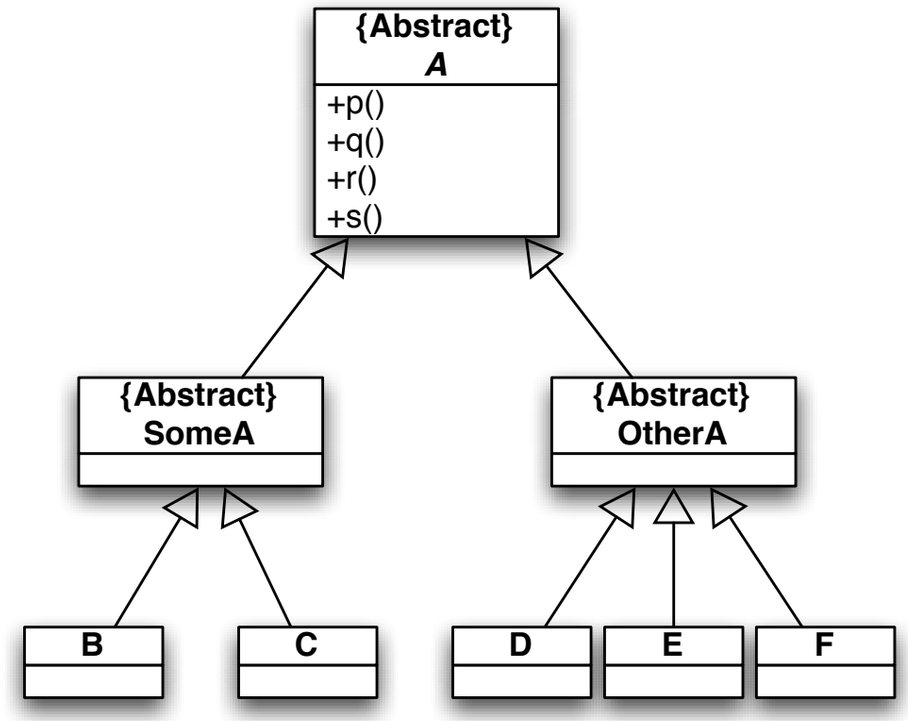
```
void concrete(B x) {
  x.p();
}
```

```
void mixed(A x) {
  if(x instanceof SomeA){
    x.p();
  } else if(x instanceof D){
    x.q();
  } else {
    x.r();
  }
}
```

1.0
0.25
0.5
0.0
0.25

1.0

```
void indirectClient(Intermediate y) {
  A x = y.getAnyAObject();
  x.p();
}
```



Level of Abstraction Metric

Source code token level

~ Relative number of classes whose instances may be referred at the token execution point

TH : Seeing Polymorphism Usage

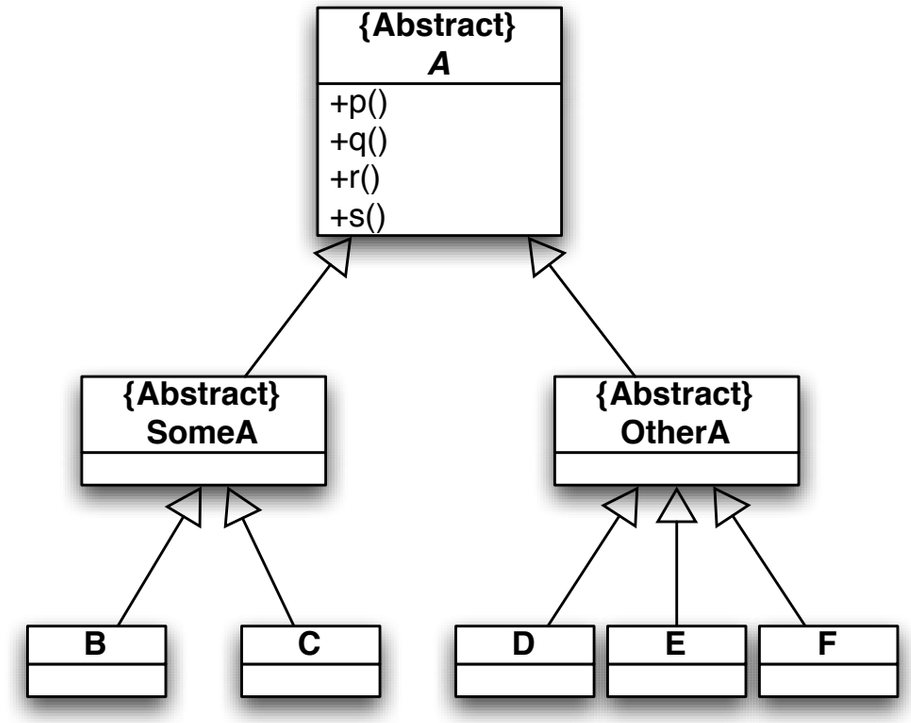
```
void polymorphic(A x) {
   x.p();
}
```

```
void pPolymorphic(SomeA x) {
   x.p();
}
```

```
void concrete(B x) {
   x.p();
}
```

