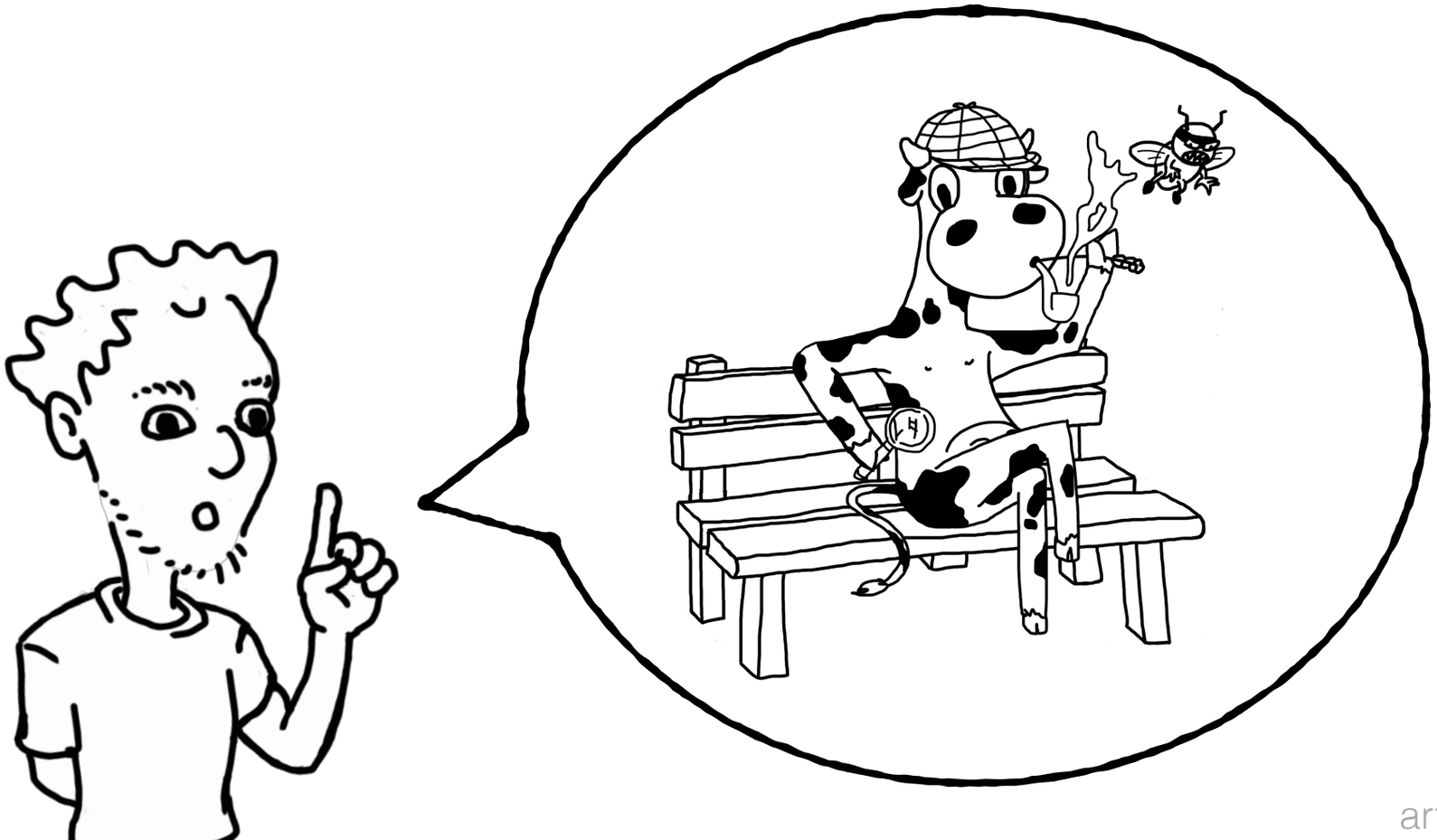
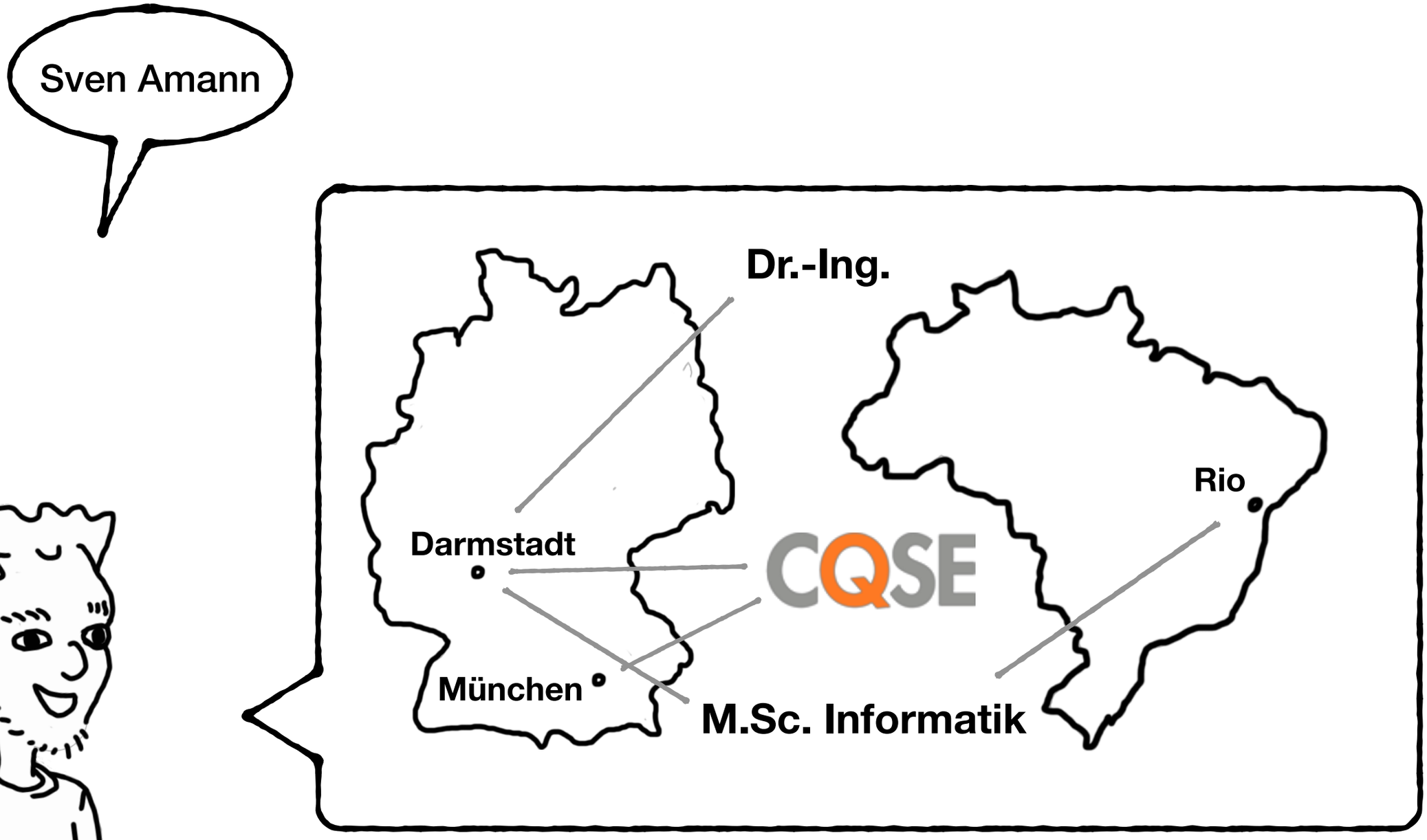
MUDetect

Nie mehr eine API falsch verwenden?

