universitätbonn ROOTS

# Locating the Meaning of Terms in Source Code

## 18th Working Conference on Reverse Engineering(WCRE)

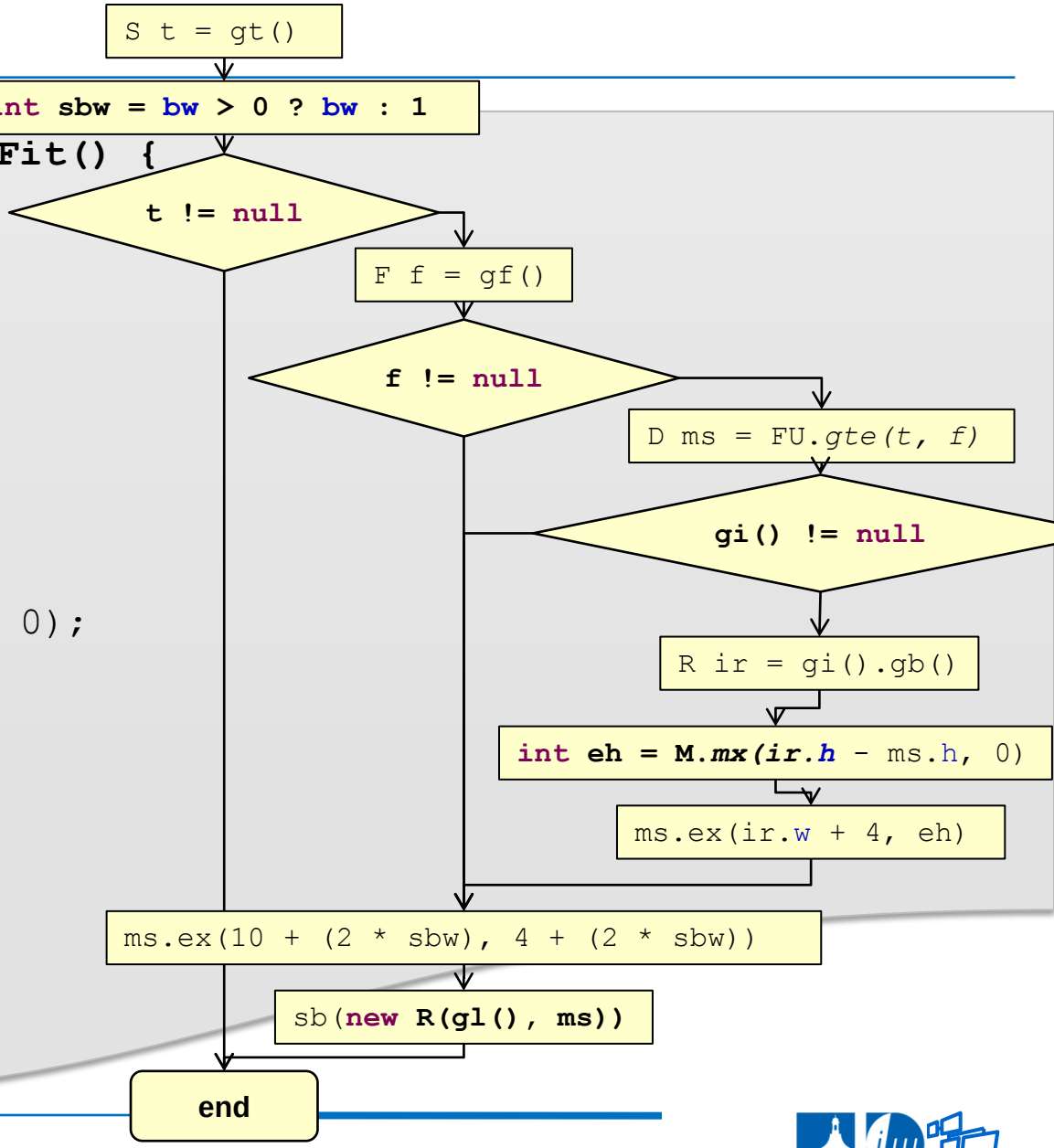
Jan Nonnen, Daniel Speicher, Paul Imhoff

Limerick, 18th October 2011

# Two Different Points of View on Source Code

# Compiler Perspective

```
protected void adjustBoundsToFit() {
  S t = gt();
  int sbw = bw > 0 ? bw : 1;
  if ((t != null)) {
    F f = gf();
    if (f != null) {
      D ms = FU.gte(t, f);
      if (gi() != null) {
        R ir = gi().gb();
        int eh
          = M.mx(ir.h - ms.h, 0);
        ms.ex(ir.w + 4, eh);
      }
      ms.ex(10 + (2 * sbw),
        4 + (2 * sbw));
      sb(new R(gl(), ms));
    }
  }
}
```



