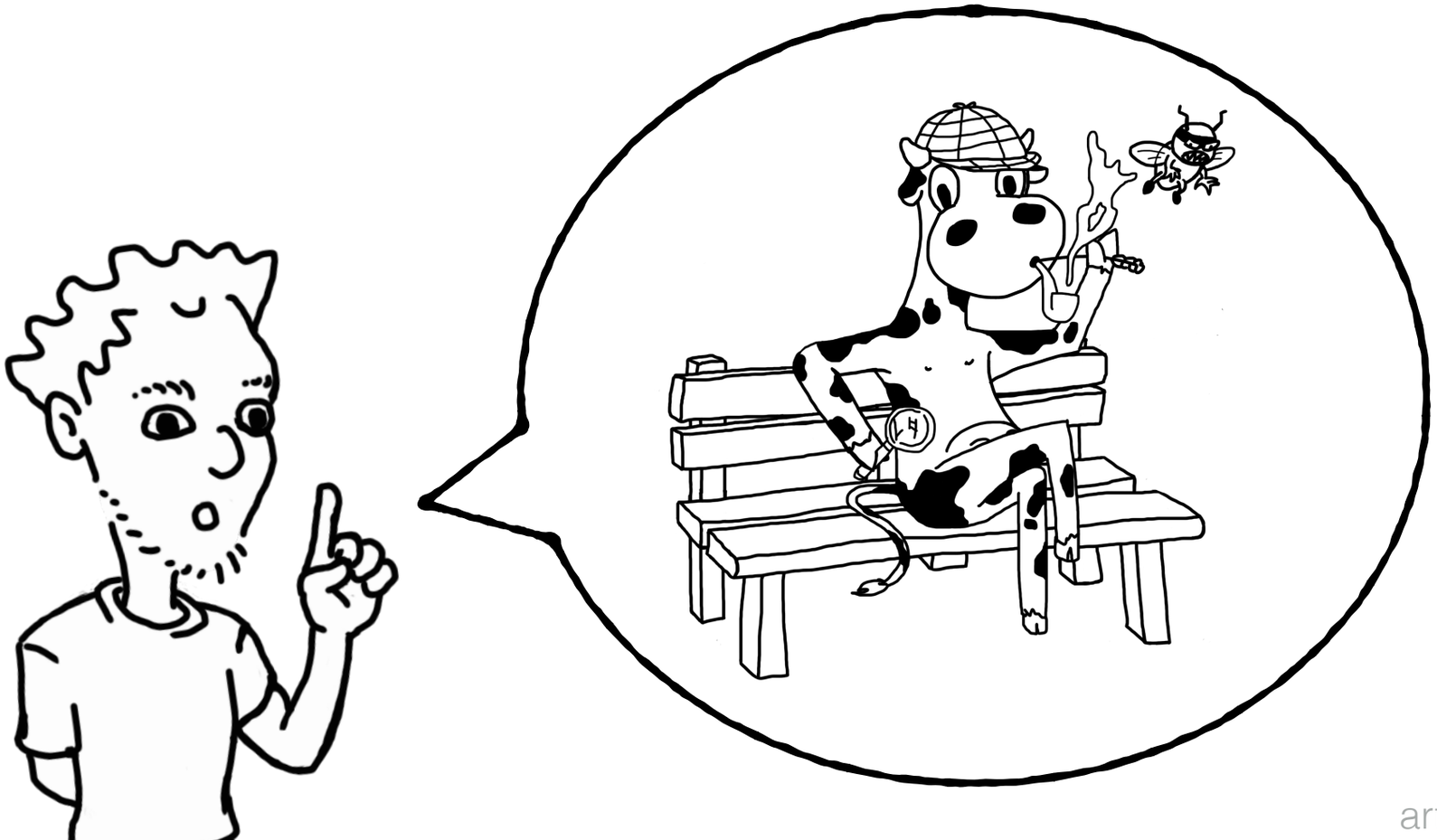
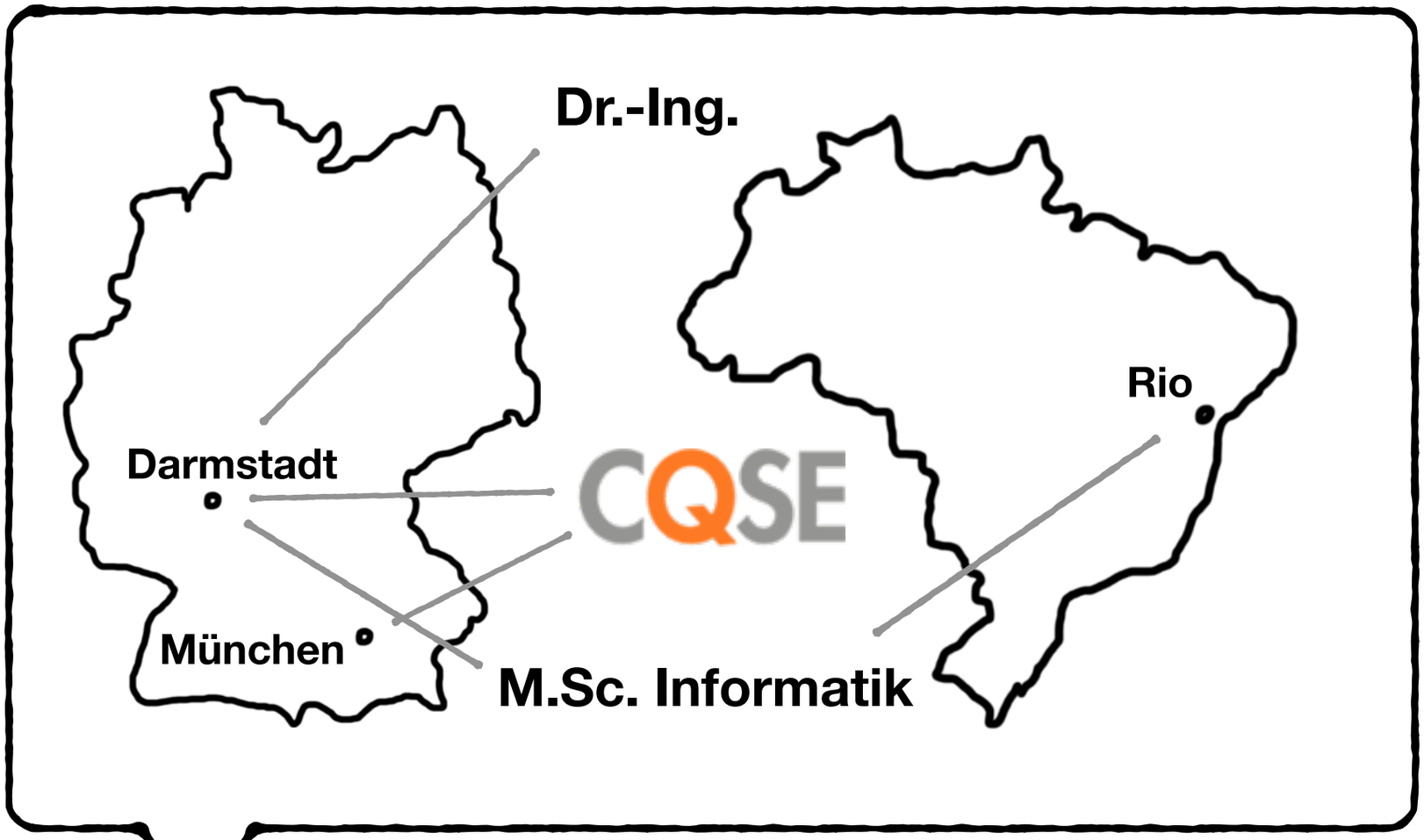
Investigating Next Steps in Static API-Misuse Detection