Entwicklertag Frankfurt 2019 - 21. Februar



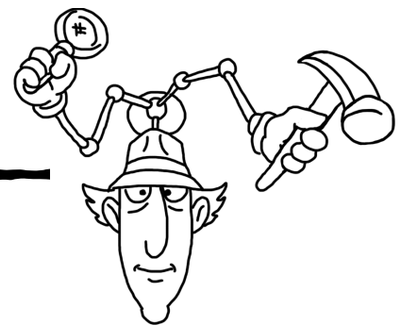


API-Misuse Detektion

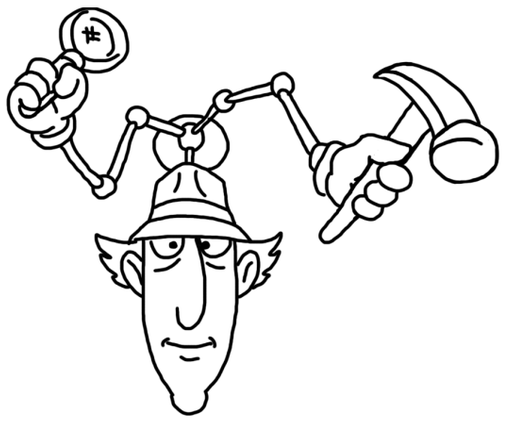
```
Collection<String> files = …;

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

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

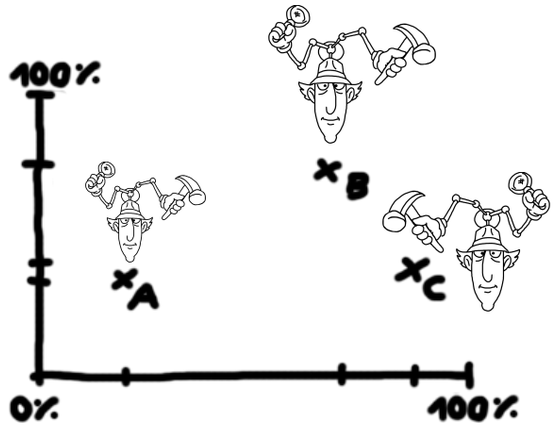
```
Collection<String> files = …;

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

Verifikation (Detektion)

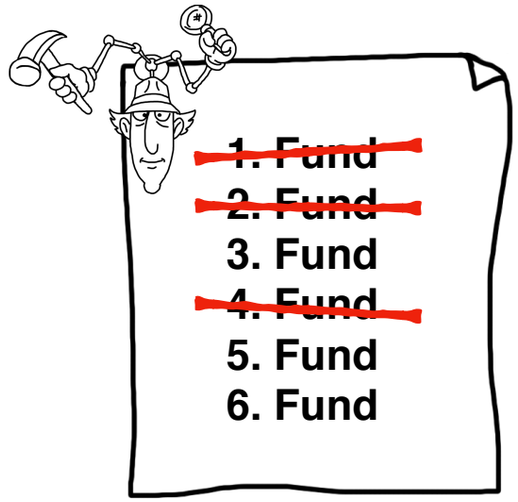
	Statisch	Dynamisch
Automatisch: Statisch	 **API-Misuse Detektoren**	
Automatisch: Dynamisch		
Manuell	**Specification Verifiers**	

Spezifikationen (Patterns)



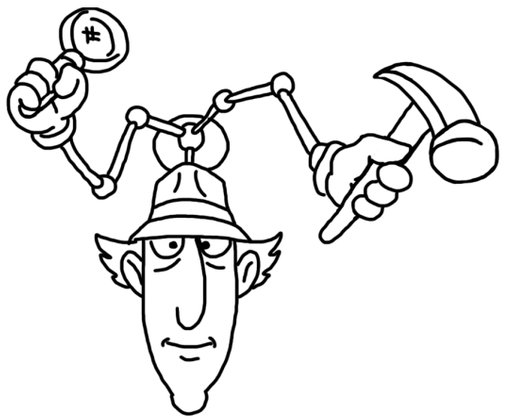


PR-Miner
Chronicler
Colibri/ML
Jadet
RGJ07
LKL08
Alattin
AX09
Car-Miner
GROUMiner
OCD
DMMC
SpecCheck
RRFinder
Tikanga
PJAG12
PG12
DroidAssist



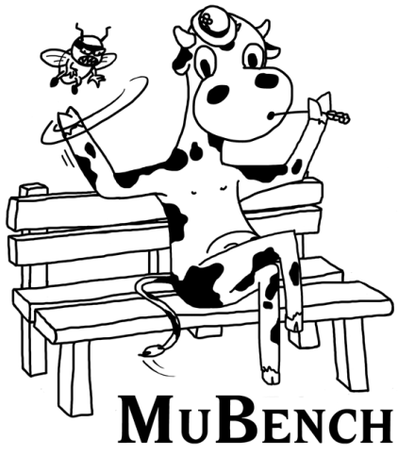
Genauigkeit in Top-X Funden

$6.5\% \leq p \leq 100\%$



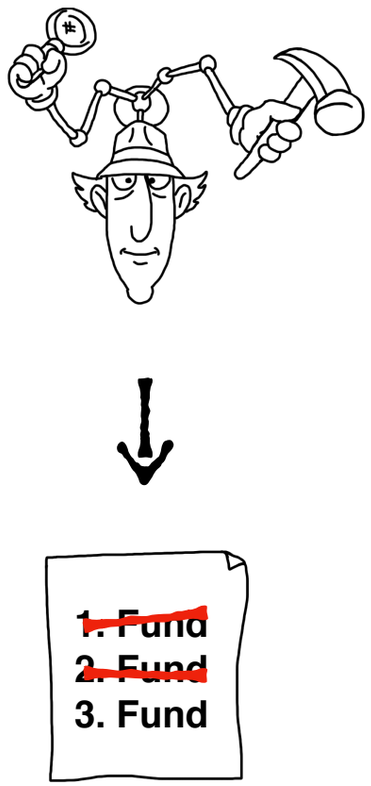
	Fehlende Methodenaufrufe	Redundante Methodenaufrufe	Fehlende null-Checks	Fehlende Zustandschecks	Fehlende Synchronisation	Fehlende Kontext-Checks	Redundante Checks	Fehlendes Fehlerbehandlung	Redundante Fehlerbehandlung	Fehlende Iteration	Redundante Iteration
PR-Miner	●	○	○	○	○	○	○	○	○	○	○
Chronicler	●	○	○	○	○	○	○	○	○	○	○
Colibri/ML	●	○	○	○	○	○	○	○	○	○	○
Jadet	●	○	○	○	○	○	○	○	○	◐	○
RGJ07	◐	○	●	●	○	○	○	○	○	○	○
LKL08	●	○	○	○	○	○	○	○	○	○	○
Alattin	◐	○	●	◐	○	○	○	○	○	○	○
AX09	◐	○	◐	◐	○	○	○	●	○	○	○
Car-Miner	◐	○	○	○	○	○	○	●	○	○	○
GROUMiner	●	○	◐	◐	○	○	○	○	○	◐	○
OCD	●	○	○	○	○	○	○	○	○	◐	○
DMMC	●	○	○	○	○	○	○	○	○	○	○
SpecCheck	●	○	○	○	○	○	○	○	○	○	○
RRFinder	●	○	○	○	○	○	○	○	○	○	○
Tikanga	●	○	○	○	○	○	○	○	○	◐	○
PJAG12	●	●	○	◐	○	○	○	○	○	○	○
PG12	◐	◐	○	◐	○	○	○	○	○	○	○
DroidAssist	●	●	○	○	○	○	○	○	○	○	○