# Developer Perspective

```
protected void adjustBoundsToFit() {
  String text = getText();
  int safeBorderWidth = borderWidth > 0 ? borderWidth : 1;
  if ((text != null)) {
    Font font = getFont();
    if (font != null) {
      Dimension minSize = FigureUtilities.getTextExtents(text, font);
      if (getIcon() != null) {
        Rectangle imageRect = getIcon().getBounds();
        int expandHeight
          = Math.max(imageRect.height - minSize.height, 0);
        minSize.expand(imageRect.width + 4, expandHeight);
      }
      minSize.expand(10 + (2 * safeBorderWidth),
        4 + (2 * safeBorderWidth));
      setBounds(new Rectangle(getLocation(), minSize));
    }
  }
}
```

# Understanding through Names

```
protected void adjust bounds to fit() {
  String text = get text();
  int safe border width = border width > 0 ? border width : 1;
  if ((text != null)) {
    Font font = get font();
    if (font != null) {
      Dimension min size = FigureUtilities.get text extents(text, font);
      if (get icon() != null) {
        Rectangle image rect = get icon().get bounds();
        int expand height
          = Math.max(image rect.height - min size.height, 0);
        min size.expand(image rect.width + 4, expand height);
      }
      min size.expand(10 + (2 * safe border width),
        4 + (2 * safe border width));
      set bounds(new Rectangle(get location(), min size));
    }
  }
}
```

# Understanding through Names

```
protected void adjustBoundsToFit() {
```

adjust bounds to fit text get safe border width null font size extents icon image rect expand height Math max min set bounds location

# Introduction ▸ Preparation / Notation

- Identifier are
  - Class-, method-, field- or local variablenames
- For our analysis we split and lower case them:
  - CamelCase notation

    `IContextProvider` yields (`i`, `context`, `provider`)
  - Special characters(e.g. „_“) or numbers

    `my_variable2string` yields (`my`, `variable`, `2`, `string`)
- Lemmatization as a normalization is afterwards applied
- These parts are further refered to as **terms**.

# Introduction ▸ The Inconsistent Context

```
class NearCompaniesContextTracking implements ServiceTrackerCustomizer {
  public static ServiceTracker openTracker(BundleContext bundleContext,
                                           Observer contextObserver)
  {
   ServiceTrackerCustomizer customizer=new NearCompaniesContextTracking(
                                                          bundleContext,
                                                          contextObserver);

   ServiceTracker serviceTracker = new ServiceTracker(bundleContext,
                        INearCompaniesContextProvider.class.getName(), customizer);

   serviceTracker.open();
   return serviceTracker;
  }
 …
}
```

Source: CSI Project of our Department

# Introduction ▸ The Inconsistent Context

```
class NearCompaniesContextTracking implements ServiceTrackerCustomizer {
  public static ServiceTracker openTracker(BundleContext bundleContext,
                                           Observer contextObserver)
  {
   ServiceTrackerCustomizer customizer=new NearCompaniesContextTracking(
                                                        bundleContext,
                                                        contextObserver);

   ServiceTracker serviceTracker = new ServiceTracker(bundleContext,
                        INearCompaniesContextProvider.class.getName(), customizer);

   serviceTracker.open();
   return serviceTracker;
  }
 …
}
```

Source: CSI Project of our Department

# Introduction ▸ The Inconsistent Context

```
class NearCompaniesContextTracking implements ServiceTrackerCustomizer {
  public static ServiceTracker openTracker(BundleContext bundleContext,
                                           Observer contextObserver)
  {
   ServiceTrackerCustomizer customizer=new NearCompaniesContextTracking(
                                                          bundleContext,
                                                          contextObserver);

   ServiceTracker serviceTracker = new ServiceTracker(bundleContext,
                        INearCompaniesContextProvider.class.getName(), customizer);

   serviceTracker.open();
   return serviceTracker;
  }
 …
}
```

Source: CSI Project of our Department

# Introduction ▸ Research Question&Approach

- How can we automatically find out, if different meanings occur?
  - Approach: Define „term introductions“ and analyse the usage
- Can we specify which meaning was meant?
  - Approach: Identify the introduction location together with the help of static code dependencies and the data flow
- Is there a component that shares ambiguous term meanings to the outside?
  - Approach: Identify these components and visualise them

# Introduction ▸ Meaning of Identifiers

What does **cat** mean?

„cat“ is an Animal

„cat“ is a Unix program to concatenate files

Which meaning is used for the method named **killCat()** ?

