What if...

[y]our code were data?

@JurgenVinju

CODAM masterclass

Amsterdam, Sep 22th, 2021

Audience Expertise

- Assembly? x86, ARM, MIPS
- Procedural? C, Pascal, Basic
- Object-oriented? C++, Java, C#, Smalltalk
- Functional? Haskell, ML, Idris
- Multi-paradigm? Python, VB, PHP, Scala

Audience Expertise

- Expert $>$ 10 years of programming
- Professional $>$ 5 years of programming
- Aspirant $>$ 1 year of programming
- Beginner $>$ 0 years of programming

Audience Expertise

- (Serious) Games
- High-tech Systems
- Finance & Admin
- Mobile applications
- Web
- Healthcare
- Everything!

Today

1. *Designing code* is interesting and fun
   - *Analyzing code* is more important
   - {sh,c,w}ould be interesting and fun too
2. Analyzing code should be automated:
   - use the generic analyses of your IDE
   - script your own analyses with **Rascal**

Fascinating Code

- Art of reading and writing source code
  - Creative imagination

- Code both *enables* and *limits* everything
  - Machine control
  - Execution of laws and regulations
  - Social interaction

- What is (good) code?
  - What does it do? not do?
  - How can we change or extend it?
  - Just *read* it... right?

Banksy


Muniz


Haring

Programming:

the **joy** of *creating* and *maintaining* **code**,

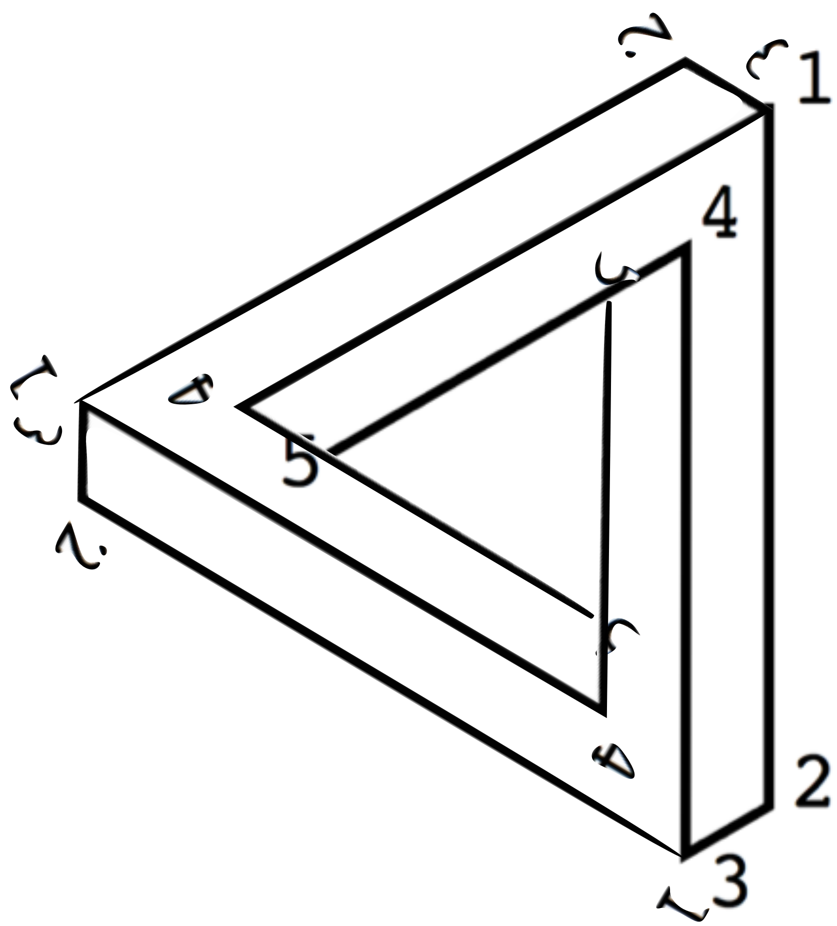
with the **responsibilility**

to "get it right"

for all the **people** that are involved

Belastingdienst

Predicting a small PS program




```
%! PS EPS
3{25 34 moveto
25 -34 lineto
17 -38.2 lineto
17 20 lineto
-17.6 0 lineto
120 rotate
}repeat stroke showpage
```

[Reutersvärd/Penrose -> Escher -> C.G. van der Laan]

`ls`; a small but old program

jurgenv — -bash

~/git/rascal-core — -bash | ~/git/rascal — -bash | ~ — -bash | ...ascal-language-servers — -bash | ...ascal-language-servers — -bash

```
~ $ ls
A.rsc              Notes              callGraph.png
ASYNC              Papers             concreteSyntax.txt
Applications       Photos             eclipse-workspace
Business           Pictures           git
Calibre Library    Presentations      impulse
DDD                Projects           mongo
DOS Games          Public             notesSyntax.md
David              Scripts            ossmeter
Desktop            Sites              rascal-0.19.4-SNAPSHOT.jar
Documents          Tools              rascal-test-project
Downloads          Workspaces         test.py
Dropbox            app.icns           testscript.json
Google Drive       bin                tmp
Library            callGraph.dot      uninstall.sh
Movies             callGraph.pdf
Music              callGraph.pfg
~ $
```

The source code of "ls"

3894 lines

367 ifs

174 cases

Real code is big

File list
5000 lines of code

Voting
70.000 line of code

MRI scanner
1M lines of code

Bank
20 M lines of code

Google
2 billion lines of code

Panta rei

code must change over time

Understanding code <u>is not required</u> to make changes

accidental code with accidental growth

Large code that is hard to understand

Analyzing absurdities

- Code is interesting: complex and large
- Code always has to change
- Look up: Lehman's Laws of Software Evolution (1974)
- Code {sometimes, often, always} does not make any sense (to us)
- Code maintenance costs are high: 15% of TCO per year (cumulative!)
- Code reading "manually" seems to be the default analysis method
- So now what?

["15%" is anecdotal]

Analyzing Code: Questions

- How does this algorithm work?
- Why do our users get NullPointerExceptions?
- *Why don't we get anything back from the database?*
- Which code depends on this component?
- Is this change architecturally compliant?
- What might break if I change this code?
- Why is this code so slow?
- Can this code cause injury or death?

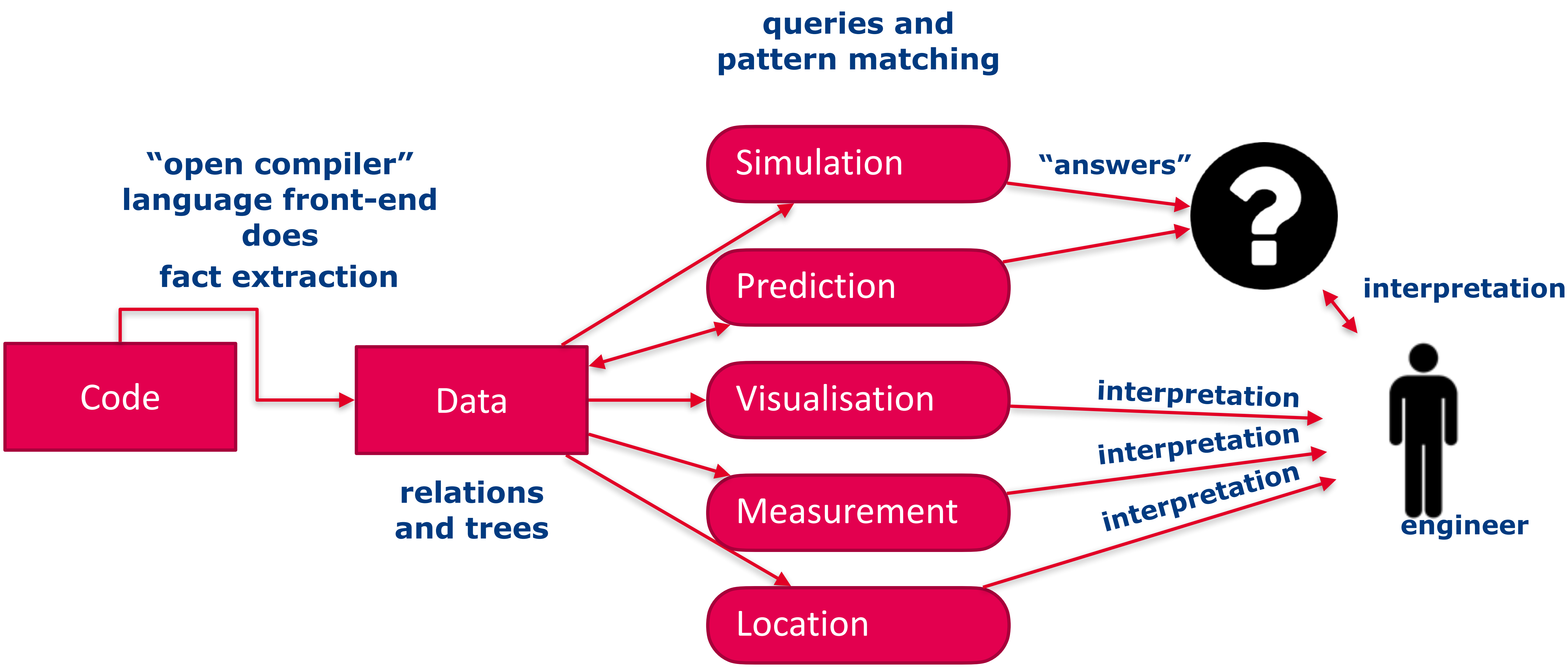
Analyzing Code: Use the Tools!