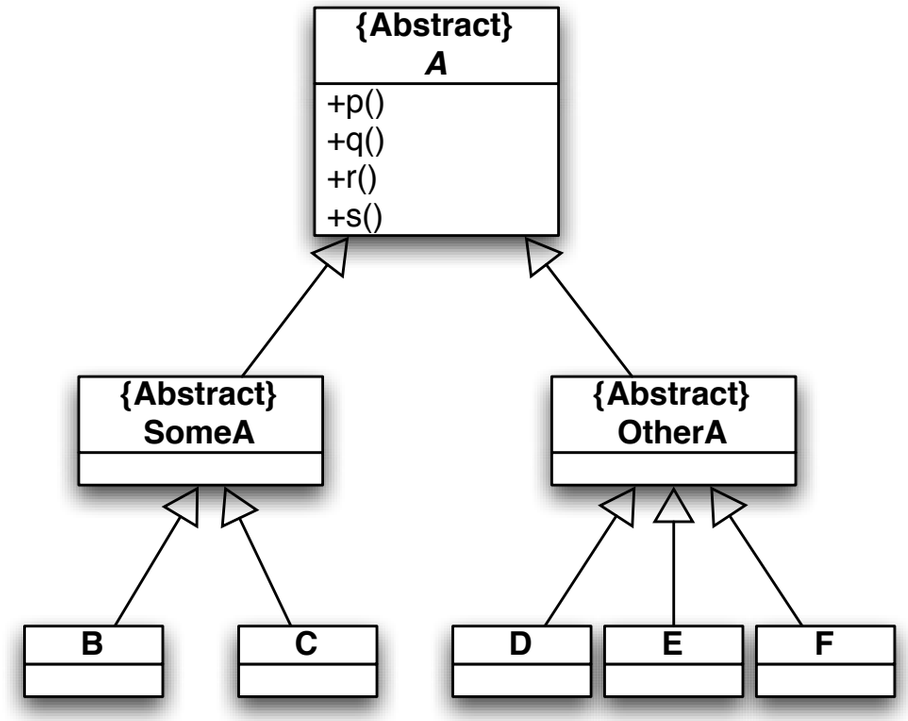
```
void mixed(A x) {
   if(x instanceof SomeA) {
      x.p();
   } else if(x instanceof D) {
      x.q();
   } else {
      x.r();
   }
}
```

```
void indirectClient(Intermediate y) {
   A x = y.getAnyAObject();
   x.p();
}
```



LA Values	Color
$LA(tk) = 1$	- Red
$LA(tk) \geq 0.5 \wedge LA(tk) < 1$	- Diluted Red
$LA(tk) > 0 \wedge LA(tk) < 0.5$	- More Diluted Red
$LA(tk) = 0$	- White
$undefined$	- Light-gray

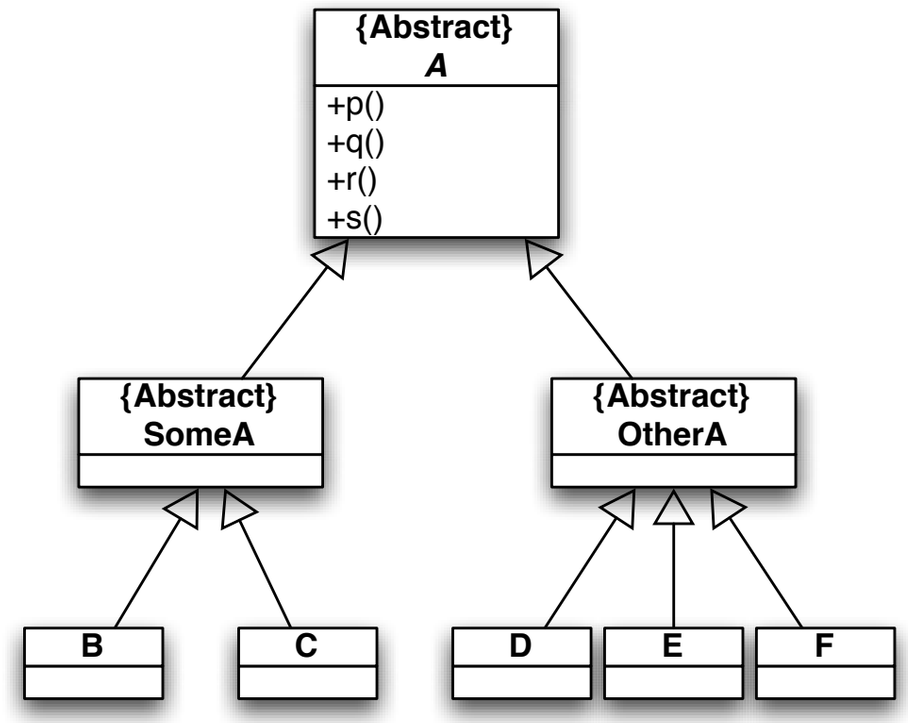
TH : Seeing Polymorphism Usage



LA Values	Color
$LA(tk) = 1$	- Red
$LA(tk) \geq 0.5 \wedge LA(tk) < 1$	- Diluted Red
$LA(tk) > 0 \wedge LA(tk) < 0.5$	- More Diluted Red
$LA(tk) = 0$	- White
$undefined$	- Light-gray