Sven Amann @ SE - 27. February 2020



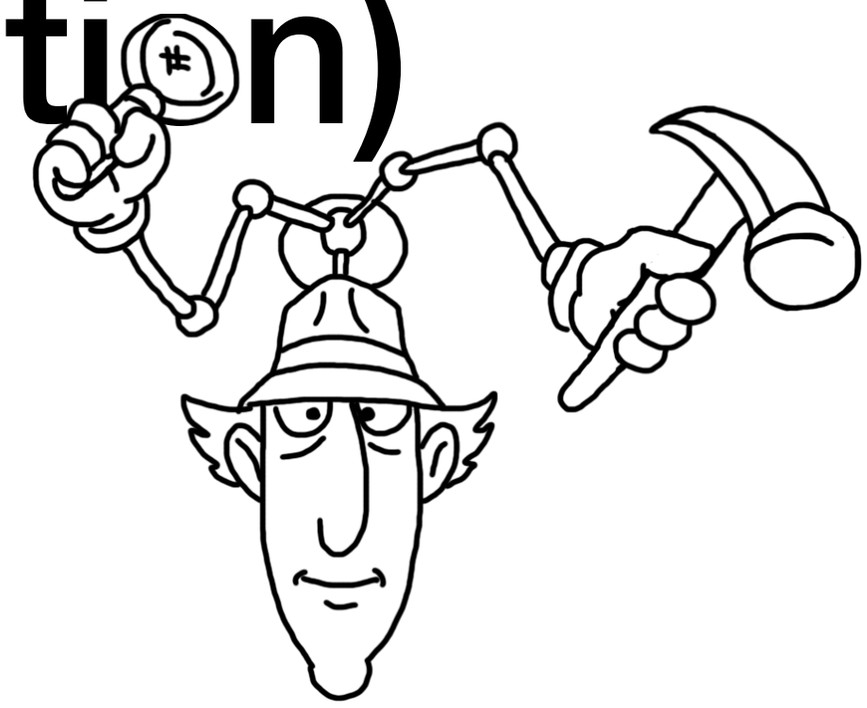


Sven Amann

AcademicsCode.com

@svamann

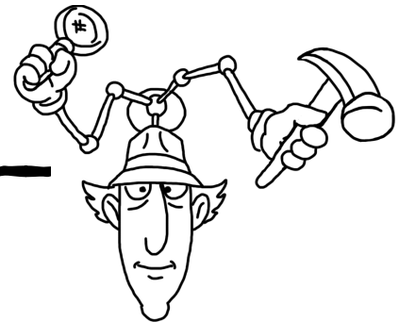
API Misuse (Detection)