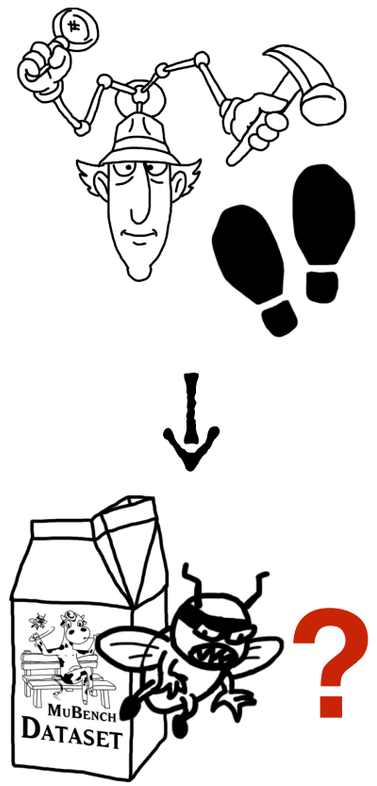
TSE'17

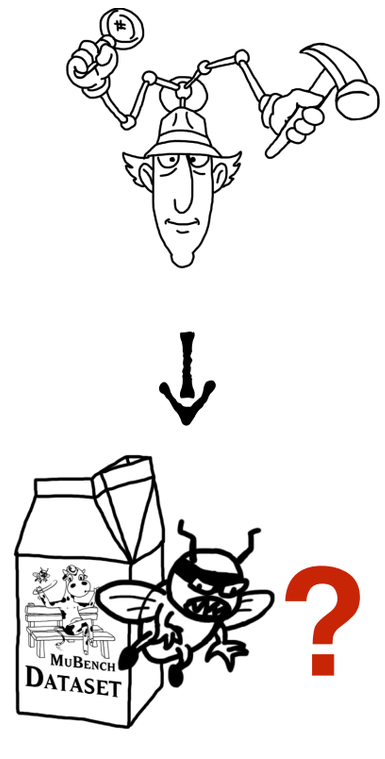


MuBench

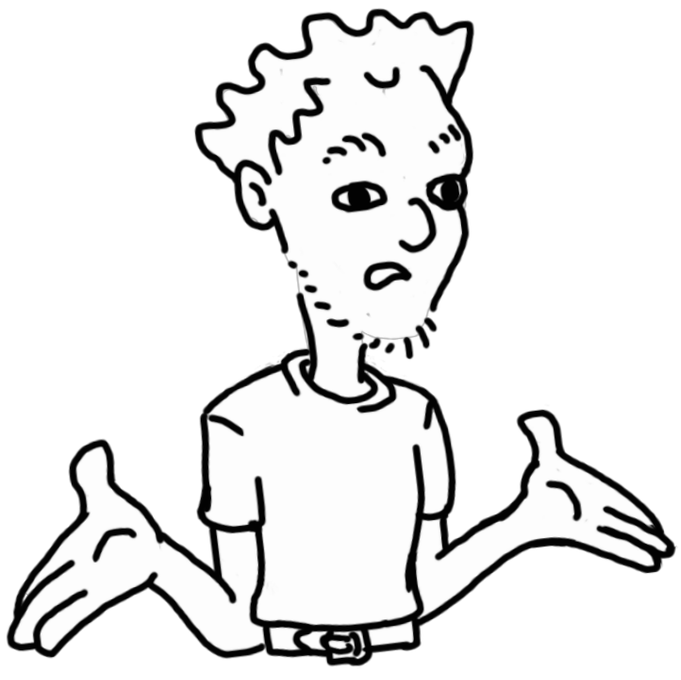
github.com/stg-tud/MUBench/







	Genauigkeit (Top 20)	Max. Trefferquote	Trefferquote
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 Misuse verursacht
~10% aller Software Bugs.

Detektoren finden nur
wenige Arten von Misuses.

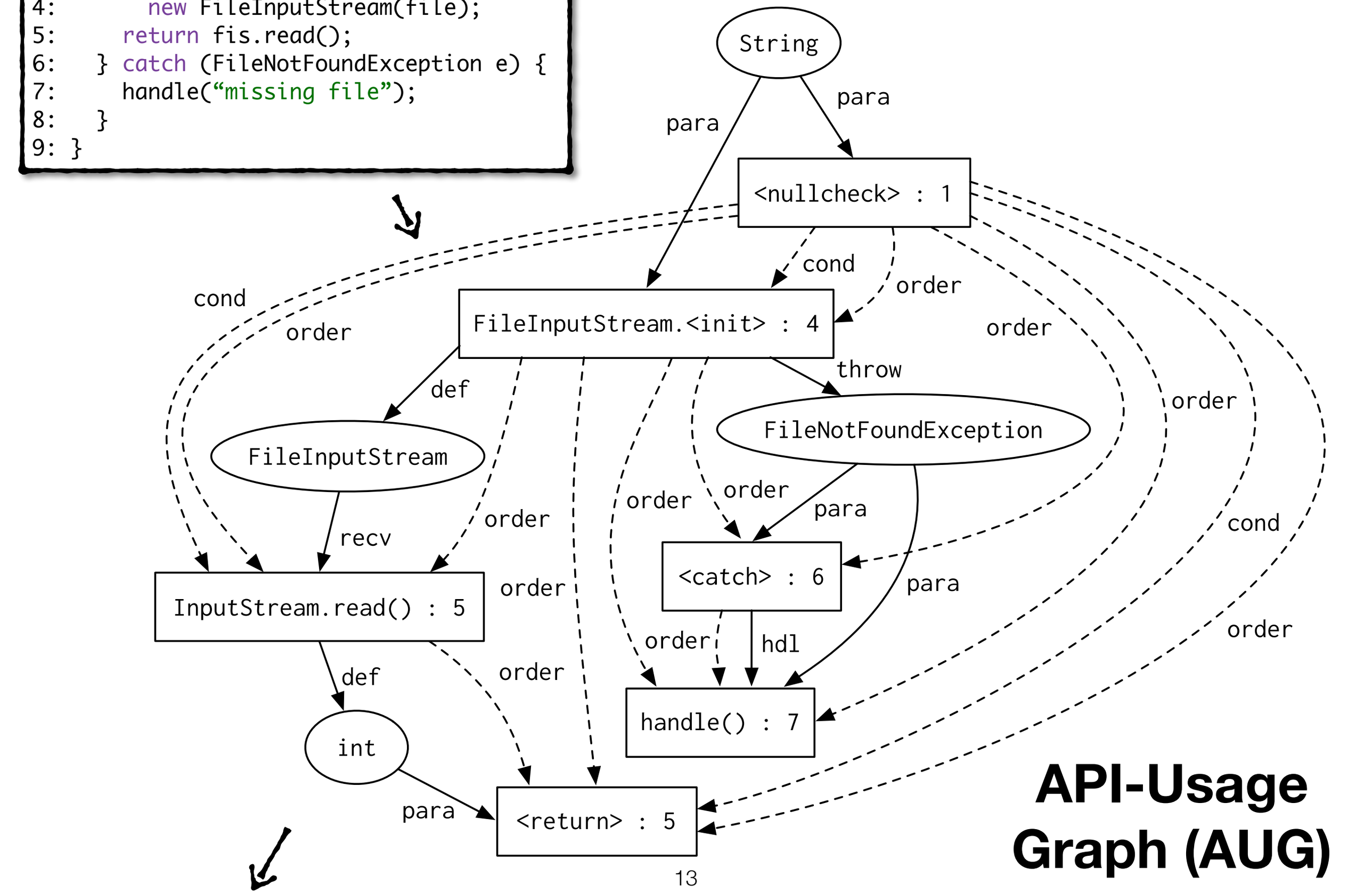
Detektoren haben niedrige
Genauigkeit und Trefferquote.

MuDetect
Ein kleiner Schritt…

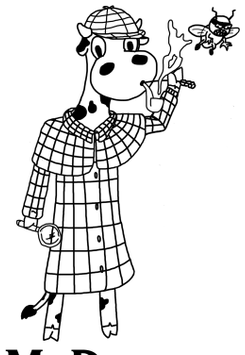
https://github.com/stg-tud/MUDetect/


```
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: }
```

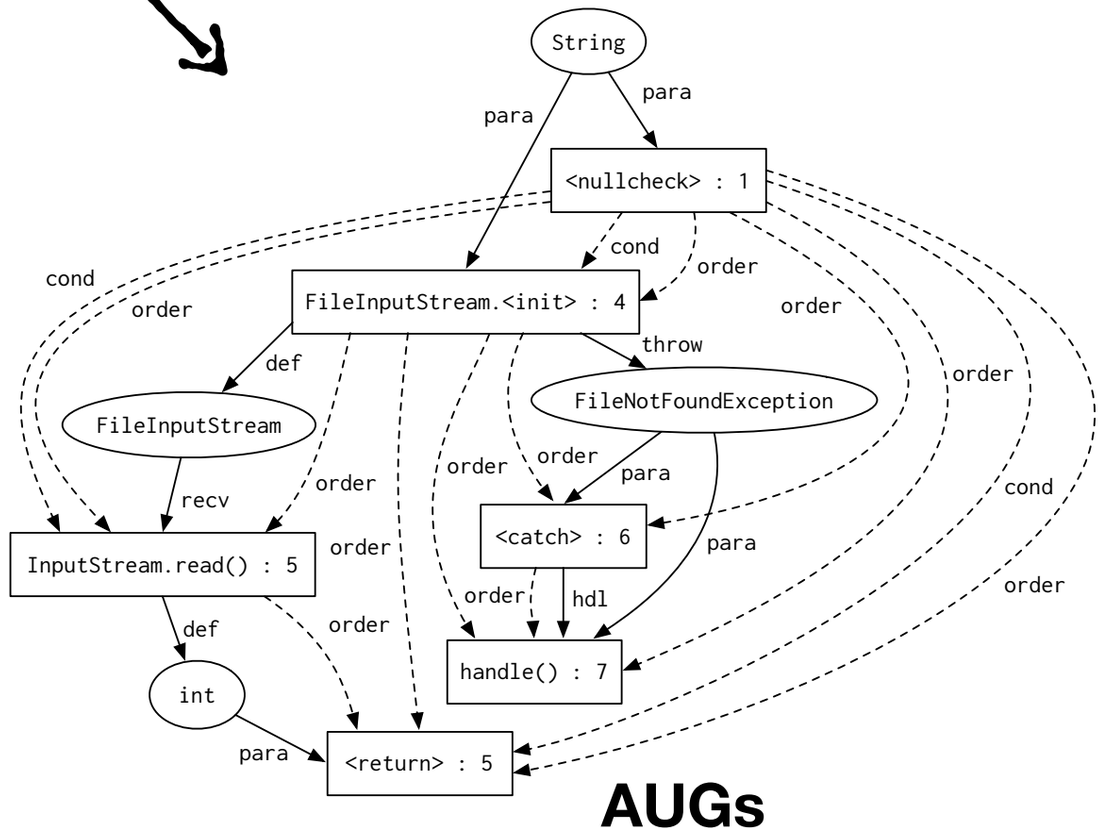
```
1: if (file != null) {
2:   try {
3:     FileInputStream fis =
4:       new FileInputStream(file);
5:     return fis.read();
6:   } catch (FileNotFoundException e) {
7:     handle(“missing file”);
8:   }
9: }
```