TH : Seeing Polymorphism Usage

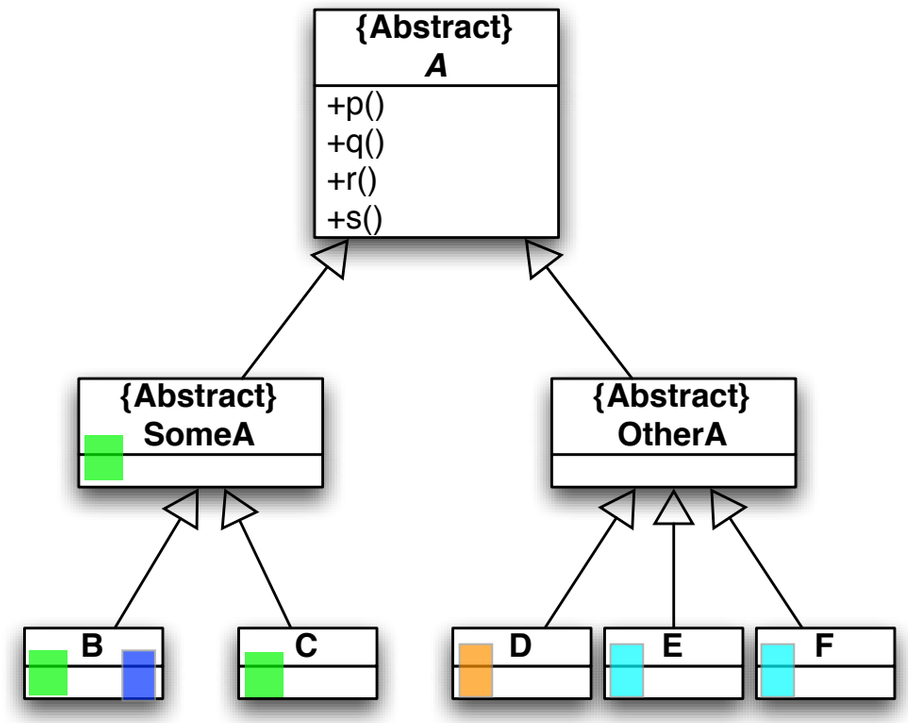
Level of Abstraction View



LA Values	Color
$LA(tk) = 1$	- Red
$LA(tk) \geq 0.5 \wedge LA(tk) < 1$	- Diluted Red
$LA(tk) > 0 \wedge LA(tk) < 0.5$	- More Diluted Red
$LA(tk) = 0$	- White
$undefined$	- Light-gray

TH : Seeing Polymorphism Usage

Level of Abstraction View



Group Discrimination View

LA Values	Color
$LA(tk) = 1$	- Red
$LA(tk) \geq 0.5 \wedge LA(tk) < 1$	- Diluted Red
$LA(tk) > 0 \wedge LA(tk) < 0.5$	- More Diluted Red
$LA(tk) = 0$	- White
$undefined$	- Light-gray