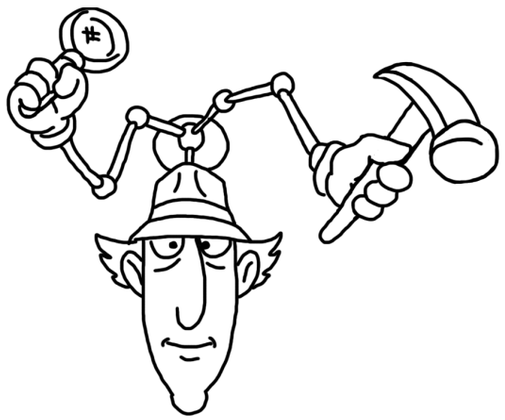


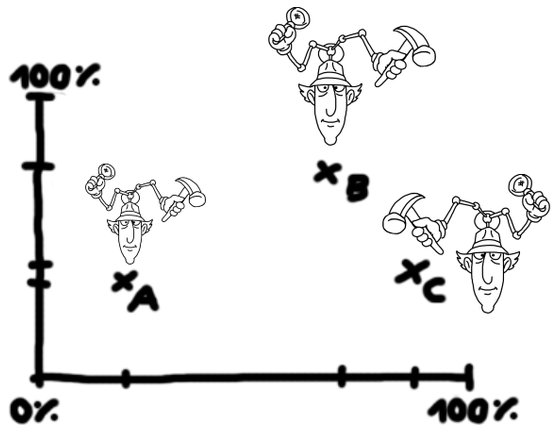
```
Collection<String> files = …;

Iterator<String> it = files.iterator();

String first = it.next();
```

```
Collection<String> files = …;

Iterator<String> it = files.iterator();
if (it.hasNext())
  String first = it.next();
```

Verification (Detection)

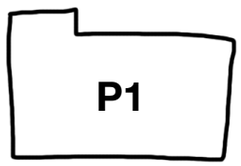
Specifications (Patterns)		Static	Dynamic
Mined	Static	 **API-Misuse Detectors**	
	Dynamic		
Hand-crafted		**Specification Verifiers**	

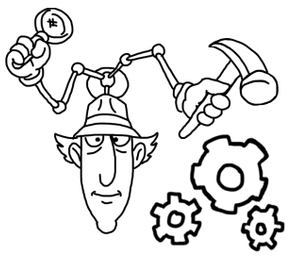


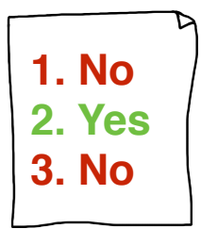


TSE'17

PR-Miner
Chronicler
Colibri/ML
Jadet
RGJ07
LKL08
Alattin
AX09
Car-Miner
GROUMiner
OCD
DMMC
SpecCheck
RRFinder
Tikanga
PJAG12
PG12
DroidAssist







Number of Projects

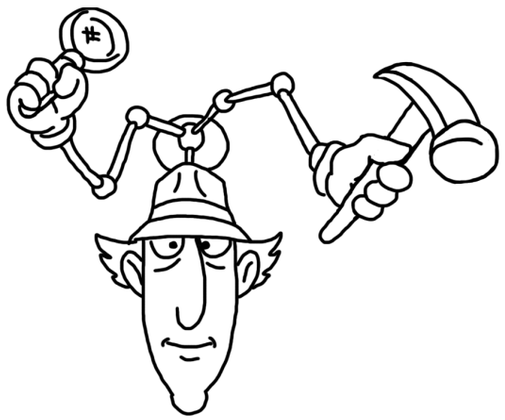
1-20 (avg. 5.7, median 5)

