Nomen est Omen: Exploring and Exploiting Name Similarities between Arguments and Parameters

Hui Liu [1] Qiurong Liu [1] **Cristian-Alexandru Staicu** [2]
Michael Pradel [2] Yue Luo [1]

[1]Beijing Institute of Technology

[2]Technical University of Darmstadt

May 23, 2016

SOLA SoftwareLab

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

How we see the source code:

```
void writeVersionFile(File file, float version) {
   DataOutputStream dos = new DataOutputStream(file);
   dos.writeFloat(version);
   dos.close();
}
```

Motivation

How we see the source code:

```
void writeVersionFile(File file, float version) {
   DataOutputStream dos = new DataOutputStream(file);
   dos.writeFloat(version);
   dos.close();
}
```

How most analyses see the source code:

```
void a(A b, B c) {
   C d = new C(b);
   d.e(c);
   d.f();
}
```

Motivation

Main Idea

Use similarities between **arguments** and **parameters** names in program analysis.

Motivation

Main Idea

Use similarities between **arguments** and **parameters** names in program analysis.

Parameters (method definition)

```
void writeVersionFile(File file, float version) {...}
```

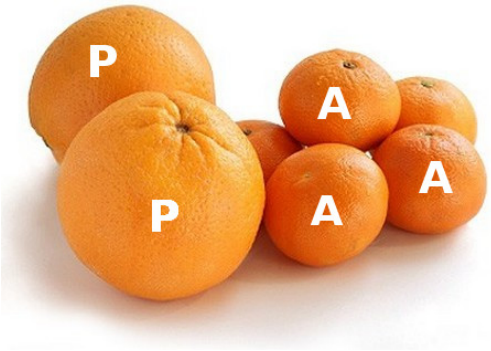
Arguments (call site)

```
writeVersionFile(file, version);
```

```
writeVersionFile(myFile, currentVersion);
```

```
writeVersionFile(target, v);
```

This talk



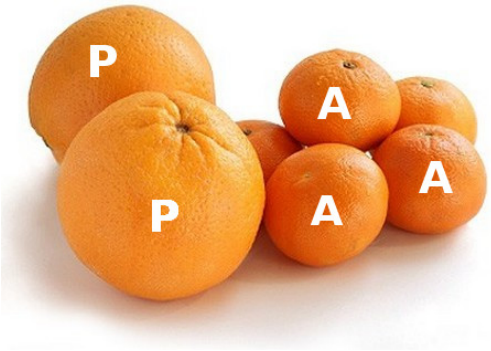
Empirical evidence that:

- names of arguments and parameters are similar
- dissimilar names can be filtered out

Two applications:

- anomaly detection
- arguments recommendation

This talk



Empirical evidence that:

- names of arguments and parameters are similar
- dissimilar names can be filtered out

Two applications:

- anomaly detection
- arguments recommendation

[Allamanis et al., FSE2014], [Allamanis et al., FSE2015], [Butler et al. CSMR2010], [Pradel and Gross, ISSTA2011]

Empirical Study

Are argument and parameter names similar?

Yes, in 31% of the cases even identical.

Empirical Study

Are argument and parameter names similar?

Yes, in 31% of the cases even identical.

Why are some dissimilar?

Mostly because of **short and generic** names.

Empirical Study

Are argument and parameter names similar?

Yes, in 31% of the cases even identical.

Why are some dissimilar?

Mostly because of **short and generic** names.

Can we eliminate dissimilar names?

Yes, a significant part of them.

Empirical Study

Are argument and parameter names similar?

Yes, in 31% of the cases even identical.

Why are some dissimilar?

Mostly because of **short and generic** names.

Can we eliminate dissimilar names?

Yes, a significant part of them.

Do developers pick the most similar arguments?

Yes, in most of the cases.

Methodology and Setup

60 popular Java programs

>600,000 arguments

Retrieve parameters using JDT's static solving

$$lexSim(arg, par) = \frac{|included\_terms(arg,par)| + |included\_terms(par,arg)|}{|terms(arg)| + |terms(par)|}$$