- **Interactive Debugger**: how does it work step-by-step
- **Memory Profiler**: what are memory bottlenecks?
- **CPU Profiler**: what are CPU/IO bottlenecks?
- **Editor** with language support (IDE):
  - jump-to-definition
  - implementations/overrides
  - type hierarchy
  - call "hierarchy"
  - refactoring tools: rename, pull-up, extract-method, ...
- **UML extractors**: what is the overall structure?

Analyzing Code: Yourself!

- What about the questions that do not have a tool?
  - .... err.... let's read the code?
  - ok, but only if all else fails
- **Script your own analyses: code is data**
  - Locate, Visualize, Transform
- Use your own, local, contextual, information:
  - "we have an NPE"
  - but "we always check input parameters for **null**"
  - so "find all methods that do not test a parameter for null"
- How? "Some understanding" + "Code as Data" + "Query"

Automated Code Analysis: Overview



Step 1. Reuse: language "front-ends" that make data out of code

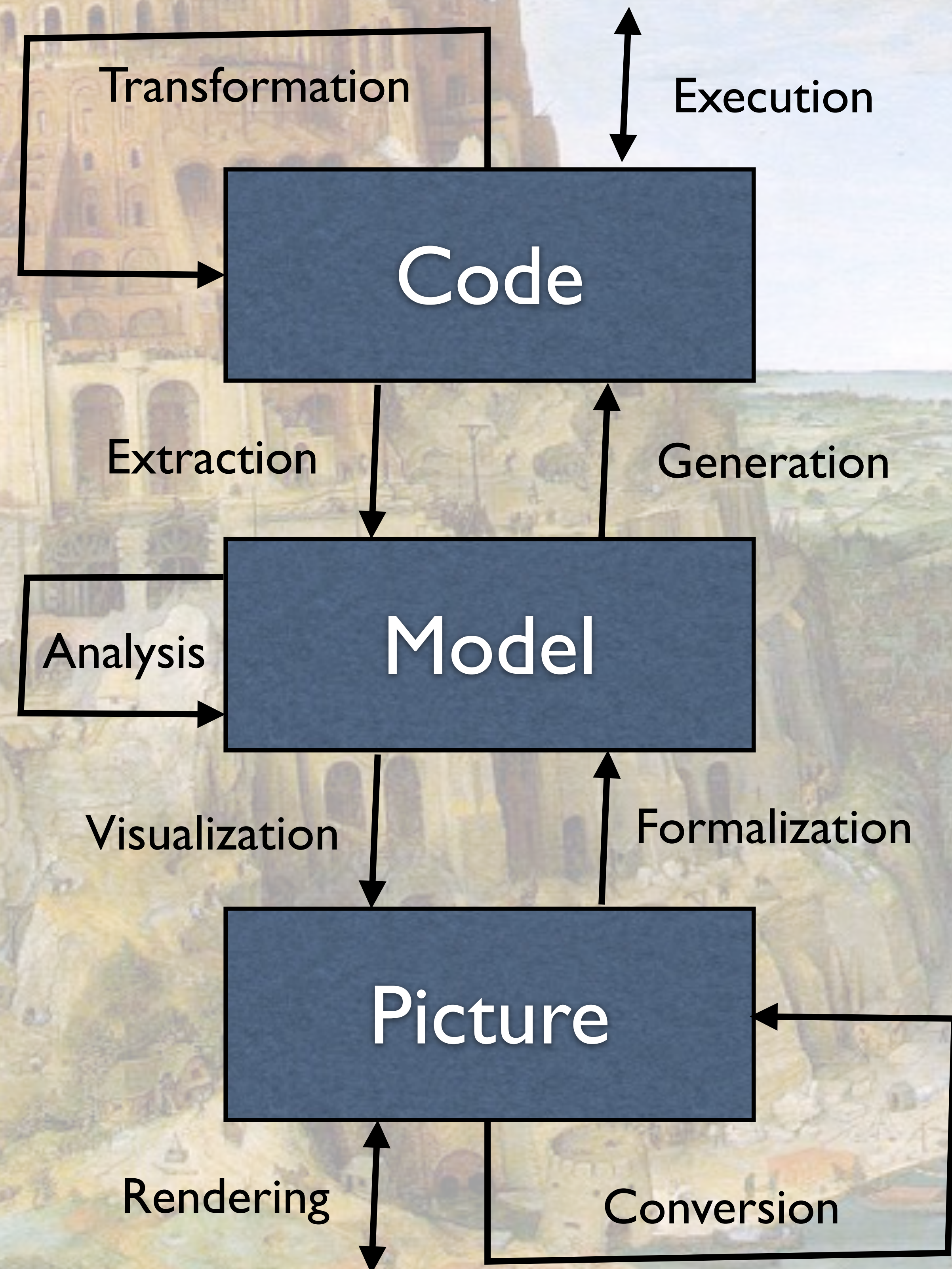
Step 2. Script: query that data

Step 3. (Optional) Script: visualize, transform code using (2)

Step 4. Manual: interpret result (2) and/or (3)


(Brueghel, Tower of Babel)

Rascal
is
a
DSL
for
meta
programming



(Brueghel, Tower of Babel)