Evaluation Setting

Per-project: 16
Cross-project: 4
Multi-project: 1

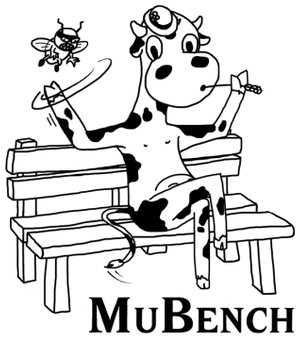
Precision in Top-X Findings

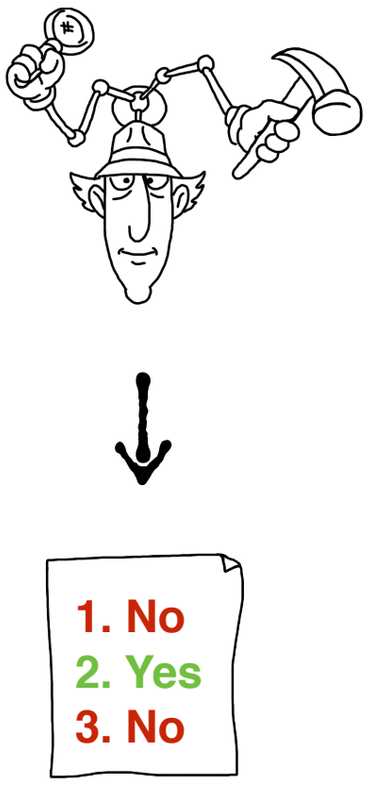
6.5% ≤ p ≤ 100%

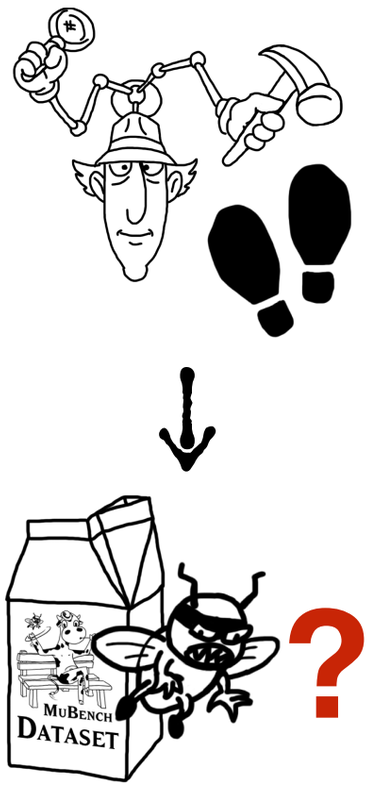


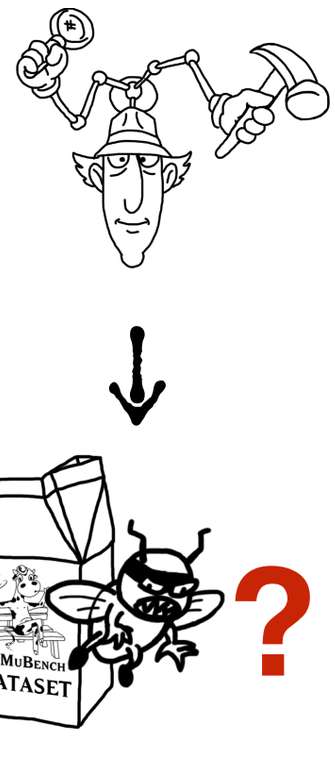
	Missing Method Calls	Redundant Method Calls	Missing null Checks	Missing Value/State Conditions	Missing Sync. Conditions	Missing Context Conditions	Redundant Conditions	Missing Exception Handling	Redundant Exception Handling	Missing Iteration	Redundant Iteration
PR-Miner	●	○	○	○	○	○	○	○	○	○	○
Chronicler	●	○	○	○	○	○	○	○	○	○	○
Colibri/ML	●	○	○	○	○	○	○	○	○	○	○
Jadet	●	○	○	○	○	○	○	○	○	◐	○
RGJ07	◐	○	●	●	○	○	○	○	○	○	○
LKL08	●	○	○	○	○	○	○	○	○	○	○
Alattin	◐	○	●	◐	○	○	○	○	○	○	○
AX09	◐	○	◐	◐	○	○	○	●	○	○	○
Car-Miner	◐	○	○	○	○	○	○	●	○	○	○
GROUMiner	●	○	◐	◐	○	○	○	○	○	◐	○
OCD	●	○	○	○	○	○	○	○	○	◐	○
DMMC	●	○	○	○	○	○	○	○	○	○	○
SpecCheck	●	○	○	○	○	○	○	○	○	○	○
RRFinder	●	○	○	○	○	○	○	○	○	○	○
Tikanga	●	○	○	○	○	○	○	○	○	◐	○
PJAG12	●	●	○	◐	○	○	○	○	○	○	○
PG12	◐	◐	○	◐	○	○	○	○	○	○	○
DroidAssist	●	●	○	○	○	○	○	○	○	○	○

TSE'17





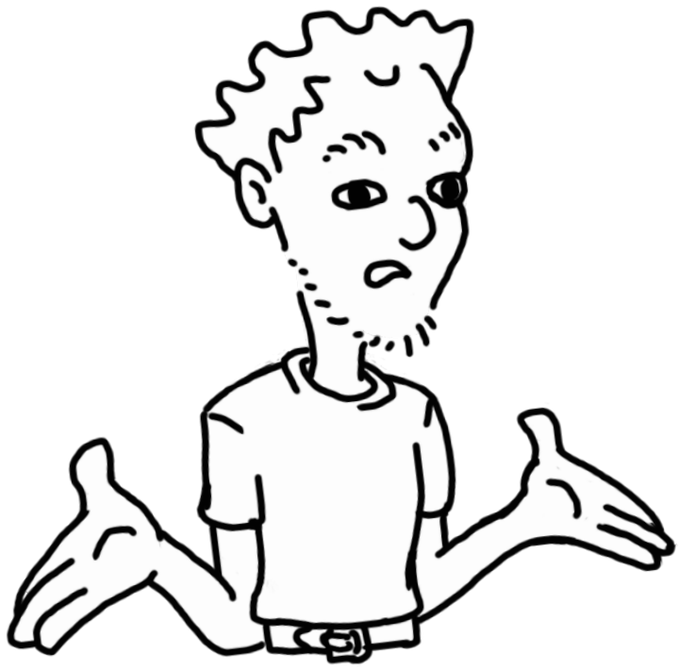




	Precision @20	Recall Upper Bound	Recall
Jadet	10.3%	23.4%	5.7%
GrouMiner	0.0%	48.4%	0.0%
Tikanga	11.4%	20.3%	13.2%
DMMC	9.9%	23.4%	20.8%

API Misuses cause ~10%
of all software bugs.