Some Experimental Results. Examples

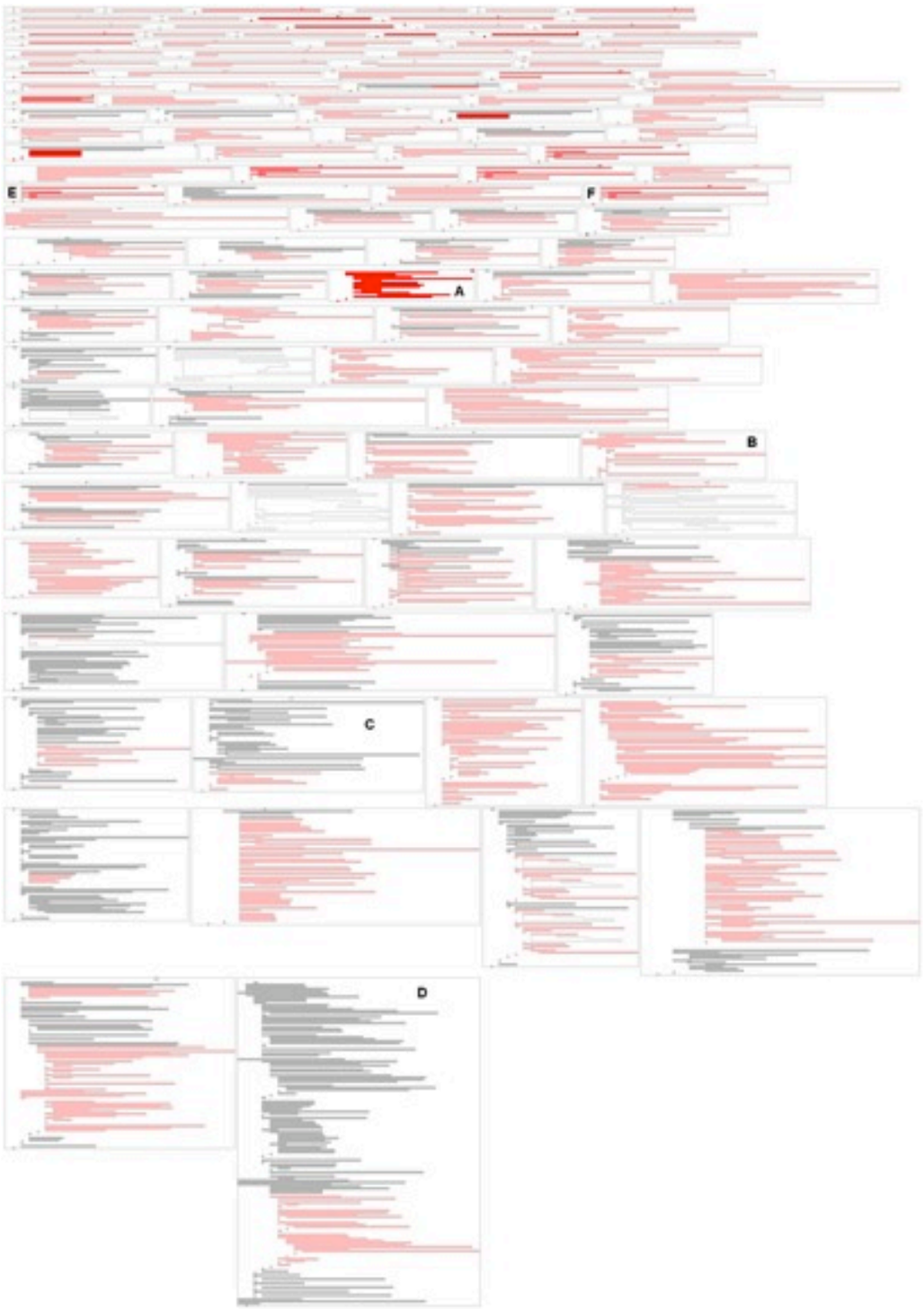
5 Fine-Grained Visual Patterns

Client level

6 Coarse-Grained Visual Patterns

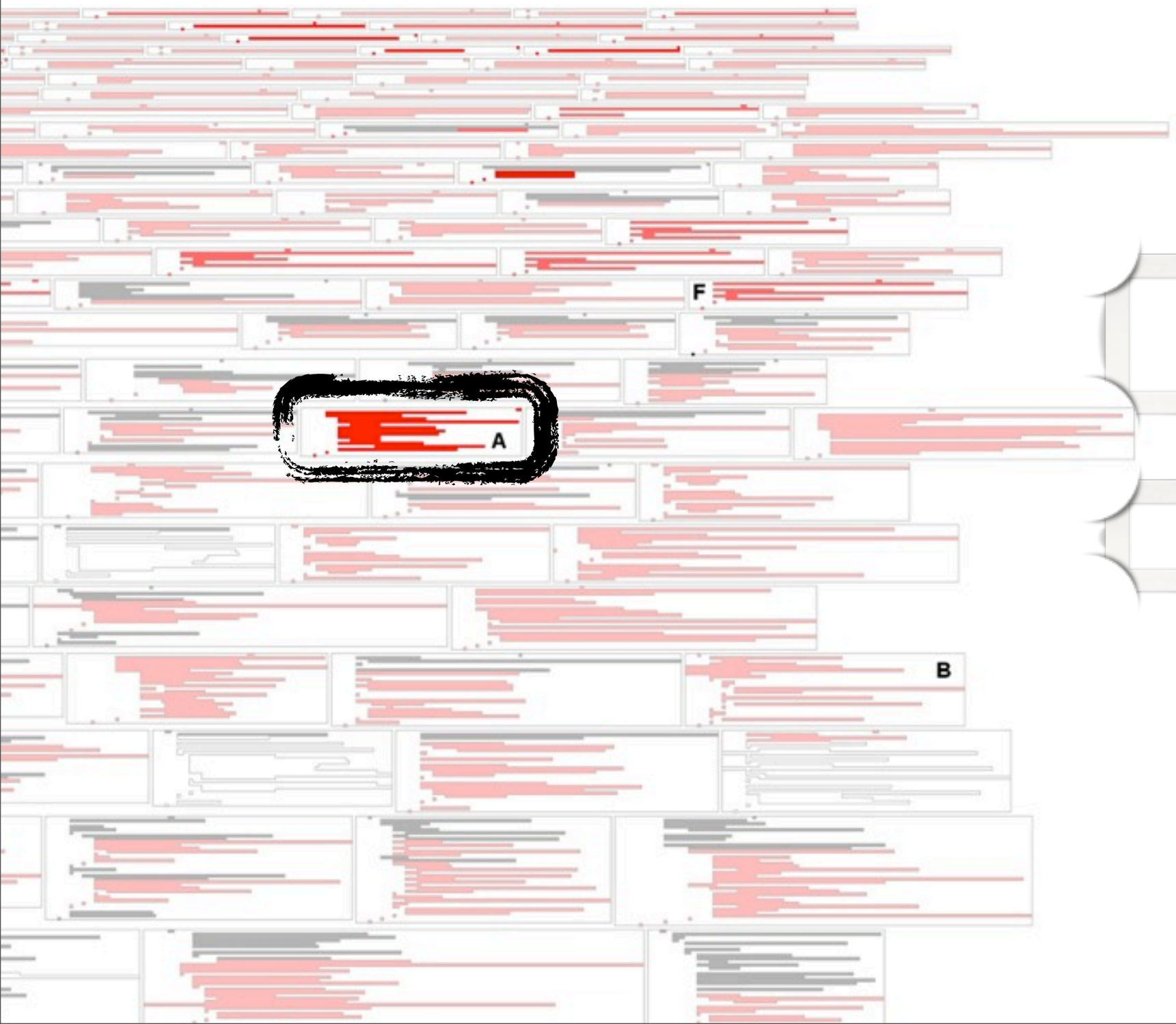
Client Population Level

Some Experimental Results. Examples



Partially Polymorphic
External Population

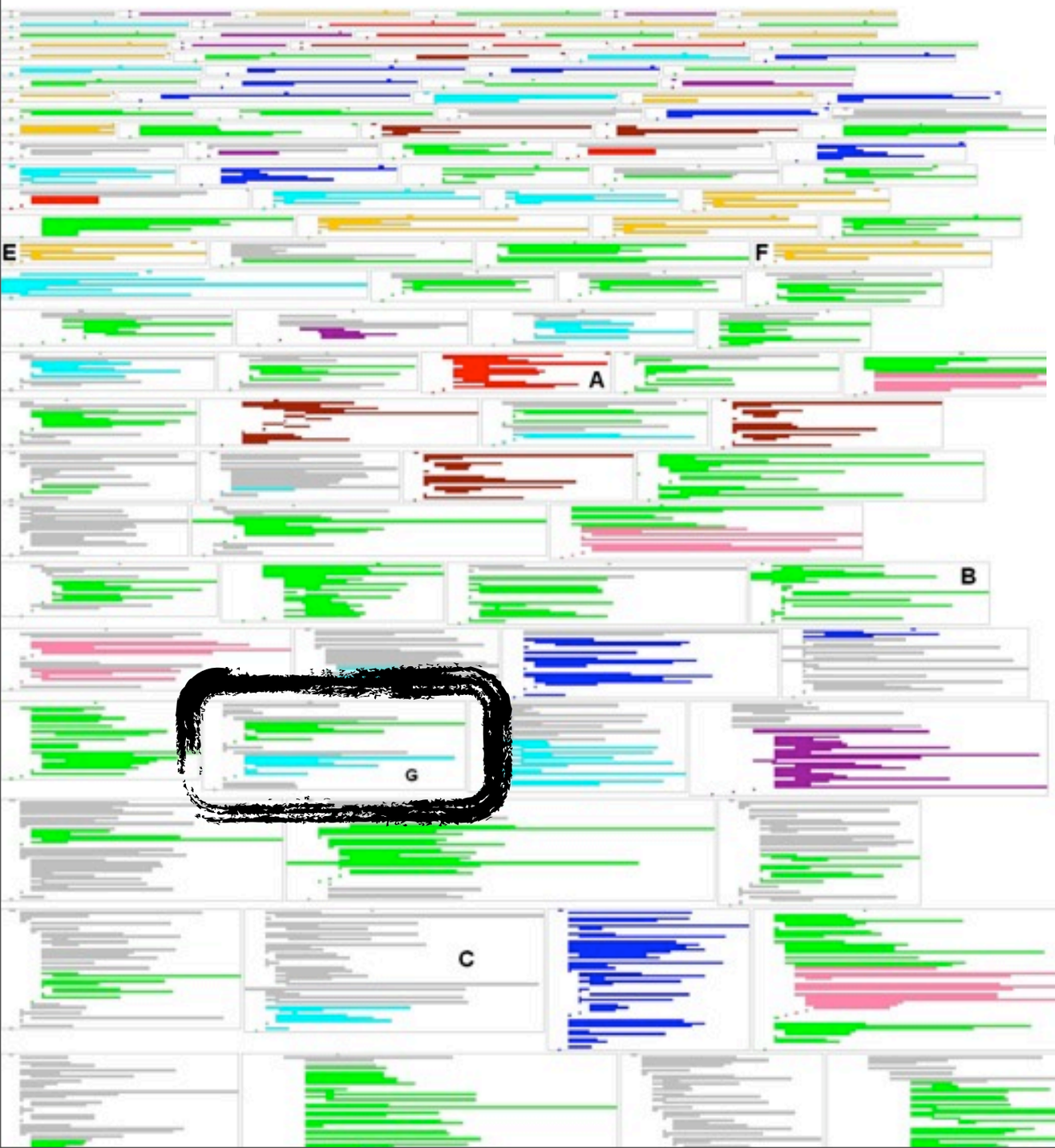
Some Experimental Results. Examples

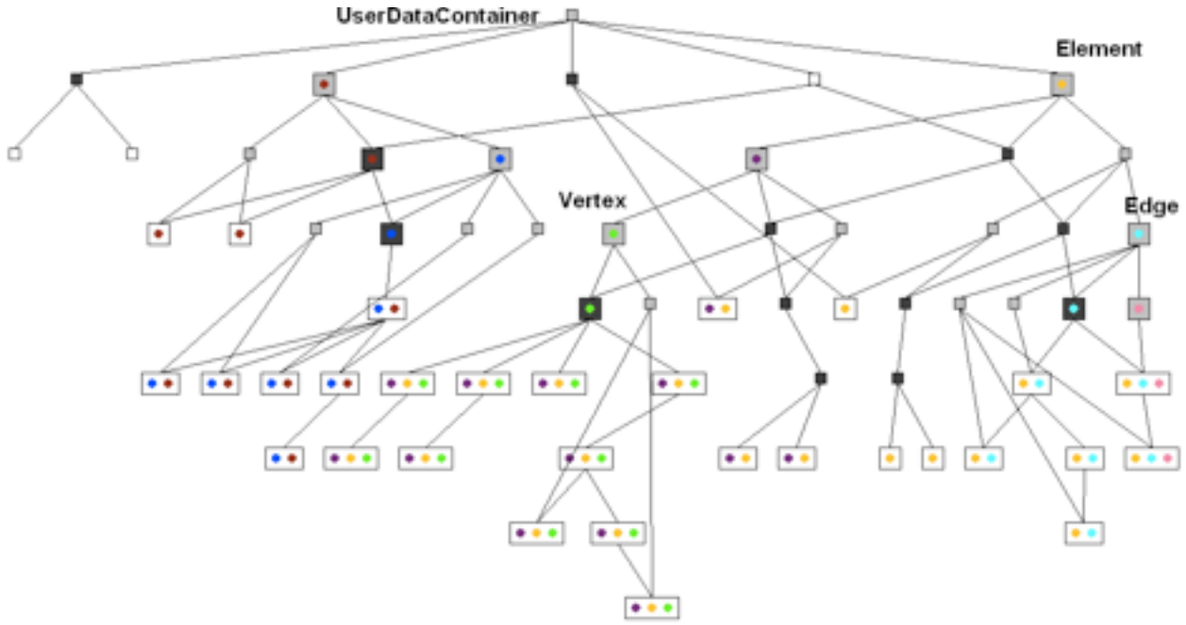


Partially Polymorphic
External Population

Polymorphic Island

Some Experimental Results. Examples





Mixed Twins

Issue 2 : The Dual Nature of Inheritance

Class hierarchies can model

Type hierarchies

Implementation hierarchies

How are they