API-Usage Graph (AUG)



MuDetect



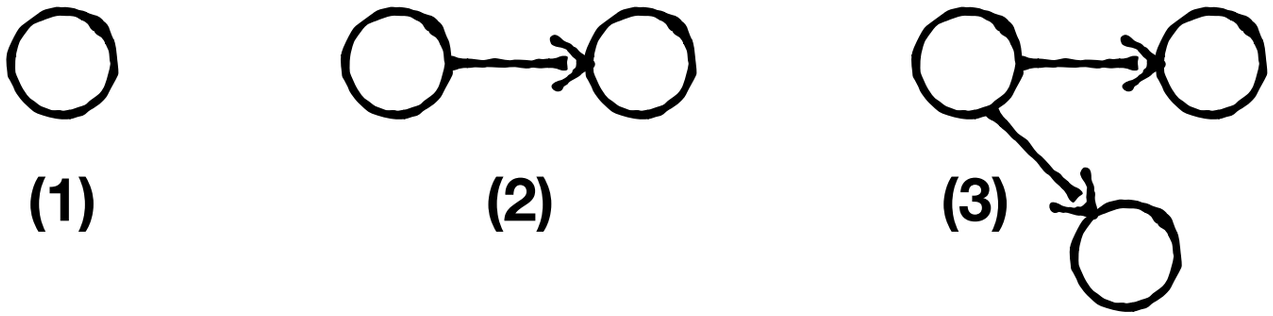
AUGs

Training

MuDetect

Training

Apriori-based Mining



MuDetect

Training

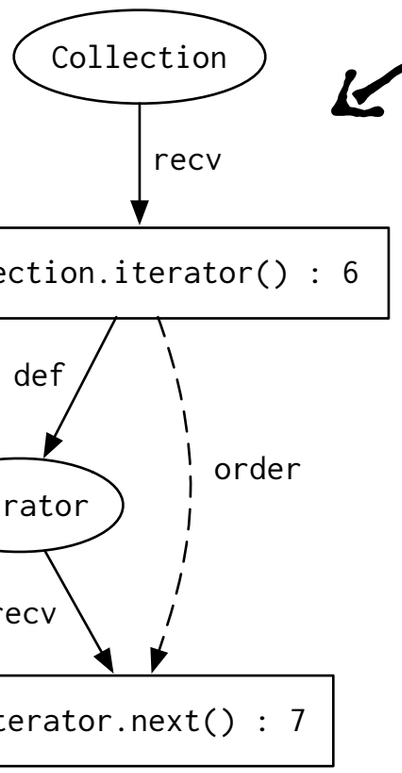
Patterns

MuDetect

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

Zielcode

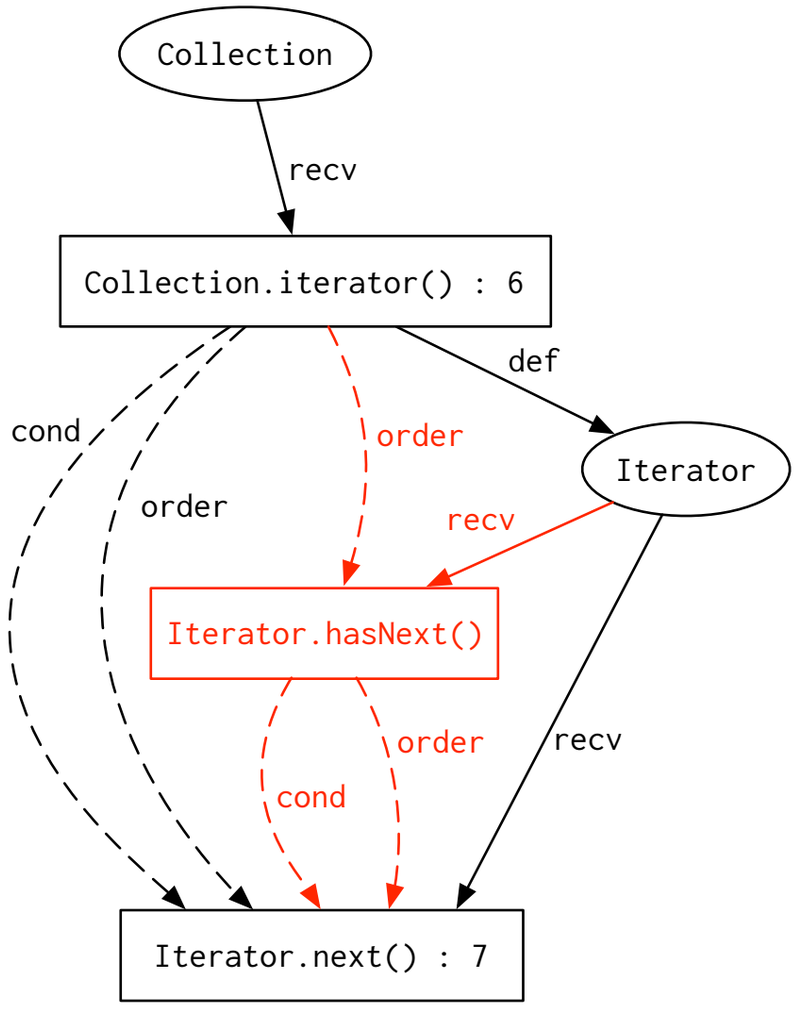
Patterns



AUGs

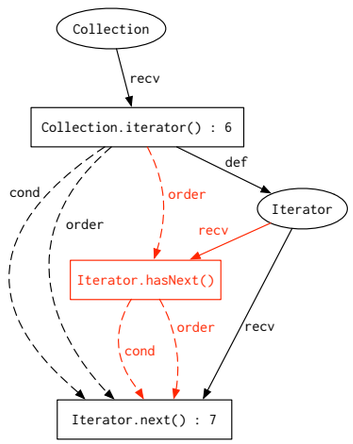
MuDetect

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



Abweichungen

MuDetect

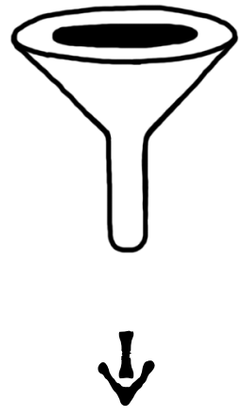


Abweichungen

Patterns

Filtern

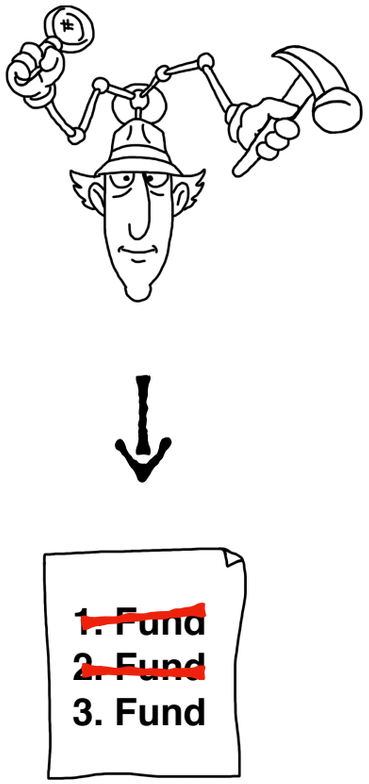
- Instanzen alternativer Patterns
- Alternative Abweichungen