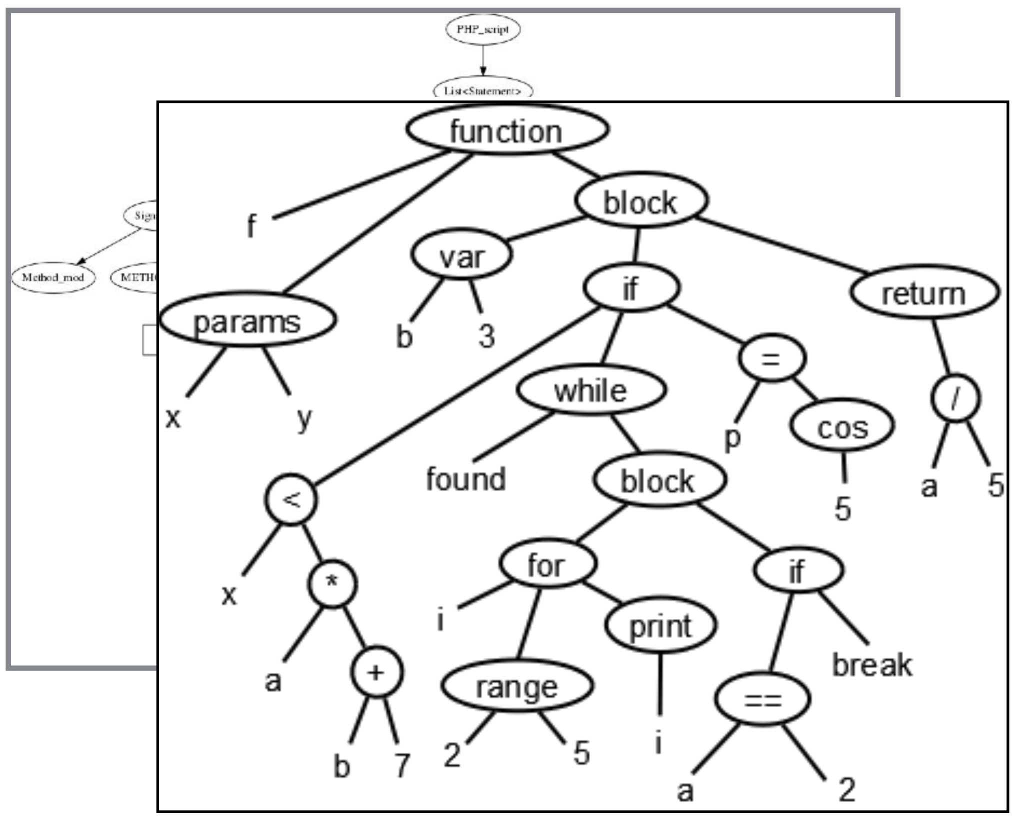
Rascal: metaprogramming language

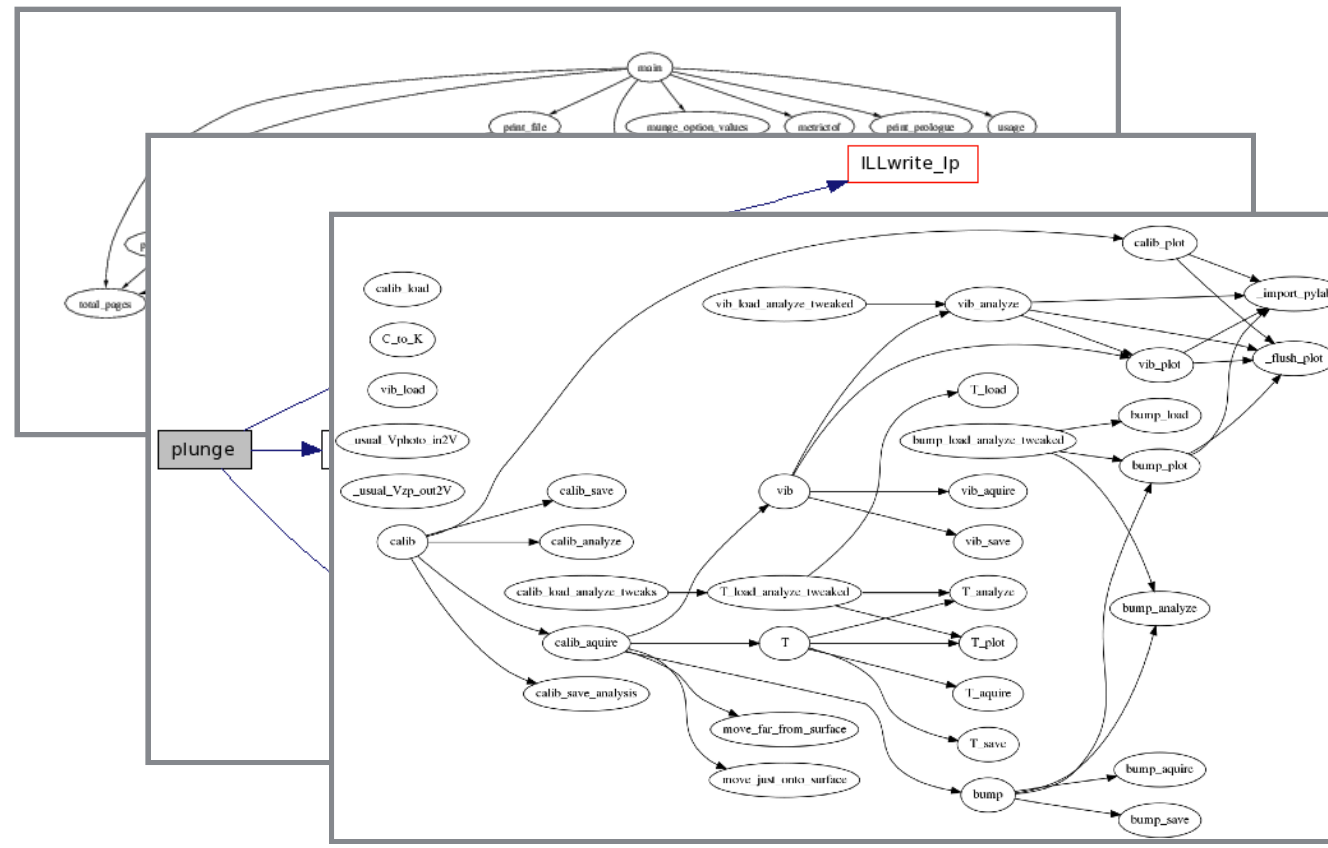
- “meta” means code is input and/or output of Rascal programs
- “programming” means that **you** can learn Rascal based on your GPL/SQL skills
- broad application area *where code is always data*:
  - model driven engineering: model-to-code, code-to-model, model-checking
  - domain specific languages: parsers, code generators, checkers, LSP based editors
  - reverse engineering: architecture reconstruction
  - (large scale) re-engineering: software renovation, rejuvenation
  - **(small scale) code query: software maintenance activities**
  - **refactoring: automated software transformation**
  - software analytics: code metrics, issues, versions, test results, ...

Data model

- **(Annotated) Trees**
  - abstract syntax trees (with qualified names and locations)
  - concrete syntax trees (with locations)
- **Relations (Tables)**
  - *definitions* (name x loc) and *uses* (loc x name)
  - *containment* (name x name)
  - *calls* (name x name)
  - *overrides*, *implementations*, *inheritance* (name x name)
  - ... etc.

Rascal “M3” data model





Language specific syntax trees

+

Language agnostic relational models (tables)

Source Locations

Source Locations (`loc`) link to any artefact.

loc	type
`file:///tmp/Hello.java`	Physical
`project://myProject/Hello.java`	Physical
`java+interface://myProject/java/util/List`	Logical
`java+method://myProject/java/util/List/contains(Object)`	Logical

Declarations

```
1.  interface Fruit {
2.      boolean eat();
3.  }
4.
5.  class Apple implements Fruit {
6.      boolean eat() {
7.          peal();
8.          consume();
9.      }
10.
11.     void peal() { … }
12. }
```

name x loc

java+interface:///Fruit	file:///MyFile.java(1,2)
java+class:///Apple	file:///MyFile.java(5,12)
java+method:///Fruit/eat	file:///MyFile.java(2,2)
java+method:///Apple/eat	file:///MyFile.java(6,9)
…	…

Containment

```
interface Fruit {
    boolean eat();
}

class Apple implements Fruit {
    boolean eat() {
        peal();
        consume();
    }

    void peal() { … }
}
```

name x name

java+interface:///Fruit	java+method:///Fruit/eat
java+class:///Apple	java+method:///Apple/eat
java+class:///Apple	java+method:///Apple/peal
…	…