API-misuse detectors focus on
only a subset of all misuses.

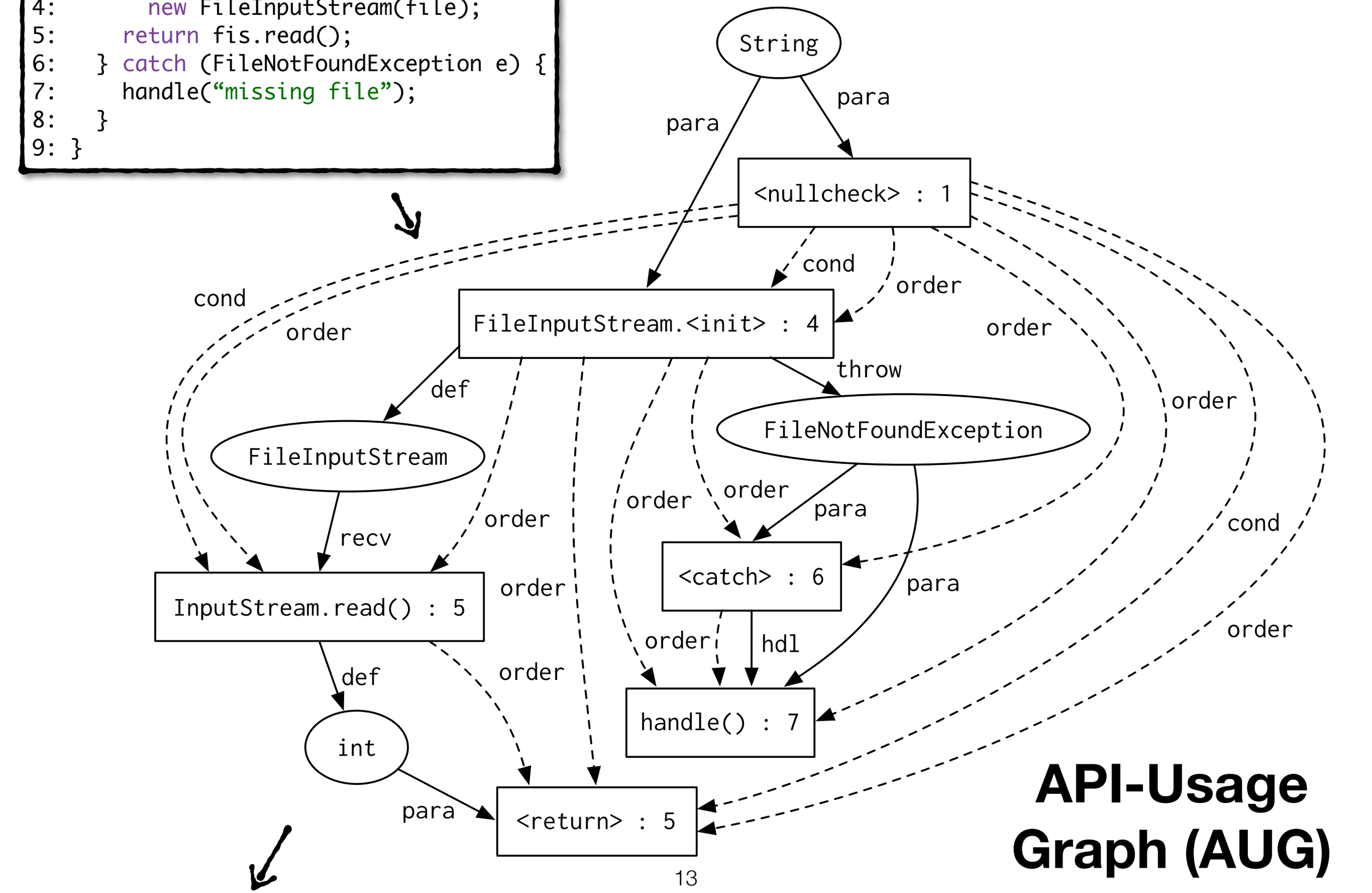


Detectors have very low precision
and recall in practice.