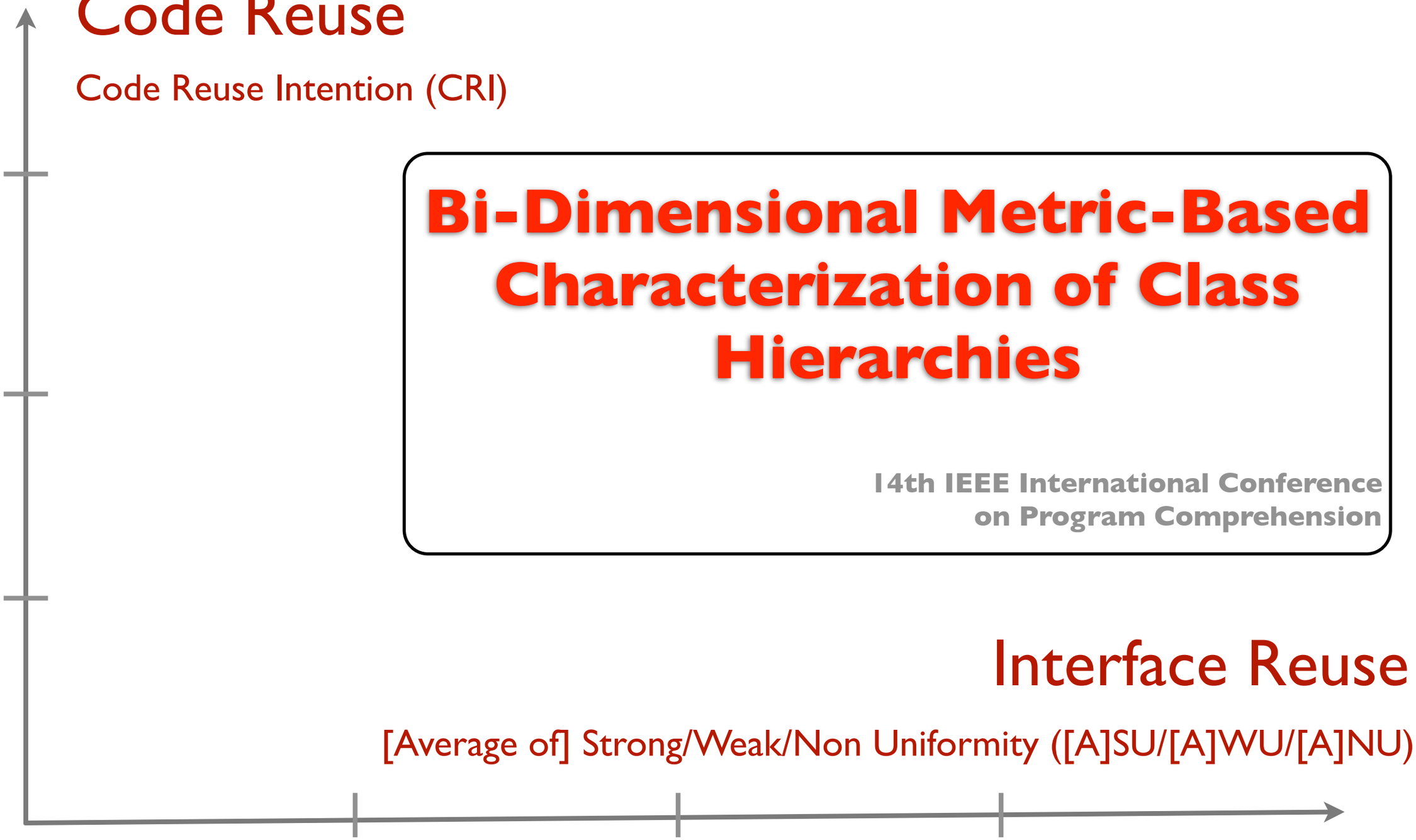
used in a legacy system?

Level of Abstraction View Limitations

Visual detection of interface reuse nature

Almost nothing about code reuse nature

Approach


Code Reuse
Code Reuse Intention (CRI)
Bi-Dimensional Metric-Based Characterization of Class Hierarchies
14th IEEE International Conference on Program Comprehension
Interface Reuse
[Average of] Strong/Weak/Non Uniformity ([A]SU/[A]WU/[A]NU)

Experimental Results

Case studies

System \ Metrics	NOC	NOM	LOC
Recoder	490	6 795	42 259
Jung	391	3 038	22 447
FreeMind	455	5 228	52 904
InternalProduct	124	1 002	11 210

Base classes

System	All Base Classes	Base Classes having $CRI > 0.75$
`Recoder`	219	40
`Jung`	168	54
`Freemind`	239	29
`InternalProduct`	20	5

Detailed investigation

Class	CRI	ASU	AWU	ANU
`AbstractArrayList`	1.0	0.0	0.12	0.88
`LineAdapter`	0.77	0.0	0.88	0.12
`AbstractLayout`	0.94	1.0	0.0	0.0

Experimental Results

Case studies

System \ Metrics	NOC	NOM	LOC
Recoder	490	6 795	42 259
Jung	391	3 038	22 447
FreeMind	455	5 228	52 904
InternalProduct	124	1 002	11 210

Base classes

System	All Base Classes	Base Classes
Recoder	219	
Jung	168	
Freemind	239	
InternalProduct	20	

Detailed investigation

Class		*NU*	*ANU*
AbstractArr		0.12	0.88
LineAd		0.88	0.12
Abstr	0	0.0	0.0

Metrics interpretation is consistent with our manual observations from source code