Implements

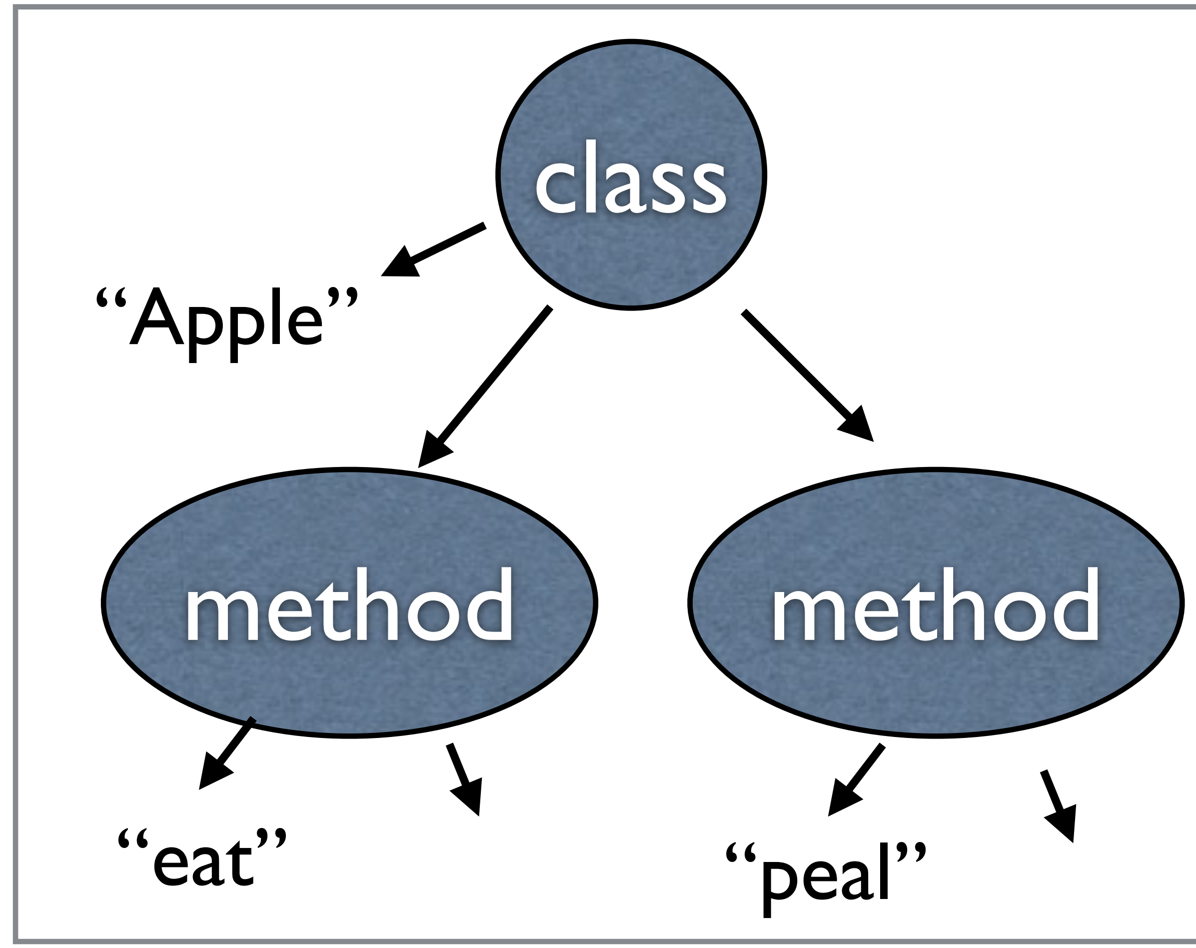
```
interface Fruit {
    boolean eat();
}

class Apple implements Fruit {
    boolean eat() {
        peal();
        consume();
    }

    void peal() { ... }
}
```

name x name

java+class:///Apple	java+interface:///Fruit
...	...

Syntax Tree: *nesting* made explicit

```
interface Fruit {
    boolean eat();
}

class Apple implements Fruit {
    boolean eat() {
        peal();
        consume();
    }

    void peal() { … }
}
```




```
class(“Apple”,
    [
        method(boolean(), “eat”, [ ], [ … ])
        method(void(), “peal”, [ ], [ … ])
    ]
)
```

Syntax trees are the {XML,YAML,JSON} of source code

Intermezzo: analysis accuracy

- Code analyses can be wrong in subtle ways
- So a small script could give us wrong answers
- And give us a false sense of security
- So before we go on, a (very) small lecture on "code analysis accuracy"

example: "find all methods that do not test a parameter for null"

A null-check idiom

```
int order(Fruit x, int amount) {
    assert x != null;

    table.put(x, amount);
}
```

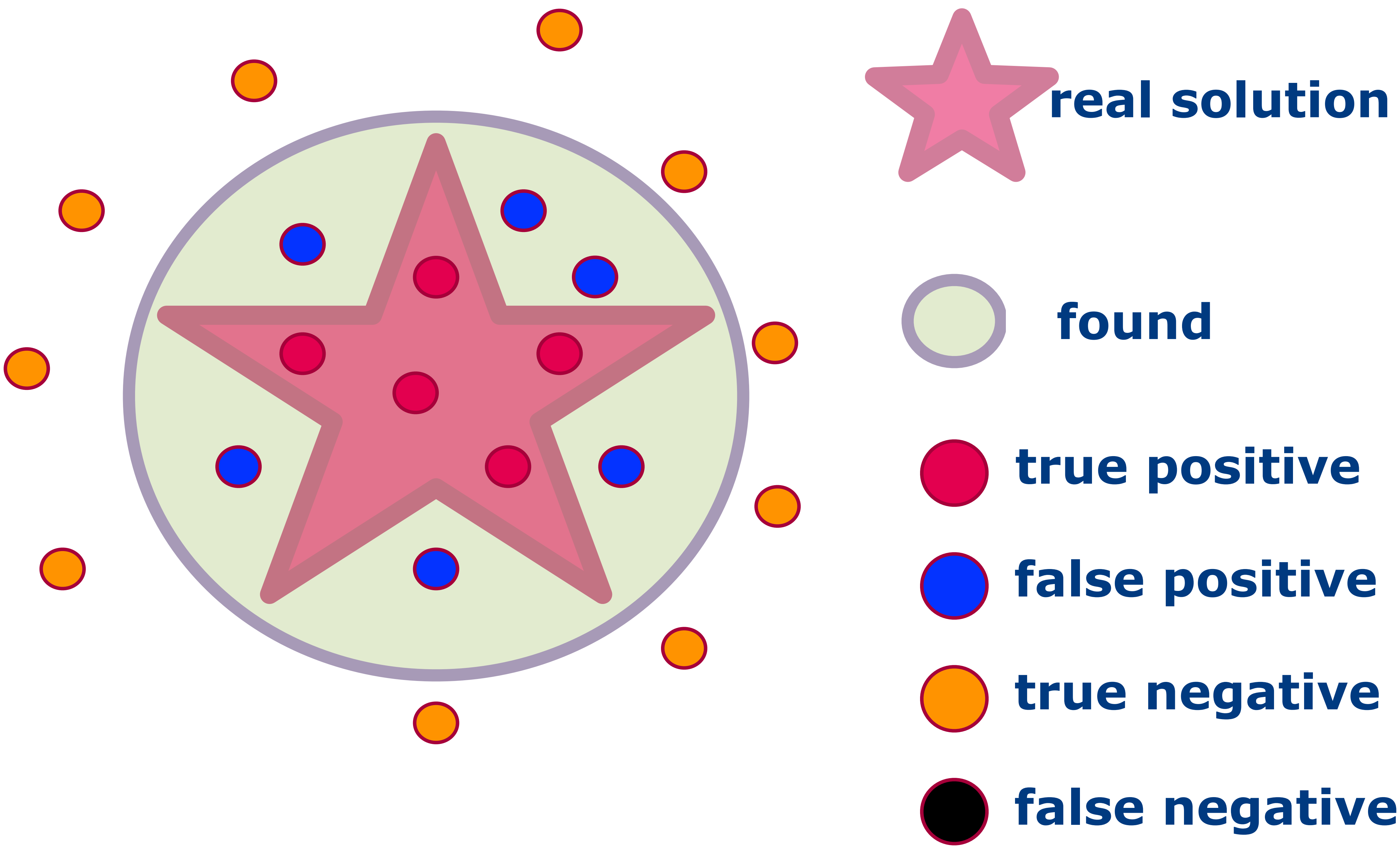
`x` should not be null if used as a key in the table