# Term-Introduction

# Term-Introduction ▸ Goal

- Find introduction locations in a project

- Hypothesis:
  - The meaning of an identifier can be derived from the code…
  - … if we have found an appropriate code „snippet“

# Term-Introduction ▸ How to find?

- Goal: Approximation with an heuristic

# Term-Introduction ▸ Approach

- Explorative empirical study:
  - Initial set of heuristics
  - Evaluation of precision und recall on a set of projects (**Exploration Phase**)
    - ⇨ Focus on Precision
  - Estimate of precision and recall based on samples
    - ⇨ Manual sample classification
  - Analysis of the results and adaptation of the heuristics
  - Validation on an independent set (**Validation Phase**)

# Term-Introduction ▸ Heuristics

- Exploration Heuristics
  - Atomic Heuristic
  - Compound Heuristic
  - Specialiser Heuristic
- Validation Heuristics
  - Combination of the above with improvements

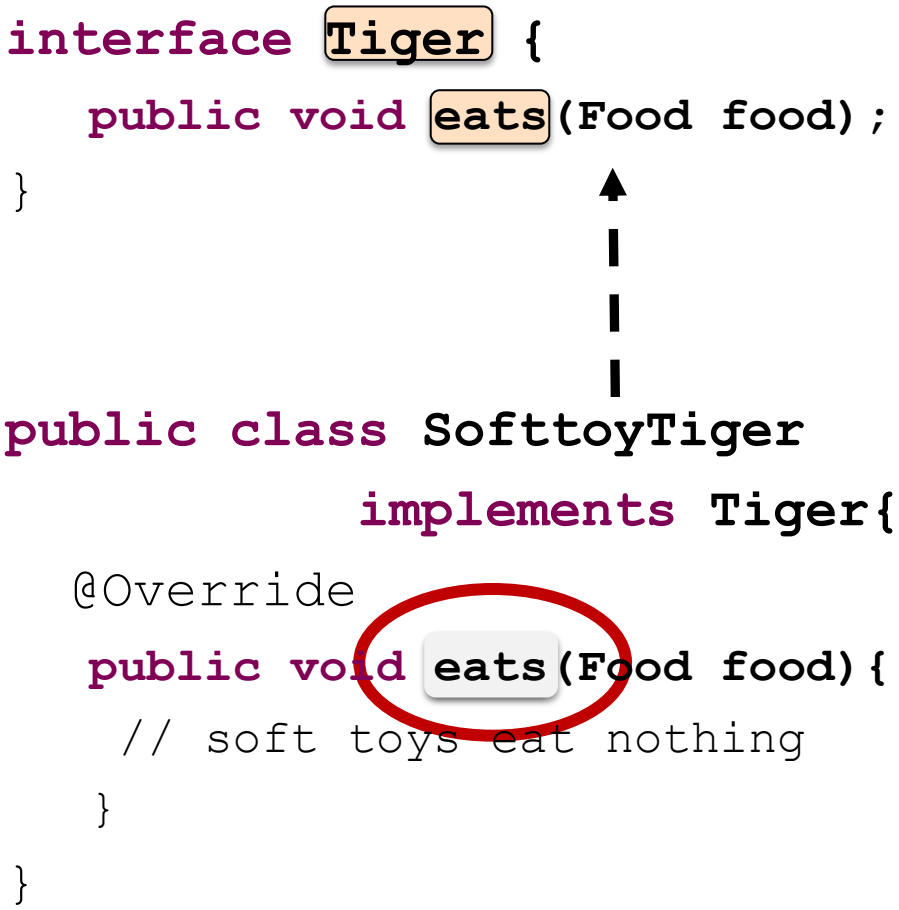
# Term-Introduction ▸ Atomic Heuristic

- Define the class as an introduction location for the term, if the term…
  - Is the class name
  - The name of a public/protected method

```
public class Calvin {

    private void likes(){

    }

    public void eats(Food food){

    }
}
```

# Term-Introduction ▸ Reduction

- Problem of the Atomic Heuristic
  - Introduction of inherited methods
- Solution: Reduction of depending introductions to the root introduction
  - Chains of introduction locations connected via static code dependencies
  - Remove depending location



# Term-Introduction ▸ Compound Heuristic

- Another problem:
  - „softtoy“concept not introduced
- Solution:
  - Use term splitting
  - Introduce iteratively single not introduced terms in a class name

```
interface Tiger {
    public void eats(Food food);
}
```

```
public class SofttoyTiger
                implements Tiger{
   @Override
    public void eats(Food food){

    }
}
```