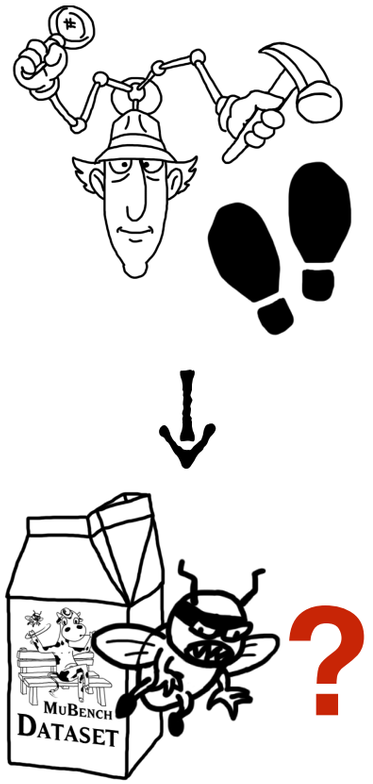


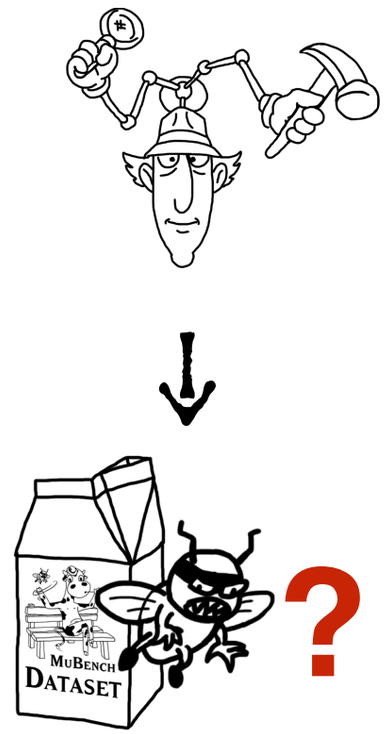
Sortierung

- Pattern Support
- Anzahl der Abweichungen pro Pattern
- Support der Abweichung

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







	Genauigkeit (Top 20)	Max. Trefferquote	Trefferquote
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%

TestNG

```
if (!m.getTestClass().getName().equals(currentClass)) {
  if (!"".equals(currentClass)) {
    xsb.pop(D);
  }
  xsb.push(D, C, "chronological-class");
  xsb.addRequired(D, m.getTestClass().getName(), C,
                  "chronological-class-name");
  currentClass = m.getTestClass().getName();
}
```

```
...
<div panel-name="chronological-MySuiteInVM" class="panel">
  ...
  <div class="main-panel-content rounded-window-bottom">
    <div class="chronological-class">
      ...
    </div>
  </div>
</div>
...
```

Google Closure Compiler

```
private Node constructAddOrStringNode(
    Iterator<CharSequence> partsIterator, Node argListNode) {

  CharSequence part = part                    t();
  ...
}
```

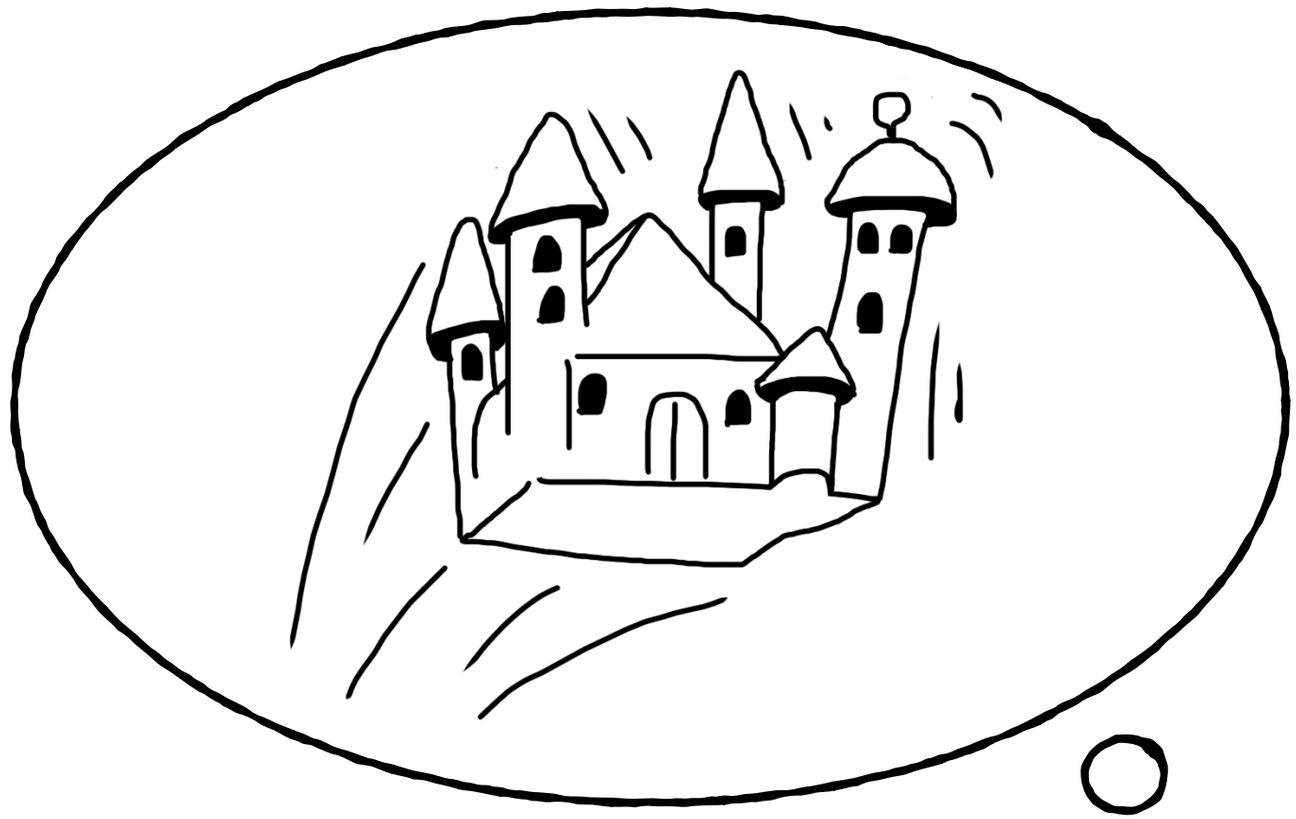
Google Closure Compiler

```
StringBuffer sb = ...
Collection<...> refersTo = ...

Iterator<...> fromIter = refersTo.iterator();
while (fromIter.hasNext()) {
  sb.append(nameLink(
    fromIter.next().getDestination().getValue().name));

  if (fromIter.hasNext()) {
    sb.append(", ");
  }
}
```

MuDetect

MuDetect ist signifikant besser als ältere Detektoren.