`amount` can not be null because it is an `int`

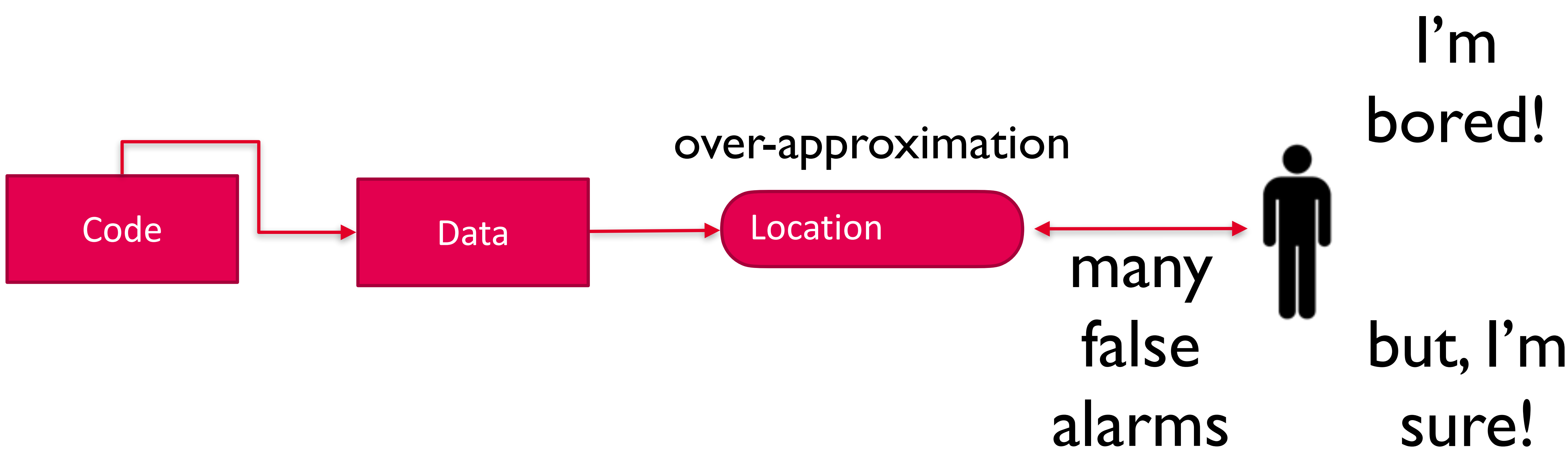
Safe (and boring) over approximation

"all parameters of all methods that are not asserted != null"

=all unchecked object parameters **and** all {integer, boolean, float} parameters that did not need to be checked



The pro's and con's of over-approximation

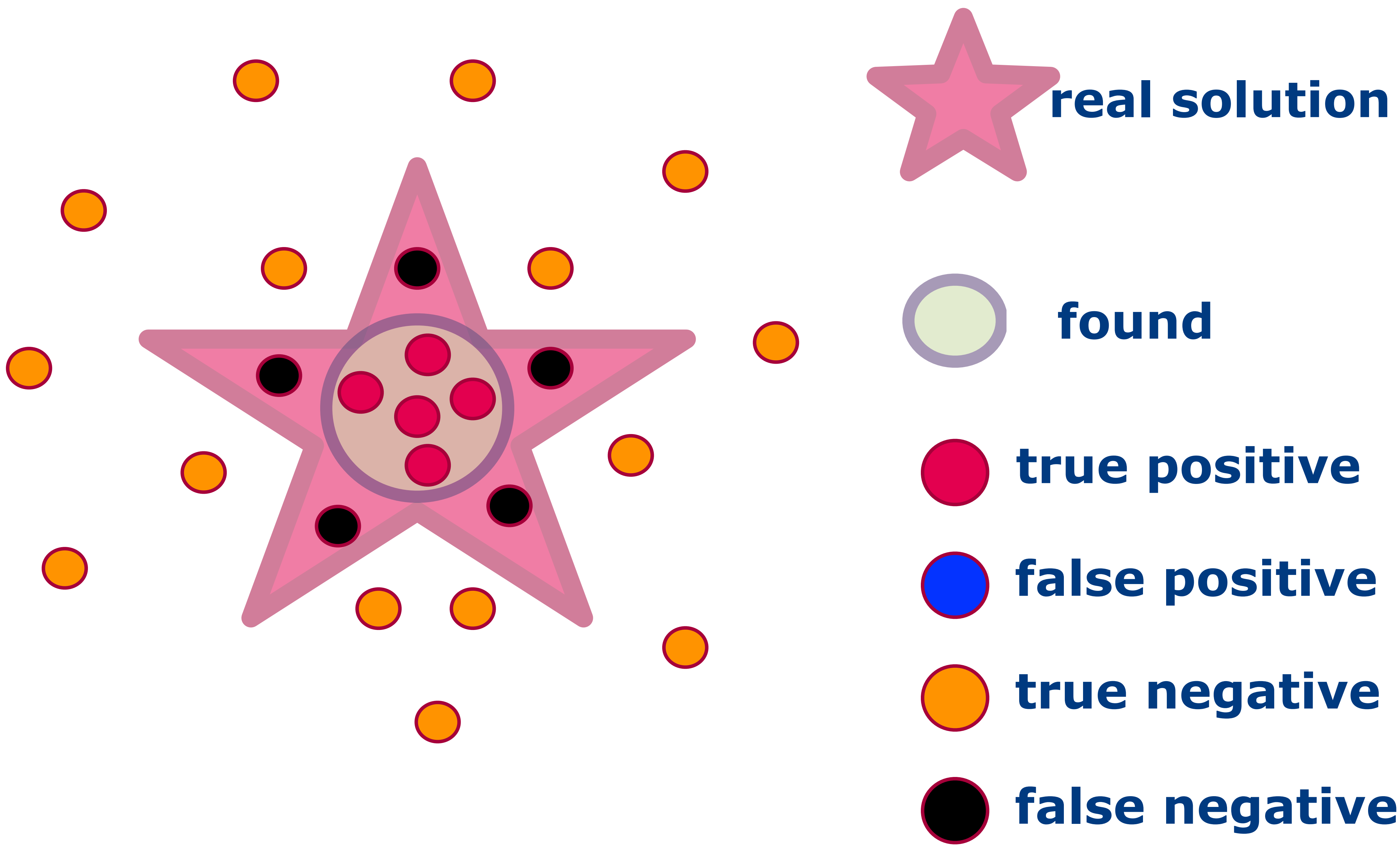


During interpretation: boredom while flipping through false alarms
After interpretation: security of having checked everything!

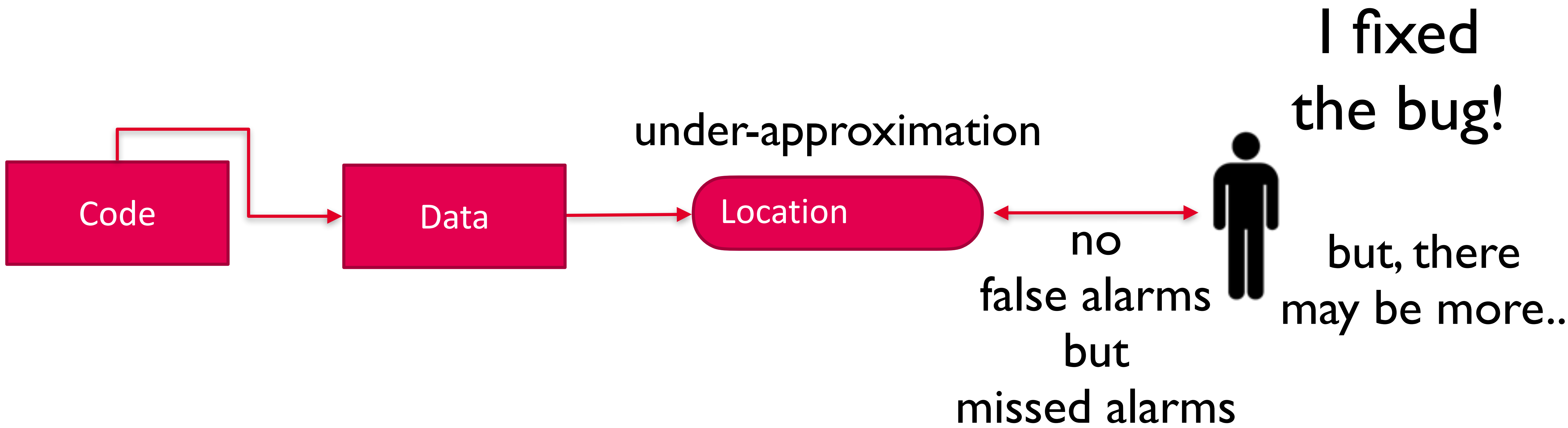
Efficient (and risky) under approximation

checked all object parameters for asserts != null

found some unchecked object parameters, but we missed null elements of array parameters