# Term-Introduction ▸ Specialiser Heuristic

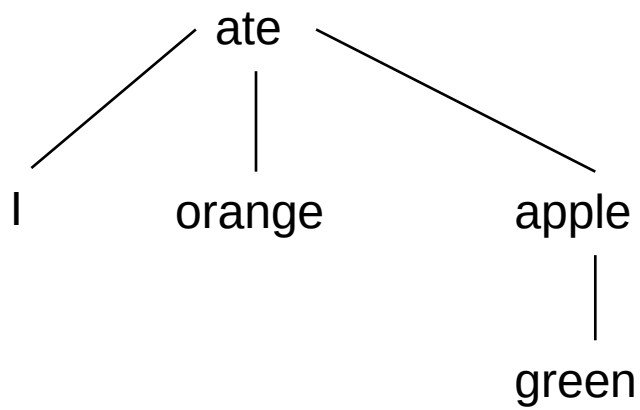
- Problem:
  - Previous heuristics only worked because there was a „tiger“ present
- Heuristic based on a work of Falleri et al.:

  *Automatic Extraction of a WordNet-Like Identifier Network from Software*

  In: 2010 IEEE 18th International Conference on Program Comprehension (ICPC)
- Creation of an internal ontology based solely on identifiers
  - Sorting of the terms based on term dominance

# Term-Introduction ▸ Dominance

I ate an orange and a green apple



# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance

```
interface AbstractTiger {
   public void eats(Food food);
}

public class SofttoyTiger
             implements
             AbstractTiger{
   @Override
   public void eats(Food food){

   }
}
```

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance

**`AbstractTiger`**

**`SofttoyTiger`**

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance

```
abstract tiger
```

```
softtoy tiger
```

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance

```
abstract tiger
  Noun   Noun

softtoy tiger
  Noun   Noun
```

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance
  - Remove common dominant terms

```
tiger abstract
 Noun   Noun

tiger softtoy
 Noun   Noun
```

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance
  - Remove common/re-occuring dominant terms
  - Define the remaining terms as introduced

```
tiger abstract
  Nomen   Nomen

tiger softtoy
  Nomen   Nomen
```

# Term-Introduction ▸ Specialiser Heuristic

- Algorithm:
  - Sort the terms based on term dominance
  - Remove common/re-occuring dominant terms
  - Define the remaining terms as introduced

```
interface AbstractTiger {
   public void eats(Food food);
}
```

```
public class SofttoyTiger
             implements
AbstractTiger{
   @Override
   public void eats(Food food){

   }
}
```

# Evaluation Project Set

**Explorations Phase**

| | | | |
|---|---|---|---|
| Apache Commons Logging | iText | JWNL | QuickUML |
| BCEL | Jaxen | Lexi | Smack |
| GlazedLists | jEdit | OpenCloud | Time & Money Code Library |
| Google Workspacemechanic | Jsch | PlanetaMessenger | TreeTagger4Java |
| IBM WALA Core | JVM Monitor.core | PMD | Zest Core |