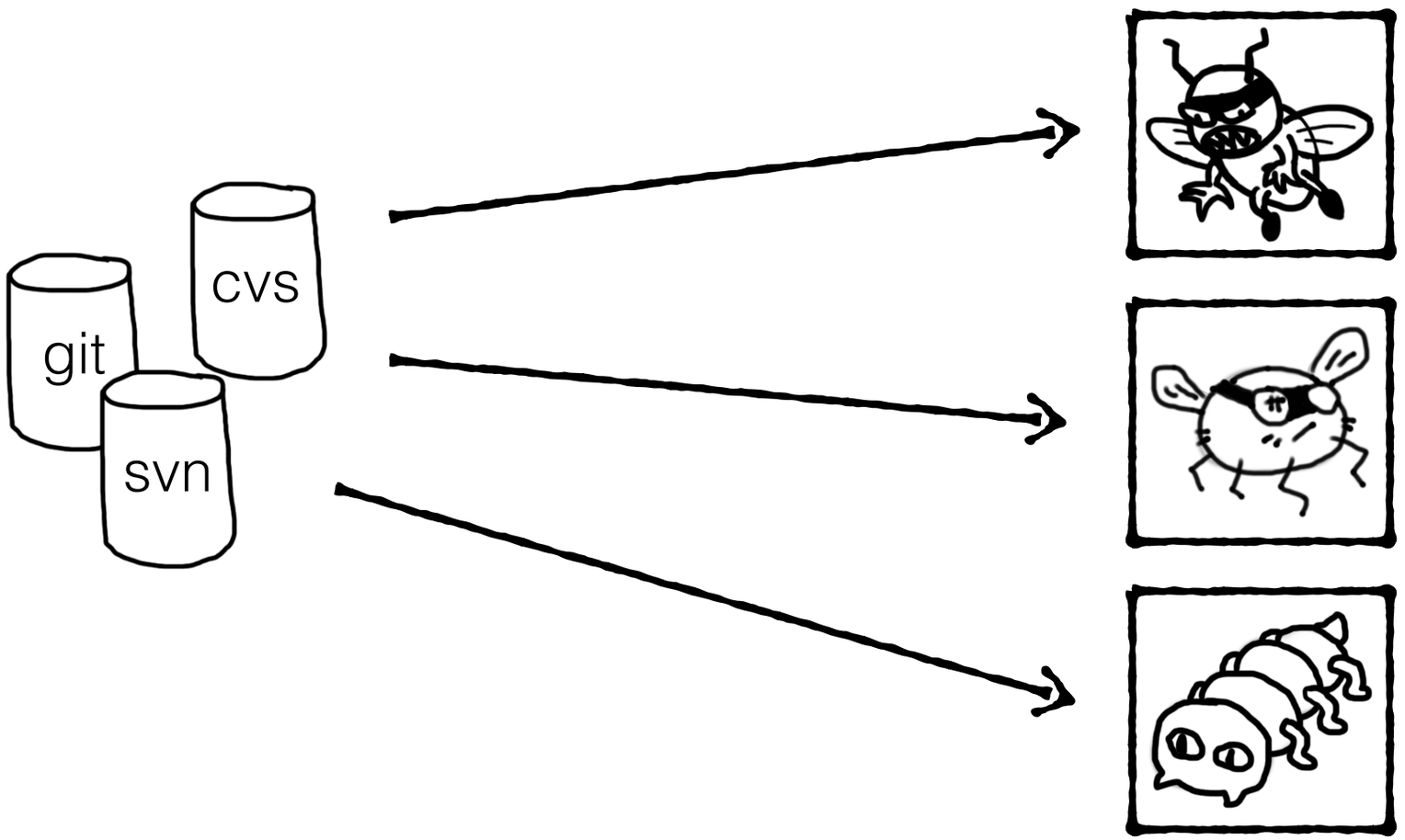
Praxistauglich?



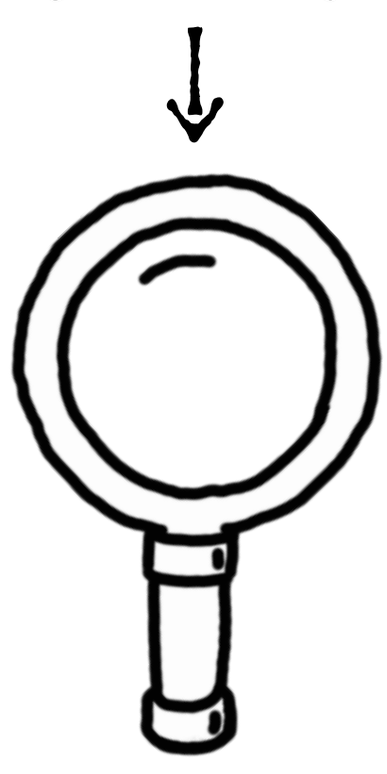
Akademische Luftschlösser?

https://blog.mozilla.org/futurereleases/2019/02/12/

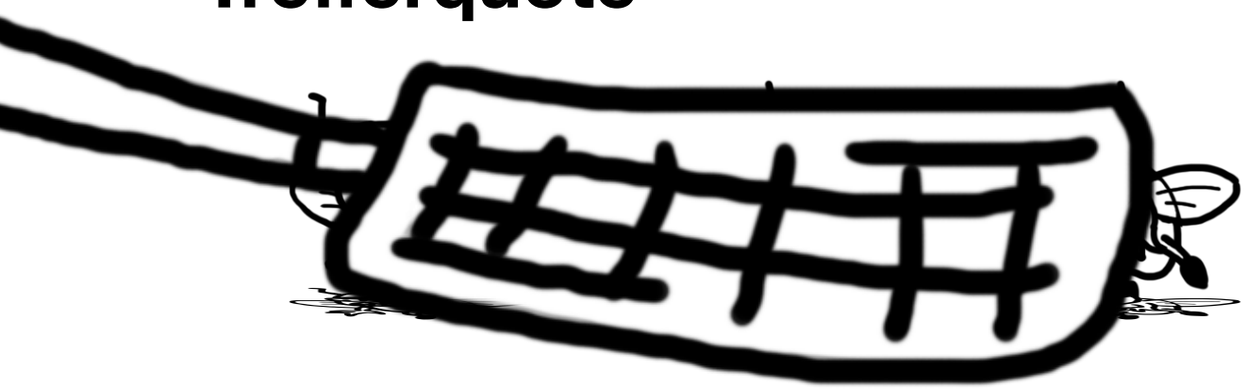
Clever Commit von Ubisoft: https://youtu.be/I5C4FUvDyCc