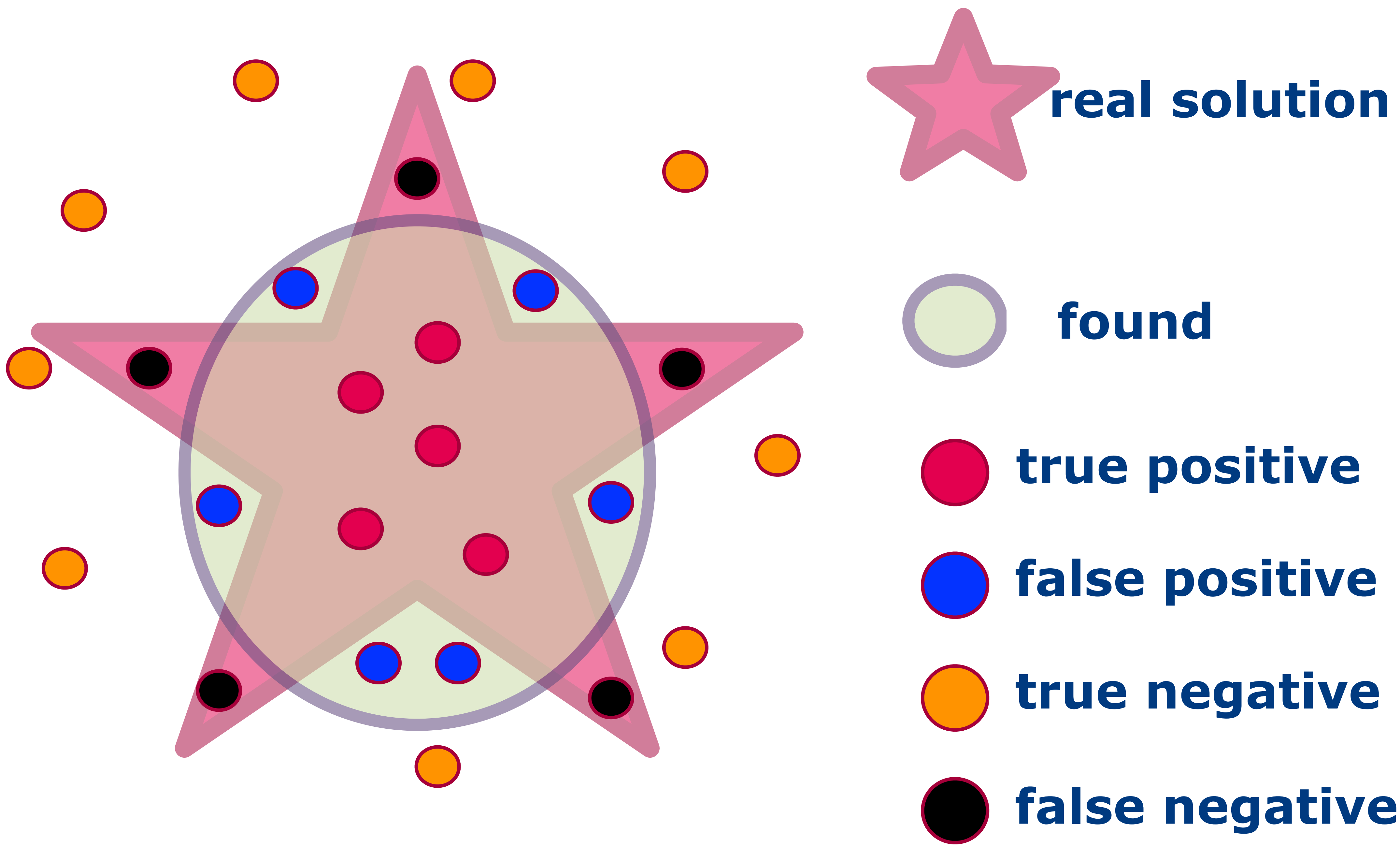
The pro's and con's of under-approximation



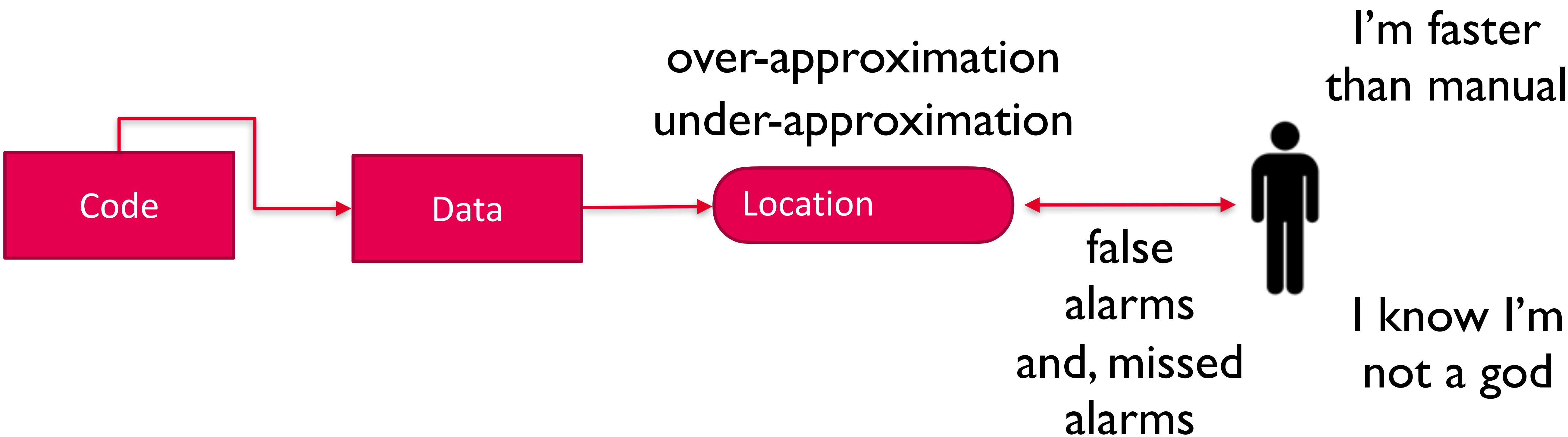
During interpretation: rapid progress, because every bug we see we can fix
After interpretation: what if there is another such bug?

Both under and over approximation (messy)

all unchecked object parameters **and** all {integer, boolean, float} parameters that did not need to be checked **and** we missed the unchecked null parameters



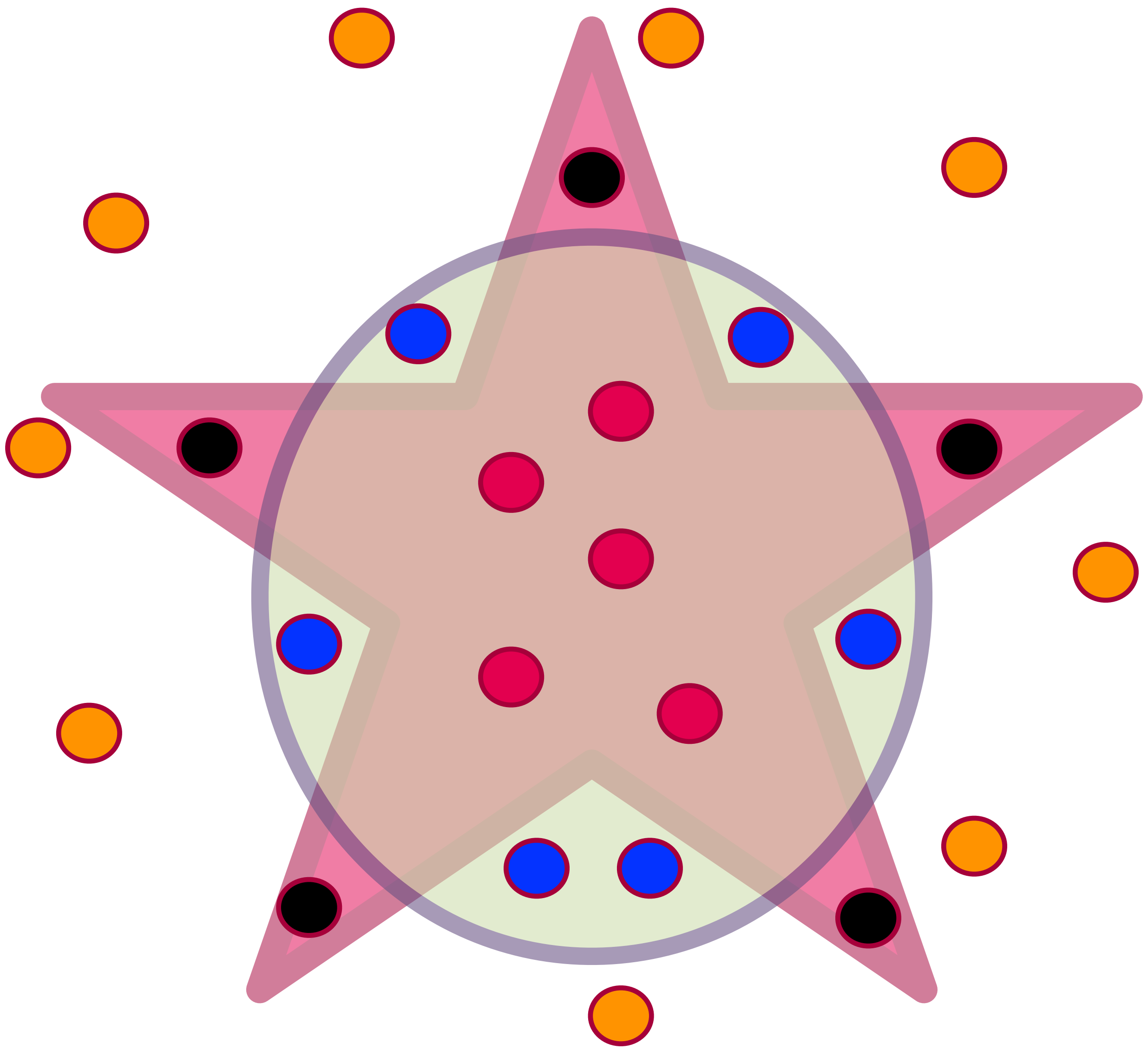
The pro's and con's of general inaccuracy