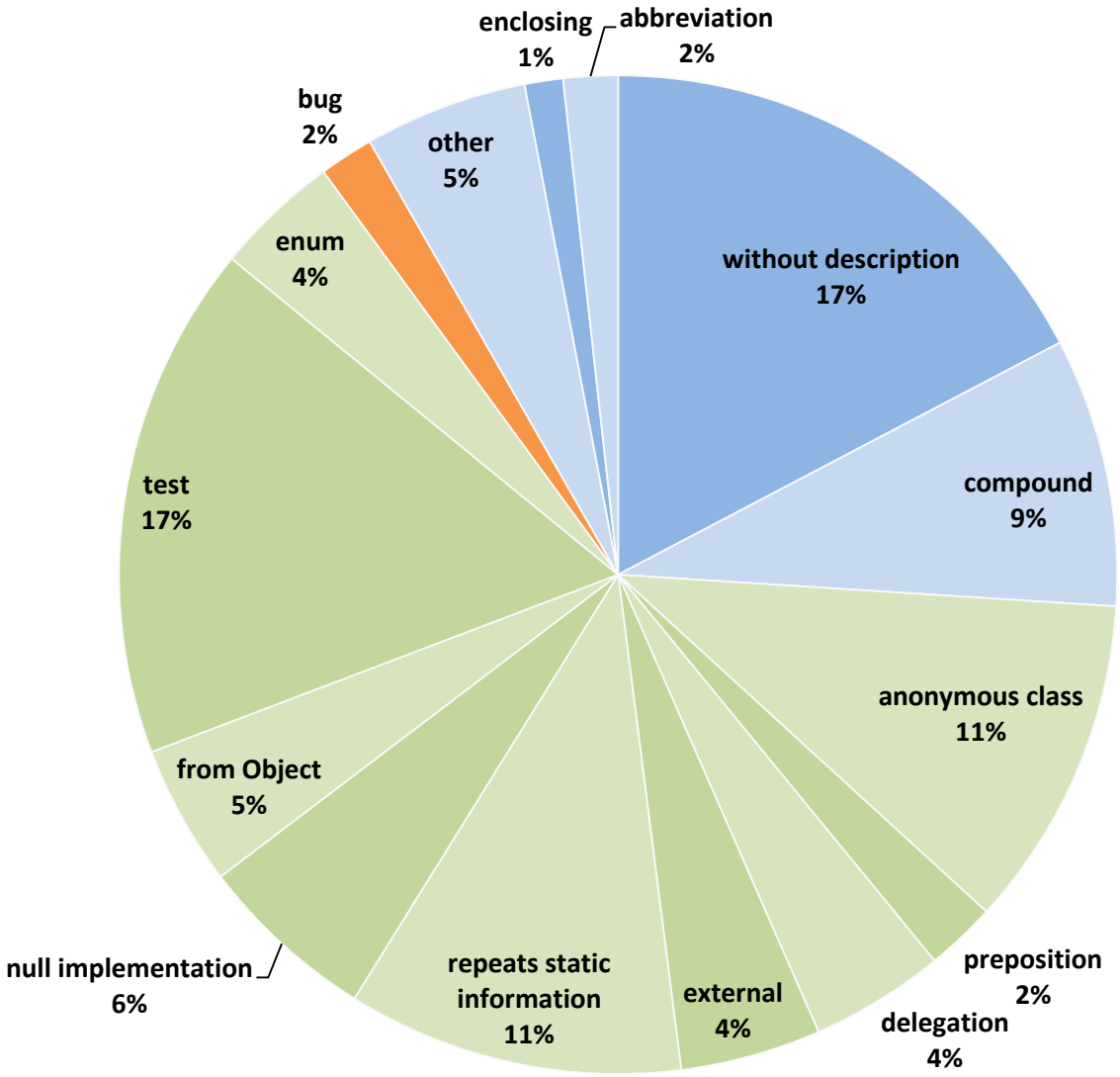
**Validations Phase**

| | | | |
|---|---|---|---|
| Bouncy Castle Crypto | Concept Explorer | NGramJ | yGuard |
| BSF | DDDSample | Rhino | zxing |