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

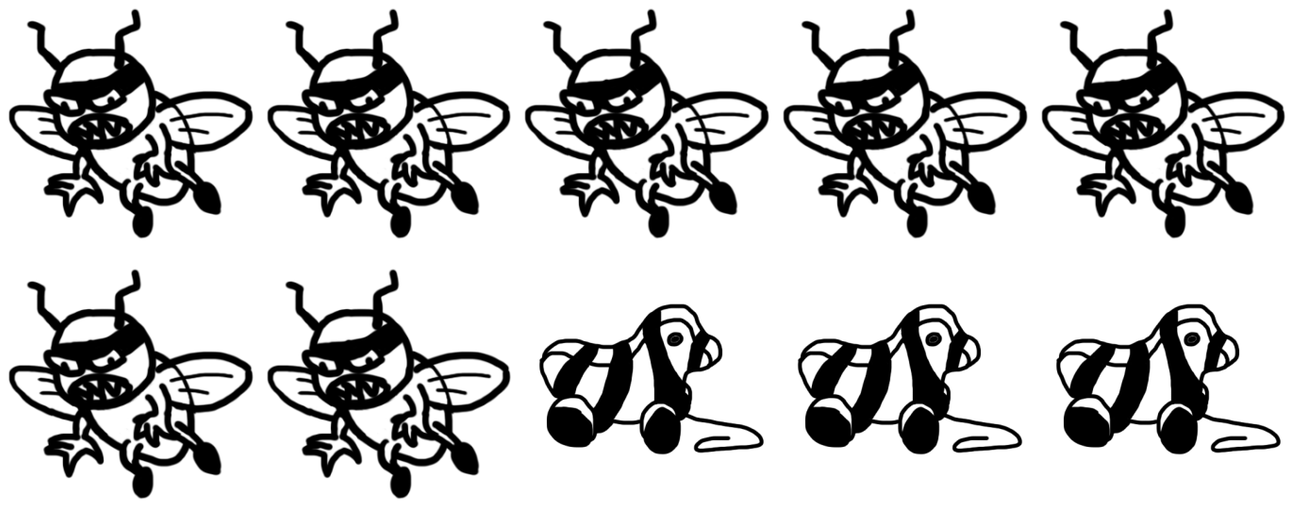


Trefferquote



60%

Genauigkeit



70%

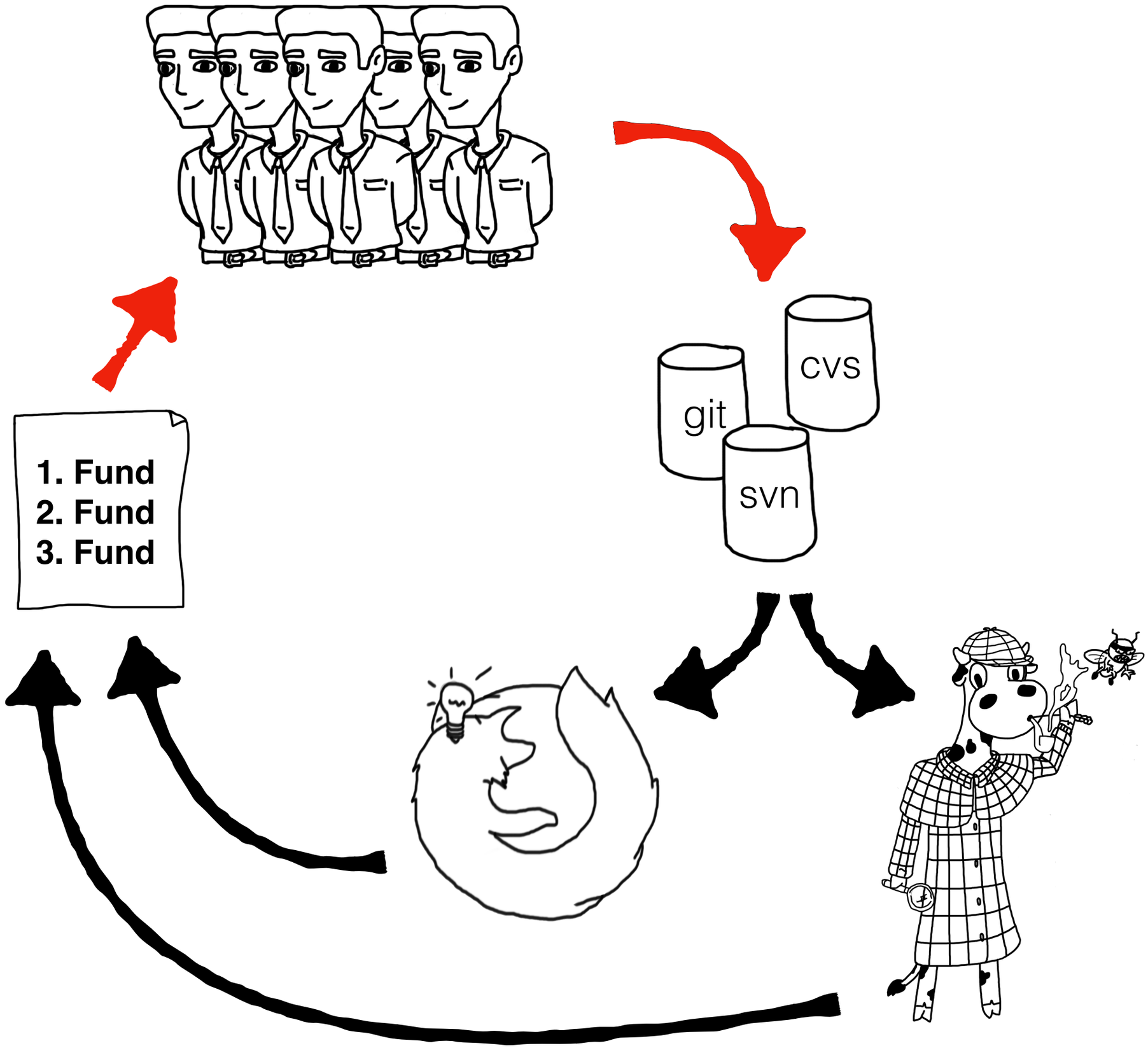
Selbstlernende Detektoren werden praktisch eingesetzt.