Issue 3 : Understanding Cost

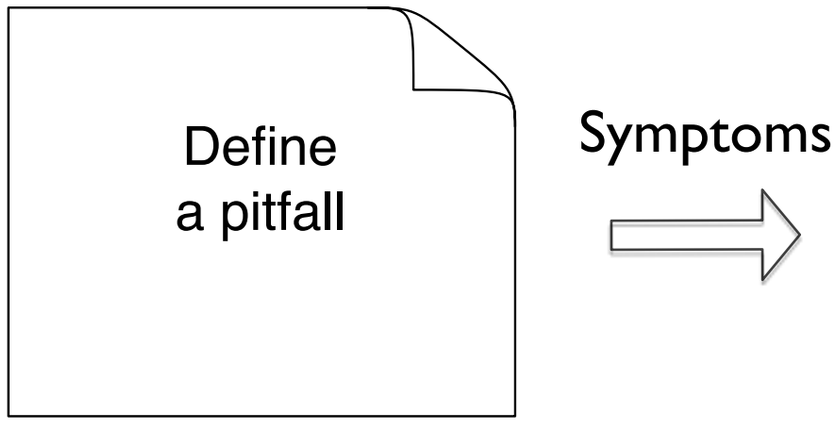
~ 50 % of maintenance costs and includes the time wasted due to **misunderstanding**

Class hierarchies may not be fully intended for polymorphism !

Discover Comprehension Pitfalls in Class Hierarchies

13th European Conference on Software Maintenance and Reengineering

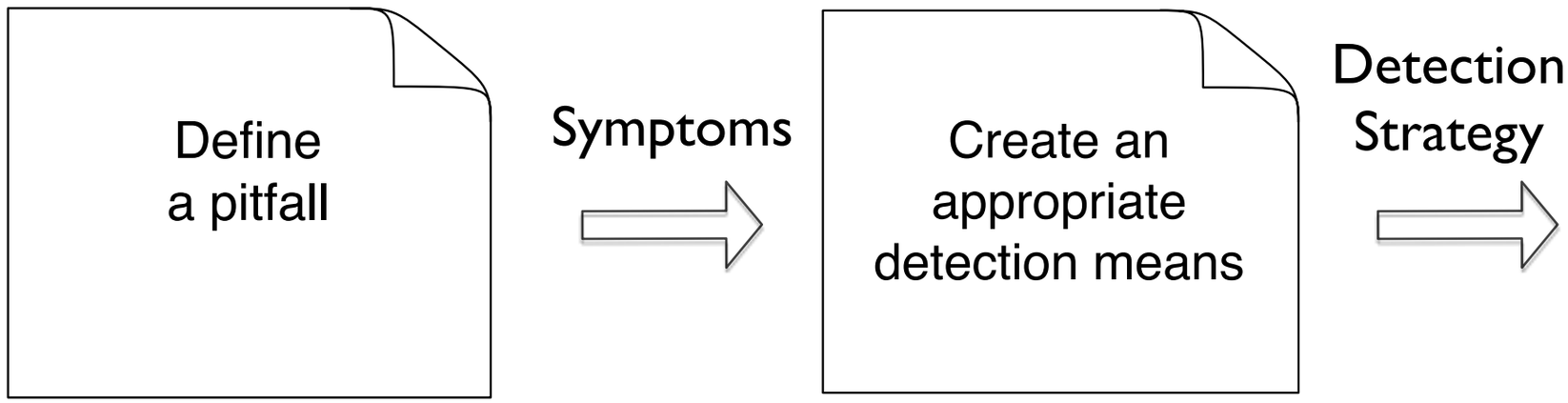
Approach



Fine-grained deviations from design heuristics

Related to polymorphism

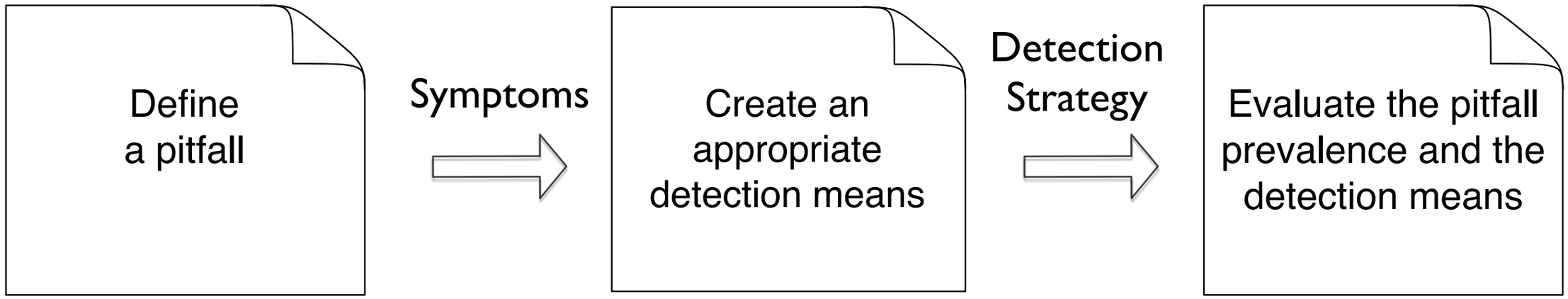
Approach



Create a corresponding metric-based rule

- Uniformity Metrics (SU, ASU, WU, AWU, NU, ANU)
- Type Affinity (TA)

Approach



Apply the rule on case studies

Manual investigation of the suspected entities

	LOC	CLASSES	METHODS
Recoder	42 259	490	6 795
FreeMind	52 904	455	5 228
Jung	22 447	391	3 038