MuDetect
The Next Step

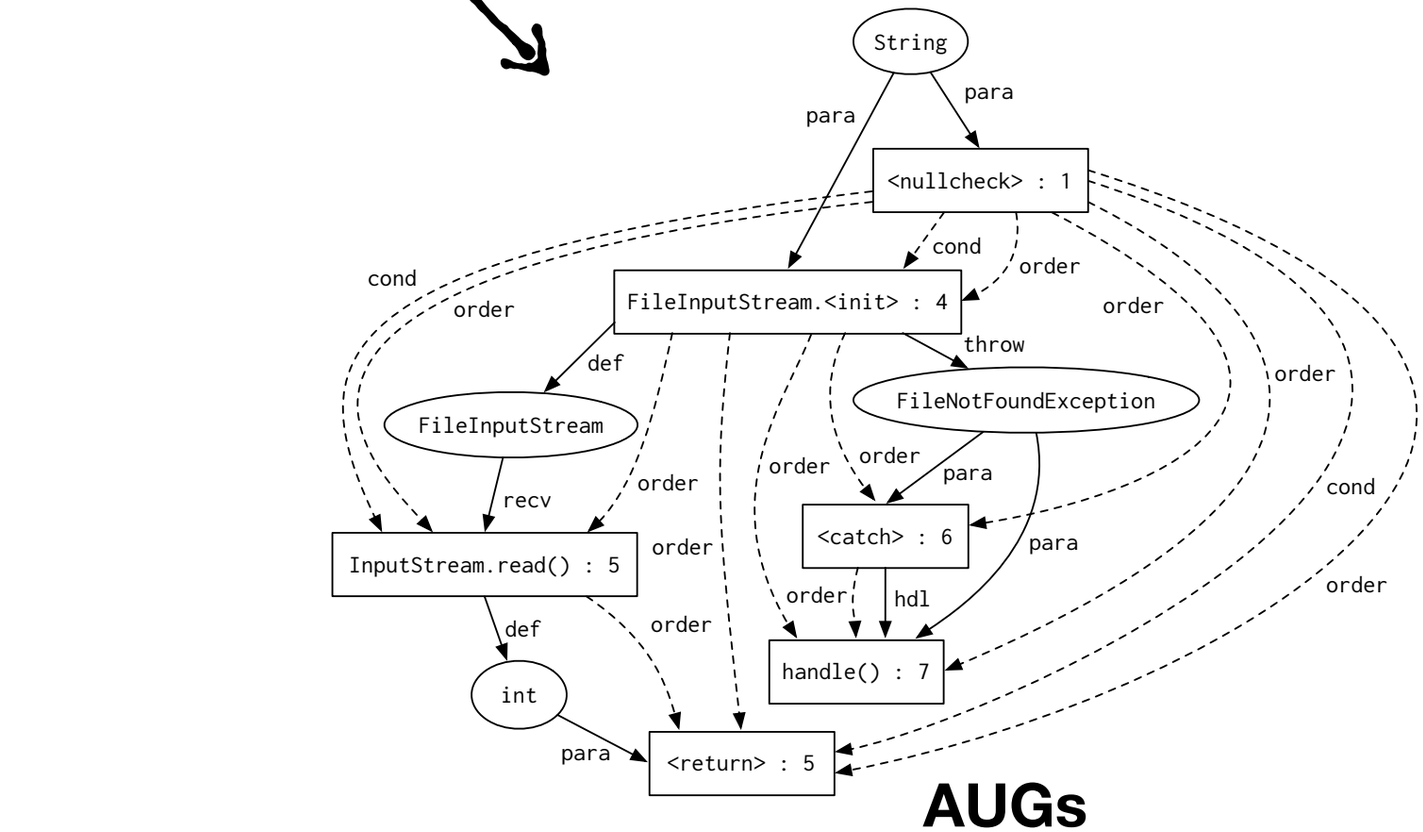

```
1: if (file != null) {
2:   try {
3:     FileInputStream fis =
4:       new FileInputStream(file);
5:     return fis.read();
6:   } catch (FileNotFoundException e) {
7:     handle(“missing file”);
8:   }
9: }
```

```
1: if (file != null) {
2:   try {
3:     FileInputStream fis =
4:       new FileInputStream(file);
5:     return fis.read();
6:   } catch (FileNotFoundException e) {
7:     handle(“missing file”);
8:   }
9: }
```



API-Usage Graph (AUG)