| Cobertura | Edu.cmu.hcii.paint | | |

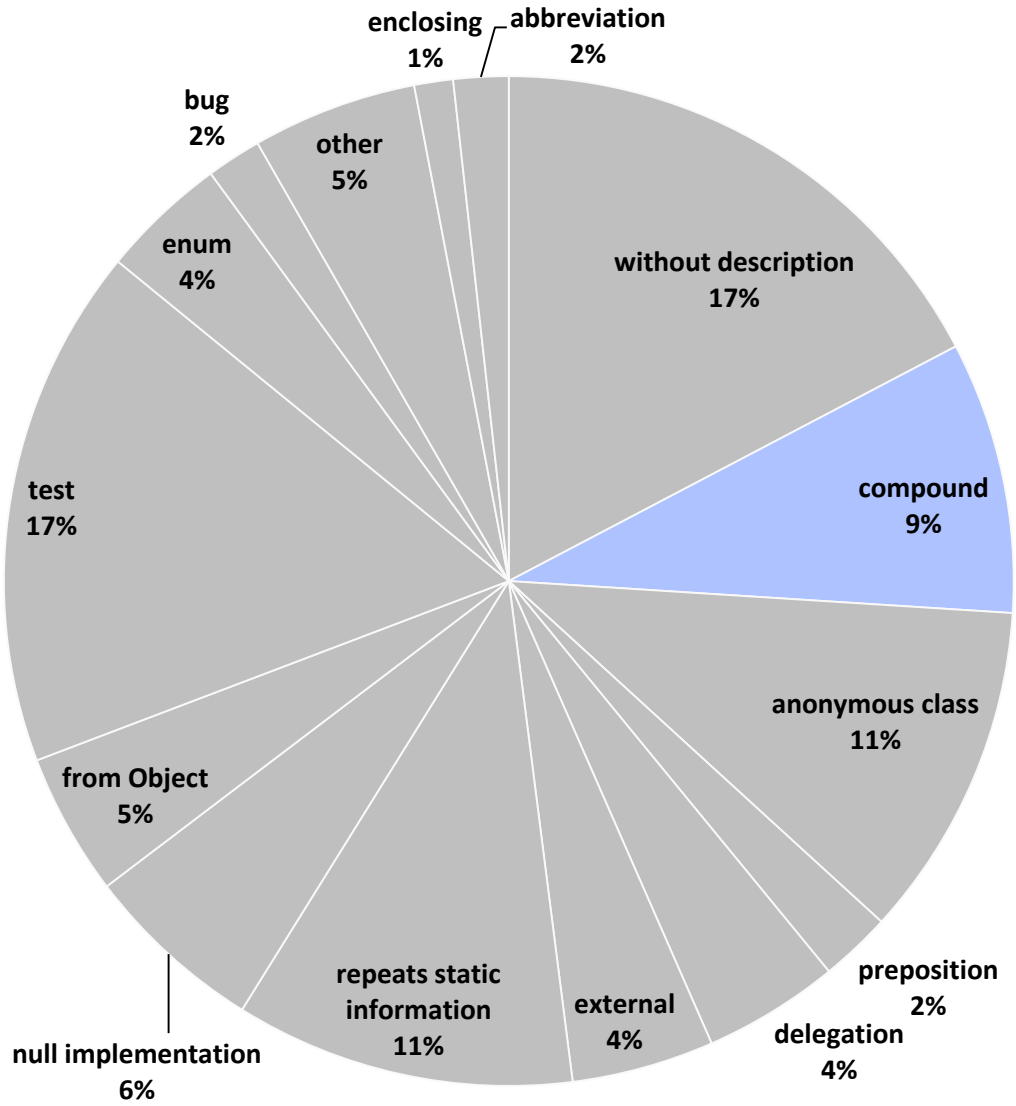
30 Open Source Projects
2.000.000 Lines of Code
8000 Manual Validated Samples