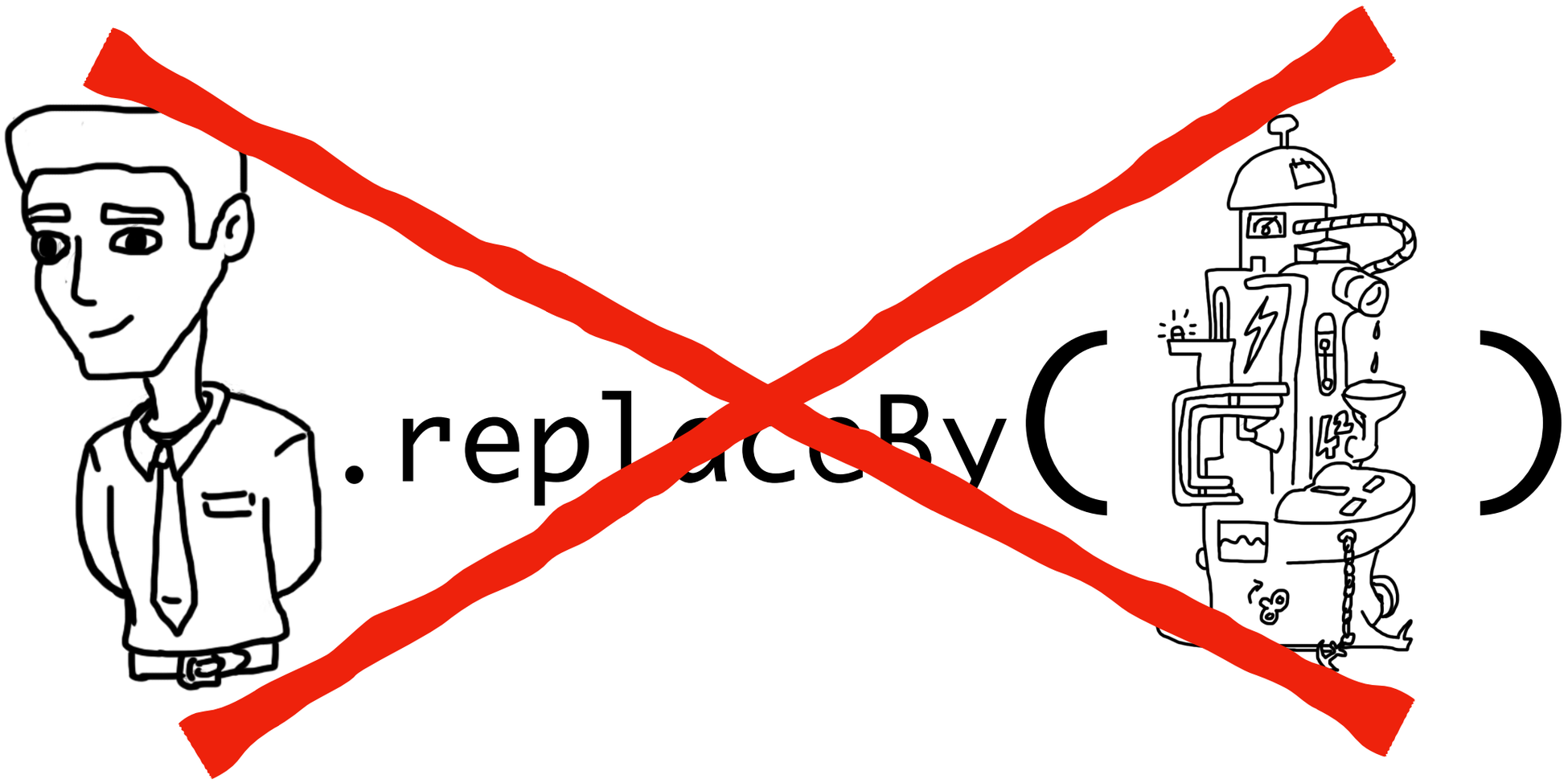
Spart 20% Entwicklungsaufwand

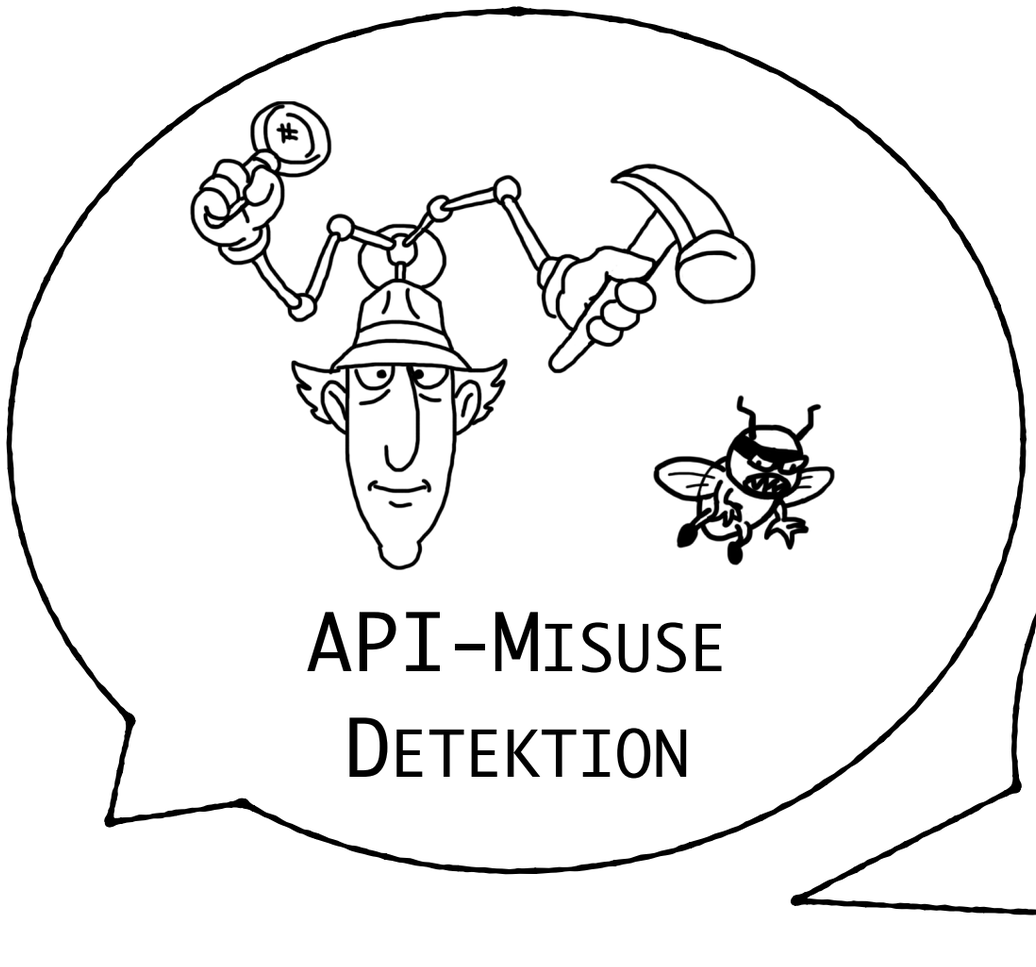
.replaceBy(

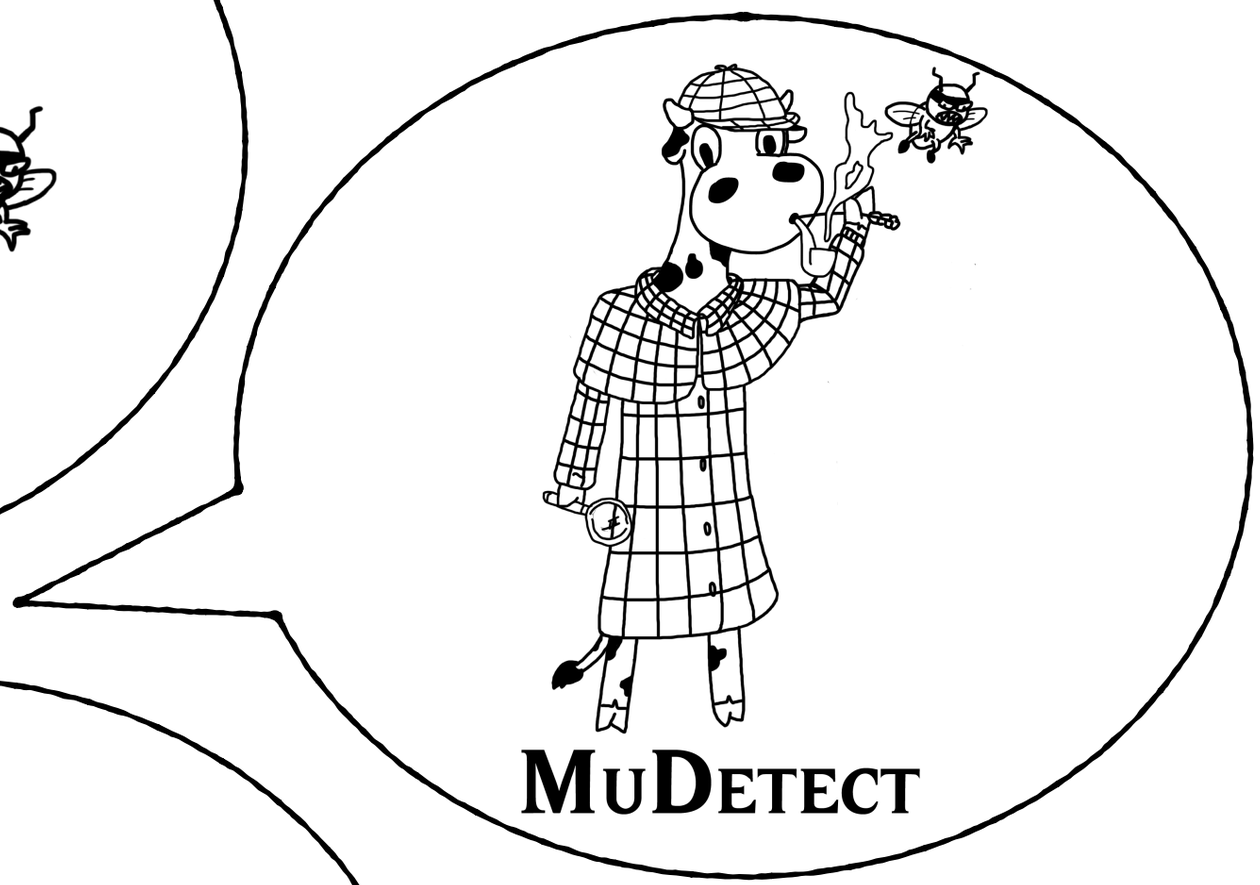
)

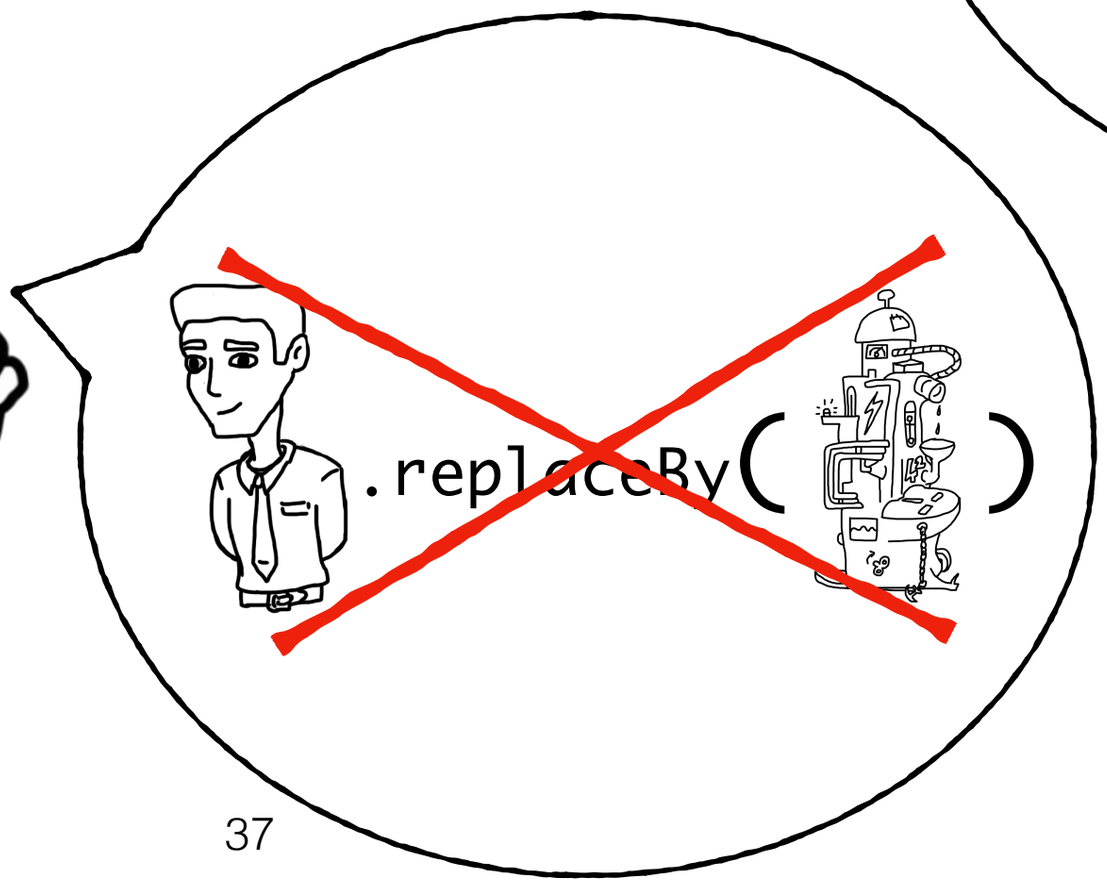
Sind Entwickler ersetzbar?











MUDetect

Nie mehr eine API falsch verwenden?