MuDetect



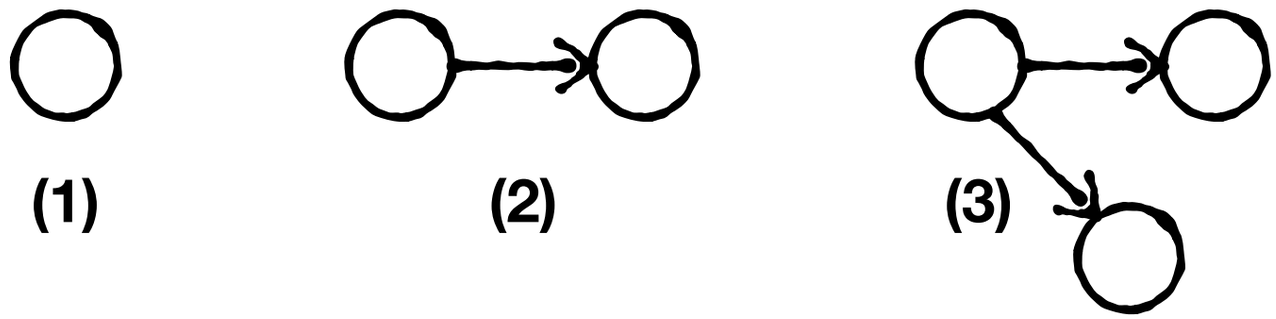
AUGs

Training

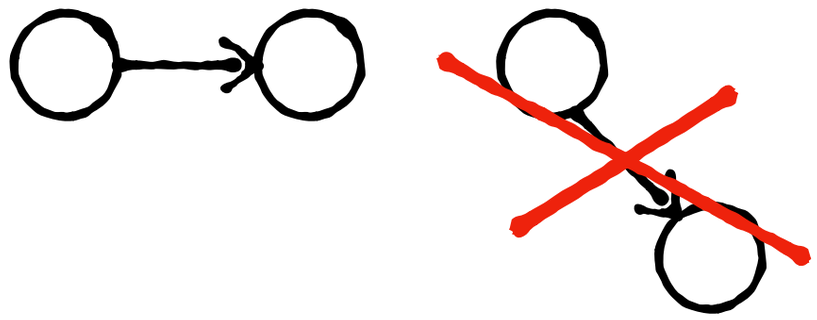
MuDetect

Training

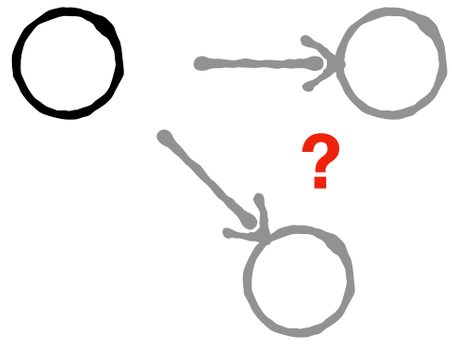
Apriori-based Mining



Greedy Extension



Semantic Extension



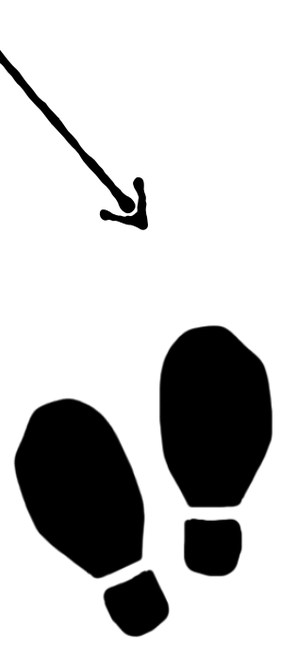
1) Call with side-effect?
2) Call producing data?
3) Operator consuming and producing data?
4) Used data node?
5) Condition with data relation?

MuDetect

Training

Patterns

MuDetect



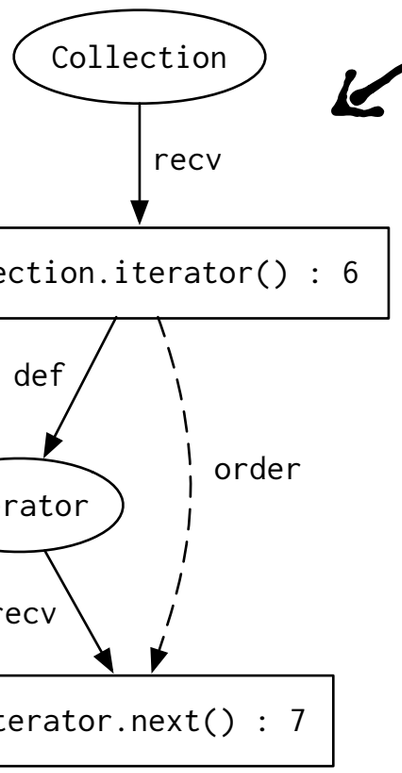
Patterns

MuDetect

```
1: Collection<String> fs = …;
...
6: Iterator<String> it = fs.iterator();
7: String first = it.next();
```