During interpretation: depending on the accuracy level, fixing bugs or being bored
After interpretation: understanding that your knowledge is still limited

Then: take any opportunity to improve the accuracy of the analysis script

All our code analyses are going to be inaccurate



It helps a lot if you can find out which it is: over, under or both

Inaccurate analysis scripts are (almost) always better than a manual analysis

Computers are fast, patient, and consistent, (and you are not).

Today's Demos

- Equality Contract
- Extract Class Diagram
- Check Architecture Conformance
- Rewrite bad idioms

a real bug

hash	object
1	"aap"
2	"noot", "gijs"
3	"mies"

- **Object.hashCode()** maps objects to integers
- **Object.equals(Object other)** checks if objects are the same
- hashCode/equals contract
  - "if two objects are equal, then they must have the same hashCode"
  - otherwise you can't find objects in hash tables or associative arrays

scheme://authority/path#fragment

```java
        private final String scheme;
        private final String authority;
        private final String path;
        private final String fragment;

        @Override
        public int hashCode() {
            return scheme.hashCode() + authority.hashCode() + path.hashCode() + fragment.hashCode();
        }

        @Override
        public boolean equals(@Nullable Object obj) {
            if (obj == null) {
                return false;
            }

            if (this == obj) {
                return true;
            }

            if (obj.getClass() == getClass()){
                FragmentPathAuthorityURI u = (FragmentPathAuthorityURI)obj;
                return scheme.equals(u.scheme)
                    && authority.equals(u.authority)
                    && path.equals(u.path)
                    && fragment.equals(u.fragment)
                    ;
            }
            return false;
        }
```

hashCode and equals methods should come together

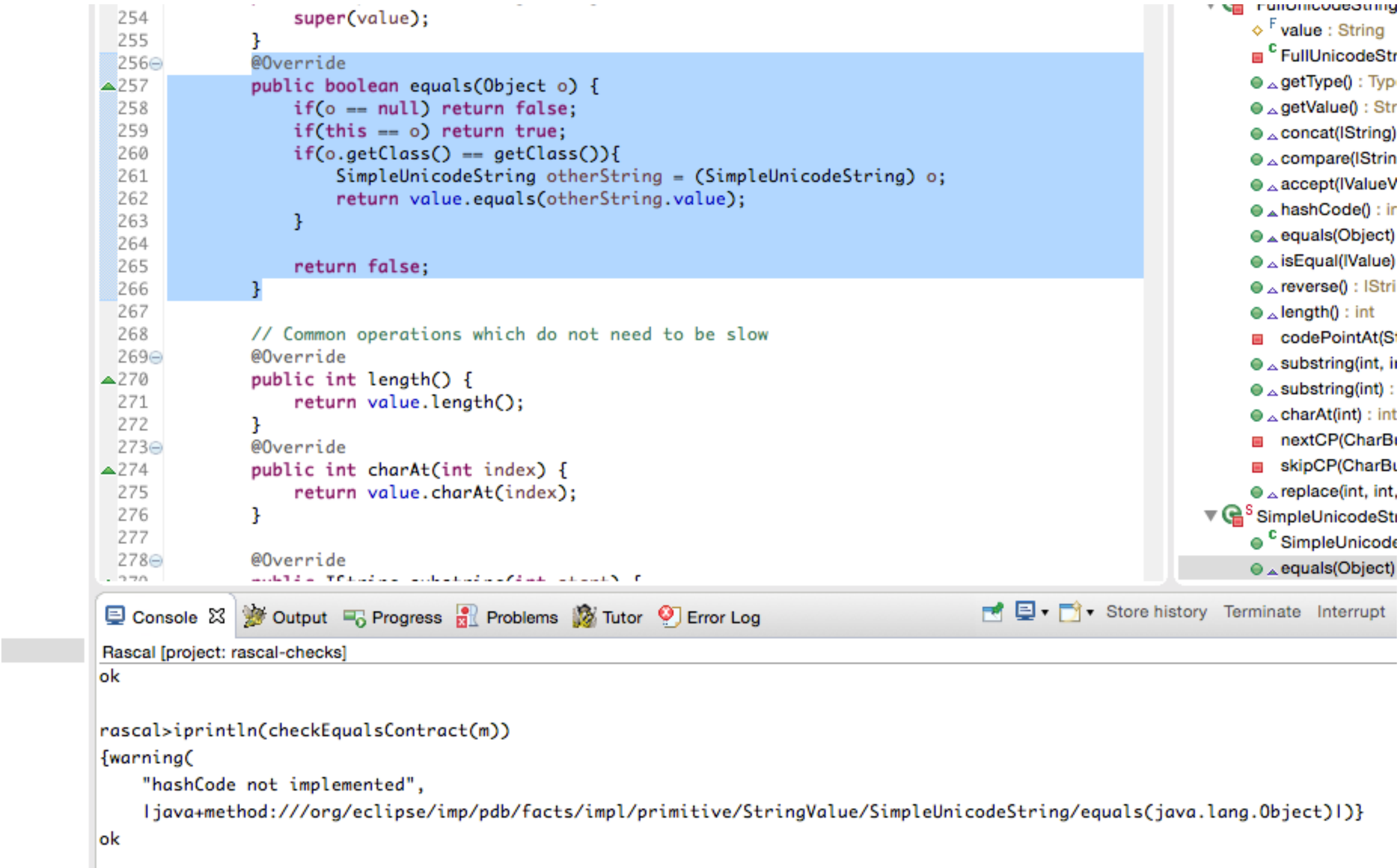
and they should use the same fields

[goto vscode]