Identified Comprehension Pitfalls

3 Pitfalls

Partial Typing

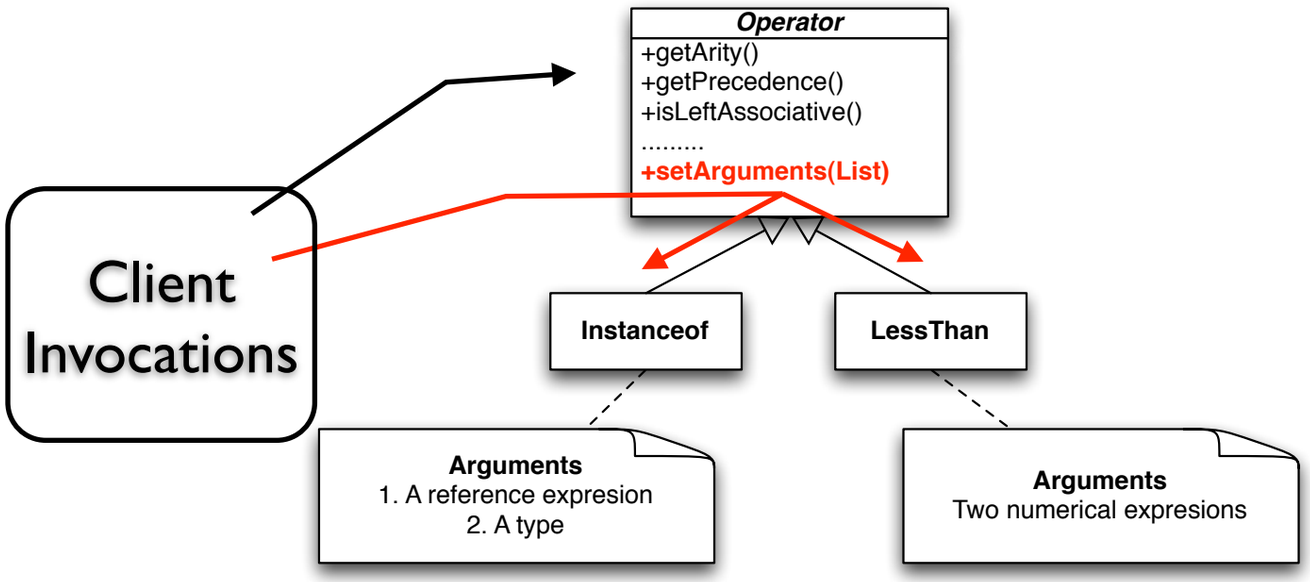
Uneven Service Behavior

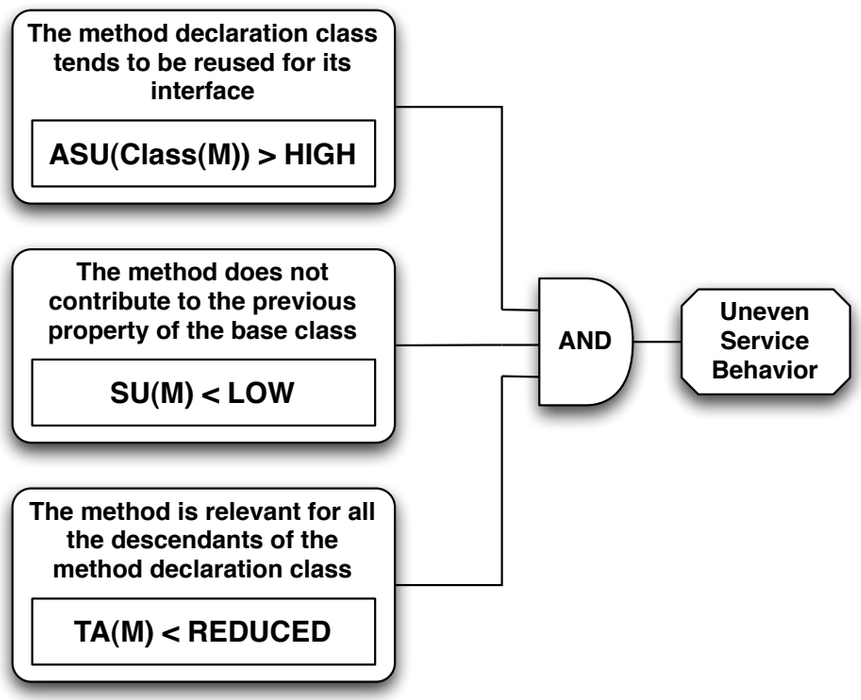
Premature Service

Identified Comprehension Pitfalls

Uneven Service Behavior

- Affected entity - base class method
- The method only enforces a non-uniform service
- The other methods in base class have an uniform semantics





Experimental Results. Detection Precision

Uneven Service Behavior

	Suspected	True Positives
Recoder	21	13
FreeMind	8	7
Jung	3	3
TOTAL	32	23

~ 72 %

Partial Typing

	Suspected	True Positives
Recoder	9	9
FreeMind	2	2
Jung	10	8
TOTAL	21	19

~90 %

Premature Service

	Suspected	True Positives
Recoder	7	4
FreeMind	6	2
Jung	0	0
TOTAL	13	6

~ 46 %

Conclusions & Future Work

3 promising contributions

- Type highlighting
- Metric-based bi-dimensional characterization
- Comprehension pitfalls and their detection strategies

Some limitations

- Metrics estimation using intra-procedural SCA

Some future work

- Stronger case studies with disinterested human subjects