Target Code

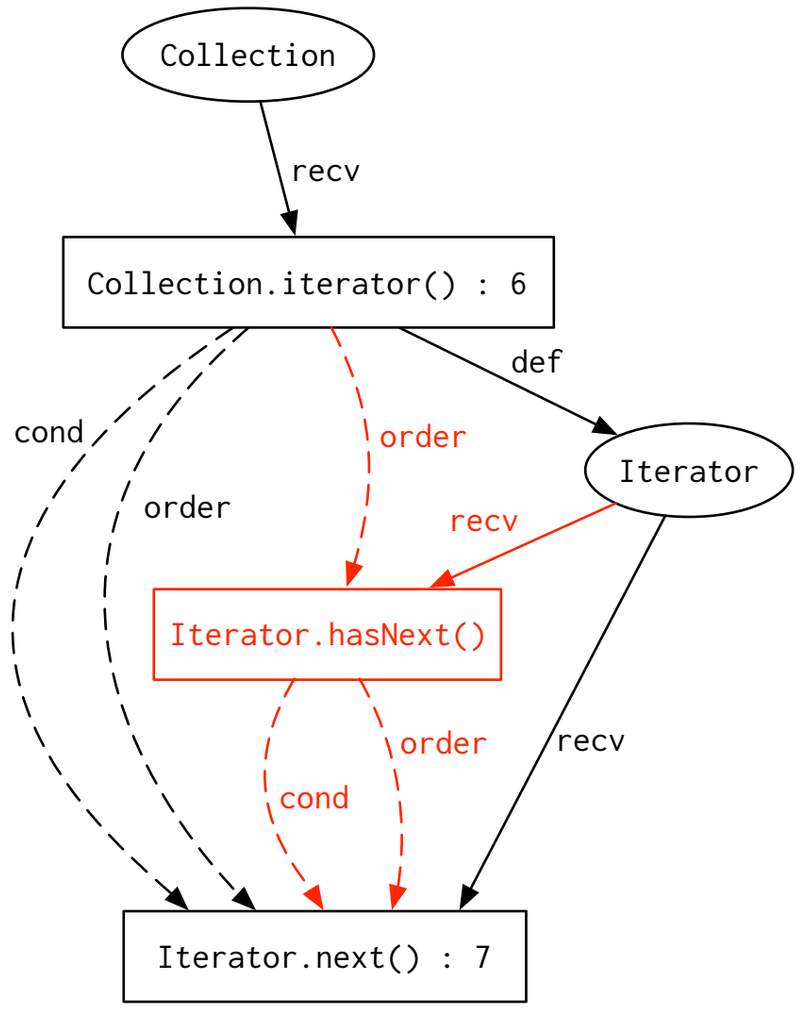
Patterns



AUGs

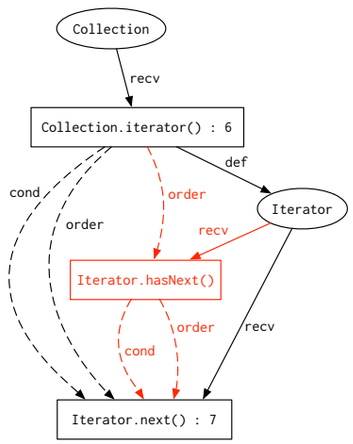
MuDetect

```
1: Collection<String> fs = …;
   ...
6: Iterator<String> it = fs.iterator();
7: String first = it.next();
   ...
```



Violations

MuDetect



Violations

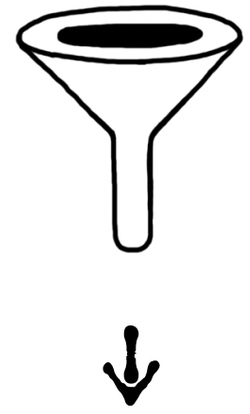
Patterns

Filtering

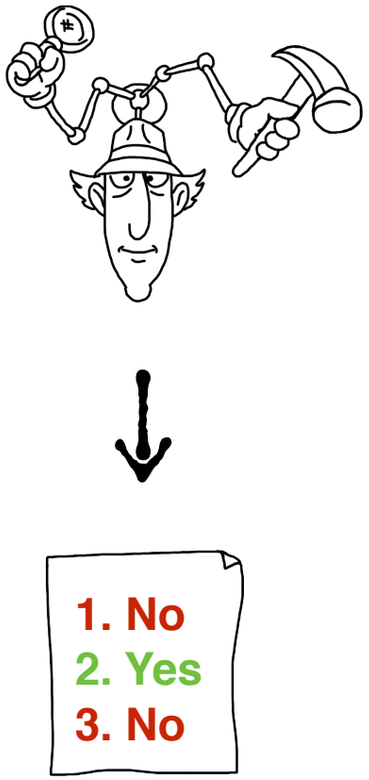
- Alternative-Pattern Instances
- Alternative Violations

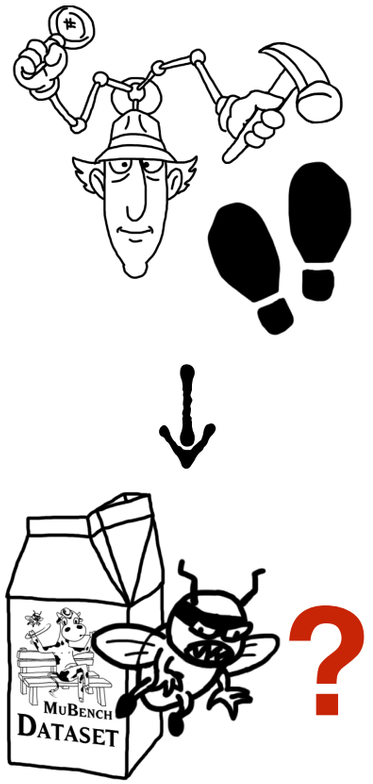
Ranking

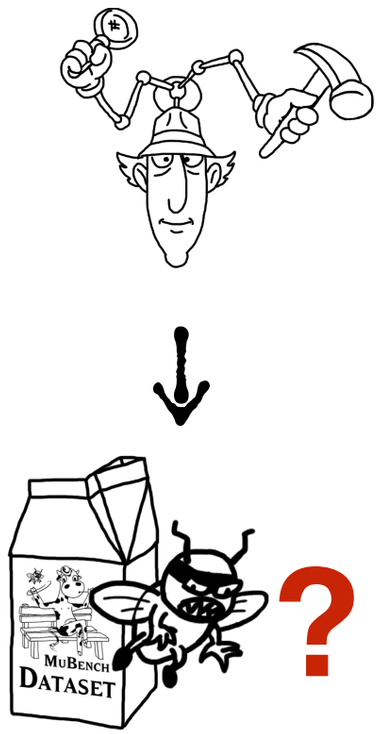
- Pattern Support
- Number of Pattern Violations
- Violation Support



1. Violation A
2. Violation B
3. Violation C
4. …







	Precision @20	Recall Upper Bound	Recall
Jadet	8.8%	16.9%	6.7%
GrouMiner	2.6%	51.2%	3.1%
Tikanga	8.9%	8.8%	7.6%
DMMC	7.5%	16.3%	10.7%
MuDetectXP	~~21.9%~~ 34.1%	72.5%	~~20.9%~~ 42.2%

Our strategic design approach led to significant improvements.

More usage examples again significantly improve performance.

There’s still room for improvement!