Check Equals Contract

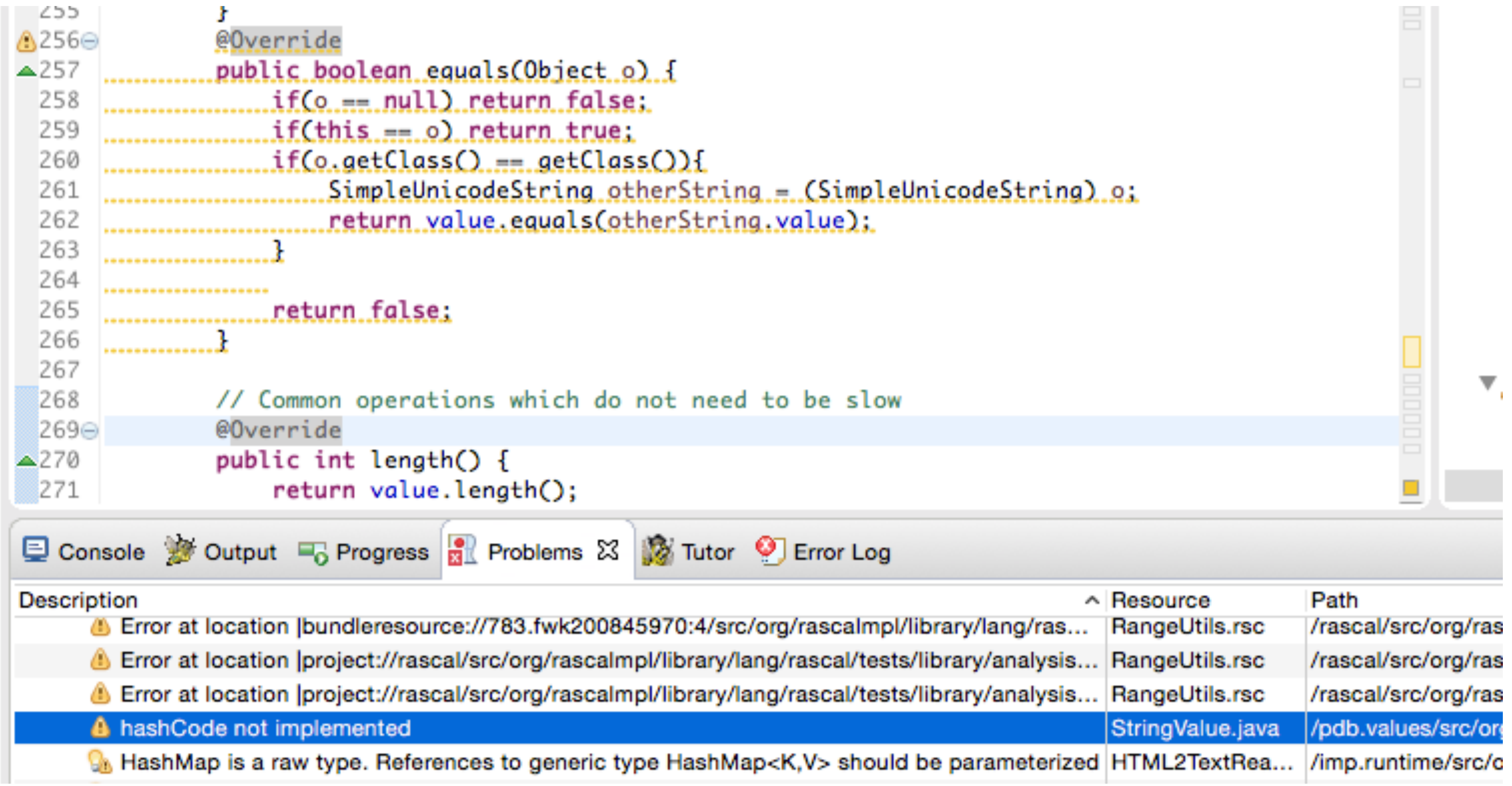
```
loc equalsMethod = |java+method:///java/lang/Object/equals(java.lang.Object)|;
loc hashCodeMethod = |java+method:///java/lang/Object/hashCode()|;

set[Message] checkEqualsContract(M3 m) {
   overrides = (m@methodOverrides<to,from>)+;

   equals = overrides[equalsMethod];
   hashCodes = overrides[hashCodeMethod];

   violators
      = m@containment<to,from>[equals]
      - (m@containment<to,from>[hashCodes])
      - {cl | cl <- classes(m), abstract() in m@modifiers[cl]};

   return { warning("hashCode not implemented", onlyEquals)
      | cl <- violators, onlyEquals <- m@containment[cl] & equals };
}
```

Result



super(value);
}
@Override
public boolean equals(Object o) {
if(o == null) return false;
if(this == o) return true;
if(o.getClass() == getClass()){
SimpleUnicodeString otherString = (SimpleUnicodeString) o;
return value.equals(otherString.value);
}
return false;
}
// Common operations which do not need to be slow
@Override
public int length() {
return value.length();
}
@Override
public int charAt(int index) {
return value.charAt(index);
}
@Override
value : String
FullUnicodeStr
getType() : Typ
getValue() : Str
concat(IString)
compare(IStrin
accept(IValueV
hashCode() : ir
equals(Object)
isEqual(IValue)
reverse() : IStri
length() : int
codePointAt(S
substring(int, i
substring(int) :
charAt(int) : int
nextCP(CharBı
skipCP(CharBu
replace(int, int,
SimpleUnicodeStı
SimpleUnicode
equals(Object)
Console
Output
Progress
Problems
Tutor
Error Log
Store history
Terminate
Interrupt
Rascal [project: rascal-checks]
ok
rascal>iprintln(checkEqualsContract(m))
{warning(
"hashCode not implemented",
|java+method:///org/eclipse/imp/pdb/facts/impl/primitive/StringValue/SimpleUnicodeString/equals(java.lang.Object)|)}
ok

Result



@Override
public boolean equals(Object o) {
if(o == null) return false;
if(this == o) return true;
if(o.getClass() == getClass()){
SimpleUnicodeString otherString = (SimpleUnicodeString) o;
return value.equals(otherString.value);
}
return false;
}
// Common operations which do not need to be slow
@Override
public int length() {
return value.length();
Console
Output
Progress
Problems
Tutor
Error Log
Description
Resource
Path
Error at location |bundleresource://783.fwk200845970:4/src/org/rascalmpl/library/lang/ras...
RangeUtils.rsc
/rascal/src/org/ras
Error at location |project://rascal/src/org/rascalmpl/library/lang/rascal/tests/library/analysis...
RangeUtils.rsc
/rascal/src/org/ras
Error at location |project://rascal/src/org/rascalmpl/library/lang/rascal/tests/library/analysis...
RangeUtils.rsc
/rascal/src/org/ras
hashCode not implemented
StringValue.java
/pdb.values/src/or
HashMap is a raw type. References to generic type HashMap<K,V> should be parameterized
HTML2TextRea...
/imp.runtime/src/o

Check Equals Contract v2

```
set[Message] checkEqualsAndHashUseSameFields(M3 m) {
   overrides = (m@methodOverrides<to,from>)+;

   equals = overrides[equalsMethod];
   hashCodes = overrides[hashCodeMethod];

   pairs
      = invert(rangeR(m@containment, equals))
      o rangeR(m@containment, hashCodes);

   return
      { warning("equals also uses <fieldName(f)>", hs)
      | <eq, hs> <- pairs, f <- m@fieldAccess[eq] - m@fieldAccess[hs]}
      +
      { warning("hashCode also uses <fieldName(f)>", hs)
      | <eq, hs> <- pairs, f <- m@fieldAccess[hs] - m@fieldAccess[eq]};
}
```

Class Diagram Extraction

```
rel[loc, loc] createModel(M3 m)
    = { <c, t> | c <- classes(m), f <- fields(m, c)
        , !isStatic(m, f), <f, loc t> <- m@typeDependency };
```

Architecture Conformance

- Manual Code Review doesn't scale
- Especially for new rules for a large system
- Automate!

“Bad” idioms

```
if (x > 0) {
  return true;
}
else {
  return false;
}
```

```
if (!(x > 0)) {
  …;
}
else {
  …;
}
```

#gotofail

```
if (x)
  y;
  return true;
```

Wrapping up

When code becomes data we can...

query it | generate it
visualize it | simplify it
transform it | check it

... automatically ...

and become a **better** at code maintenance tasks

that make up most of our days as programmers.

http://www.rascalmpl.org

bleeding edge new **VScode** extension
stable **Eclipse** version (win,linux)
Commandline version (win,linux,mac)

create your own:
languages with IDE support
code analyses
code transformations
code visualizations
code generators
code … whatever!

stable
Java, C, C++, PHP

experimental
Python, C#, JS

Open-source project

- https://github.com/usethesource/rascal
- https://github.com/usethesource/rascal-language-servers
    - **issues**: please be nice, give many details, ask anything
    - **pull requests**: talk about it with us before you start
- **questions**: https://stackoverflow.com/questions/tagged/rascal
    - ask questions that have (Rascal) code as an answer
- Growing community: CWI, TUE, UvA, OU, RUG, ECU, Bergen University, ...
- http://swat.engineering = language engineering with Rascal
    - making software better with language engineering

Take home

1. *Designing code* is interesting and fun
   - *Analyzing code* is more important
   - {sh,c,w}ould be interesting and fun too
2. Analyzing code should be automated:
   - use the generic analyses of your IDE
   - script your own analyses with **Rascal**