# What have we learned from the Exploration Phase?

# False Positives ▸ Categories

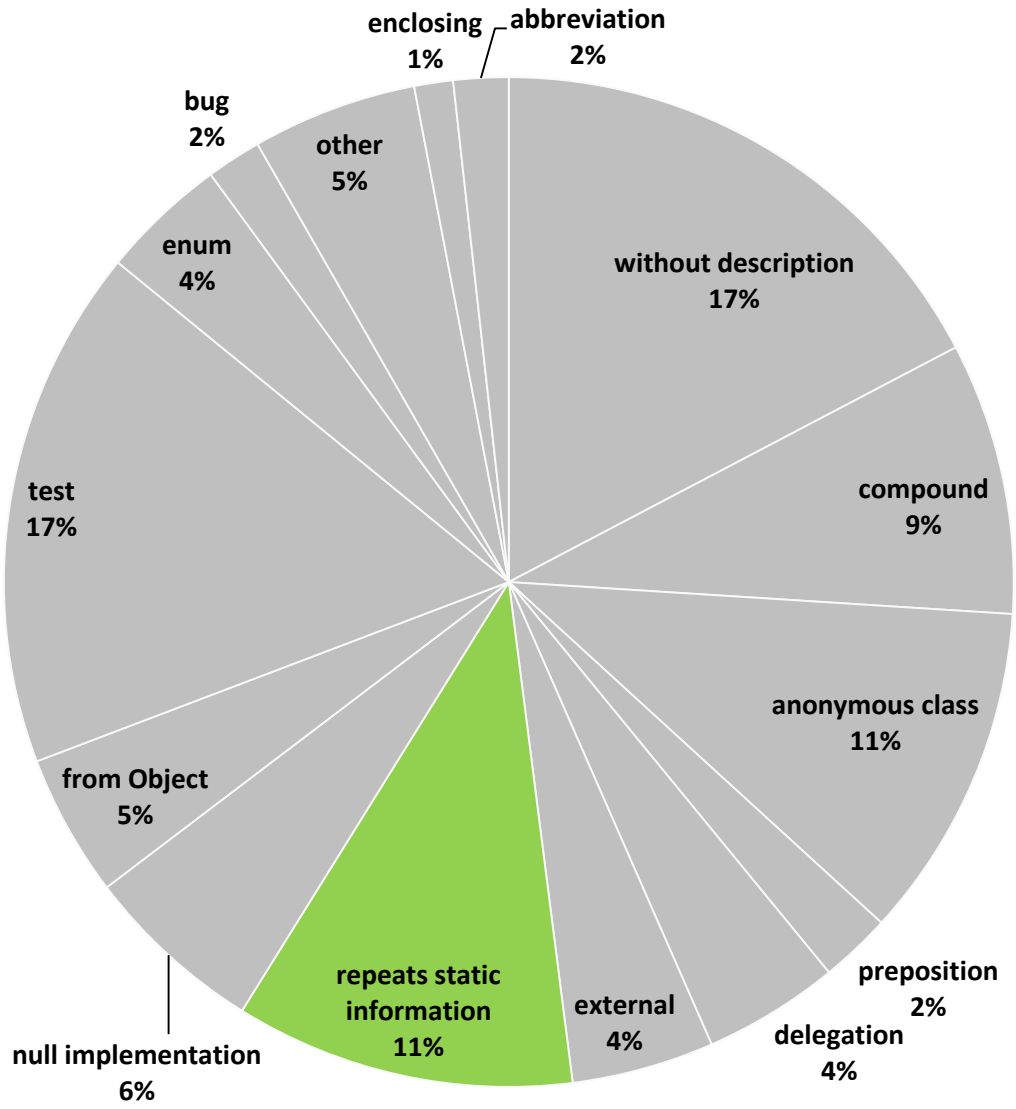


# False Positives ▸ Categories



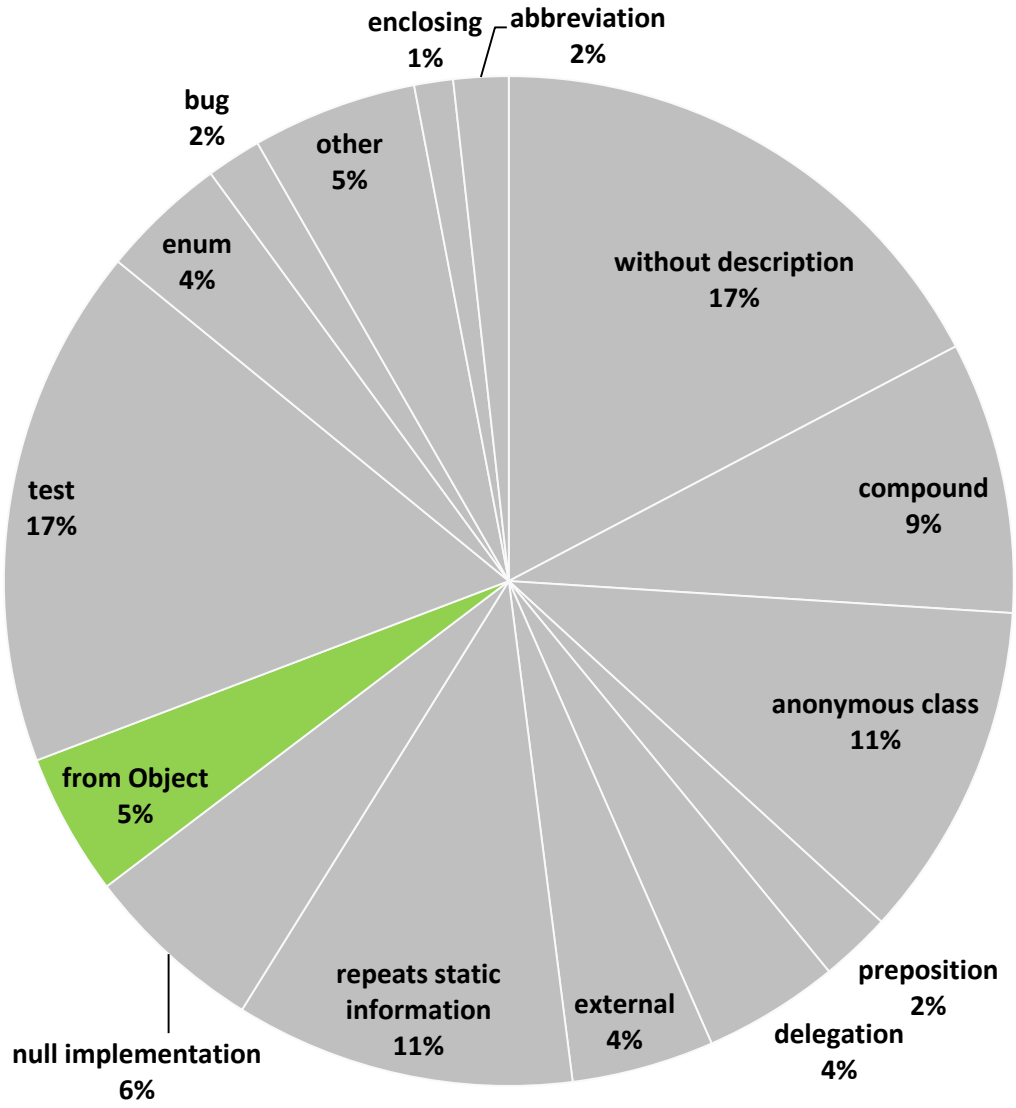

```
RichMediaFactory
```

# False Positives ▸ Categories



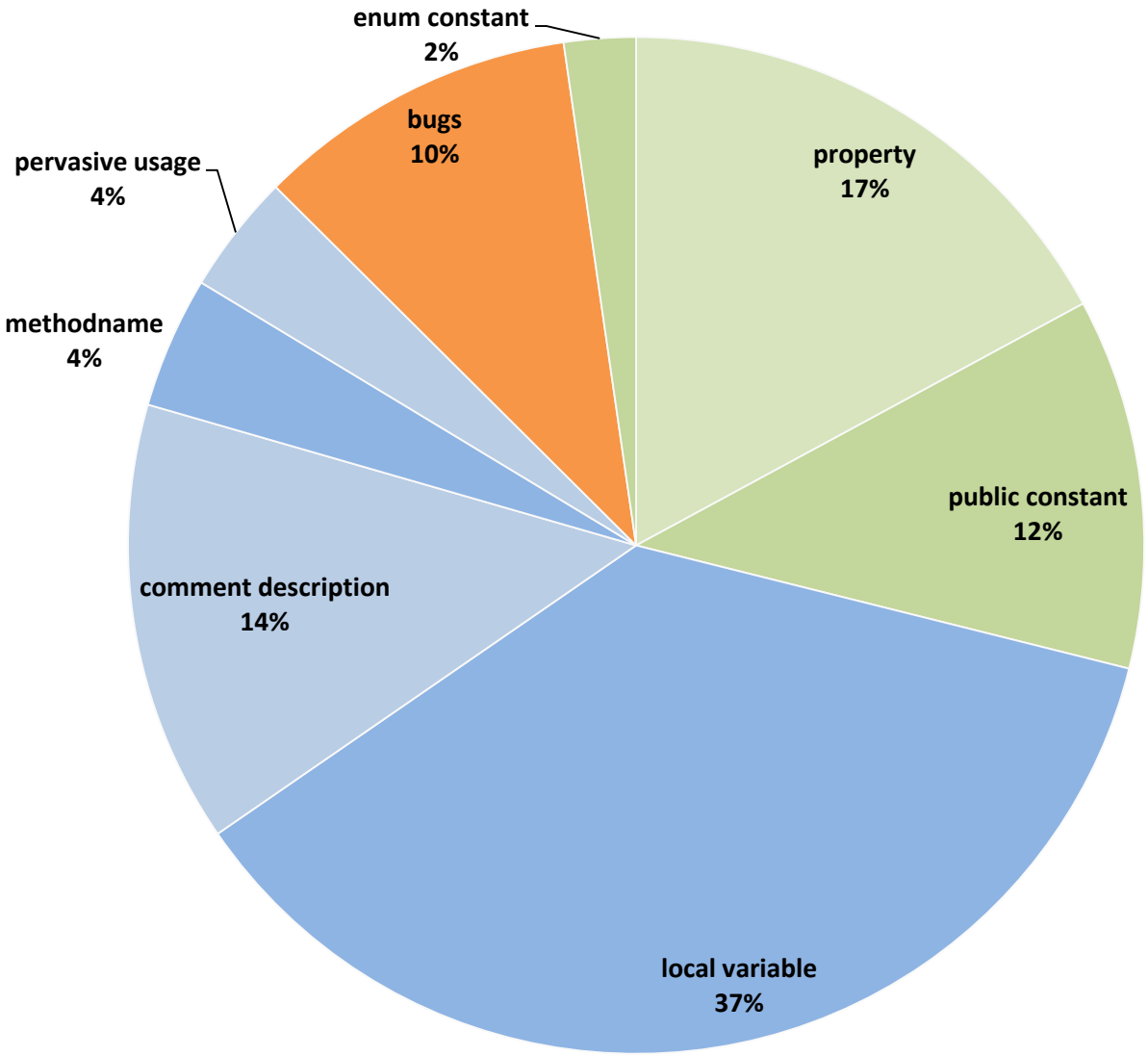

```
public abstract class
        AbstractModel{
..
}
```

# False Positives ▸ Categories




```
@Override
public boolean equals(..
```

# False Negatives ▸ Categories



# Adjustments for the Validation Phase

- Definition of **insignificant terms** and patterns to detect those
  - Example: `Abstract` at the beginning of the name of an abstract class
- Reevaluation of possible locations in form of **instructive locations**
  - Example: Enum-Classes
- Application of the rules to external code (**external introductions**)
  - Reduction preserves introduction location of libraries
- Combination of the inital heuristics to a single one

# Conclusion ▸ Summary

- Concept of „term introduction“ to localise project specific term meanings
- A heuristic derived with an explorative empirical study
  - Precision Median of 76%
- No external ontology applied
- Identification of inconsistent terms
- *Project Dictionary* and *Term Cloud* as Eclipse Plugin
  - Contains:  Term, Frequency, Introduction location, Uses
  - Possibility to denote and share term meanings in a team

# Summary ▸ Limerick

*I discussed introduction of a term,*
*still there is some more to learn,*
*but with the data to guide*
*and the heuristics applied,*
*the cats life is no longer a concern.*