Methodology and Setup

60 popular Java programs

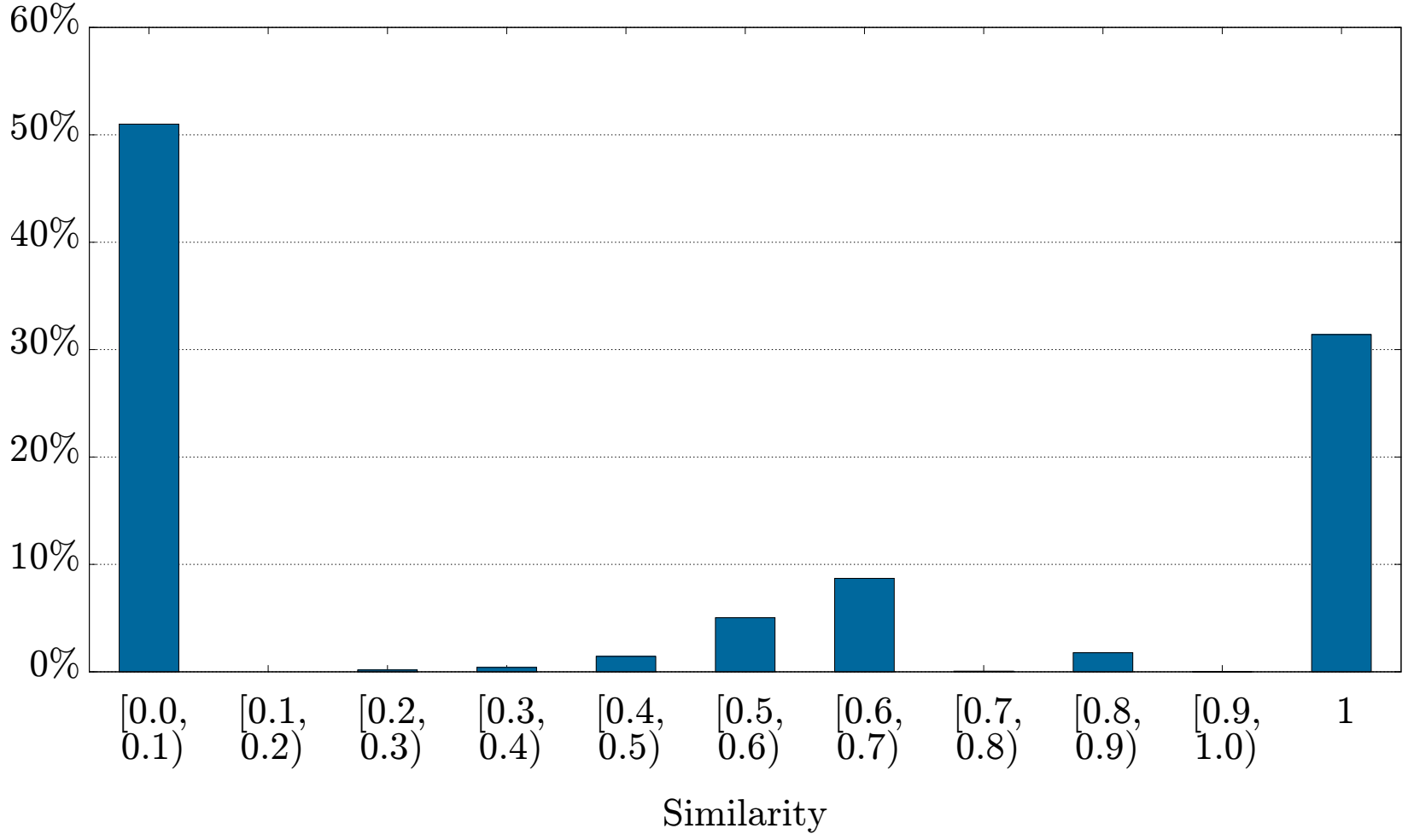
>600,000 arguments

Retrieve parameters using JDT's static solving

$$lexSim(arg, par) = \frac{|included\_terms(arg,par)|+|included\_terms(par,arg)|}{|terms(arg)|+|terms(par)|}$$

$$lexSim(\text{"length"}, \text{"inputLength"}) = \frac{1+1}{1+2} = 0.67$$

Are Argument and Parameter Names Similar?



Why are Some Names Dissimilar?

Short identifiers: 40.5% of the dissimilar pairs have a parameter of length $<= 3$

Why are Some Names Dissimilar?

Short identifiers: 40.5% of the dissimilar pairs have a parameter of length $<= 3$

Generic identifiers: `index`, `item`, `key`, `value` account for 14% of dissimilarities

Can We Eliminate Dissimilar Names?

Example of code containing generic identifier names[1]:

```
public int maxValue(int array[]){
  List<Integer> list = new ArrayList<Integer>();
  for (int i = 0; i < array.length; i++) {
    list.add(array[i]);
  }
 return Collections.max(list);
}
```

[1] stackoverflow, question 1806816

Can We Eliminate Dissimilar Names?

Example of code containing generic identifier names[1]:

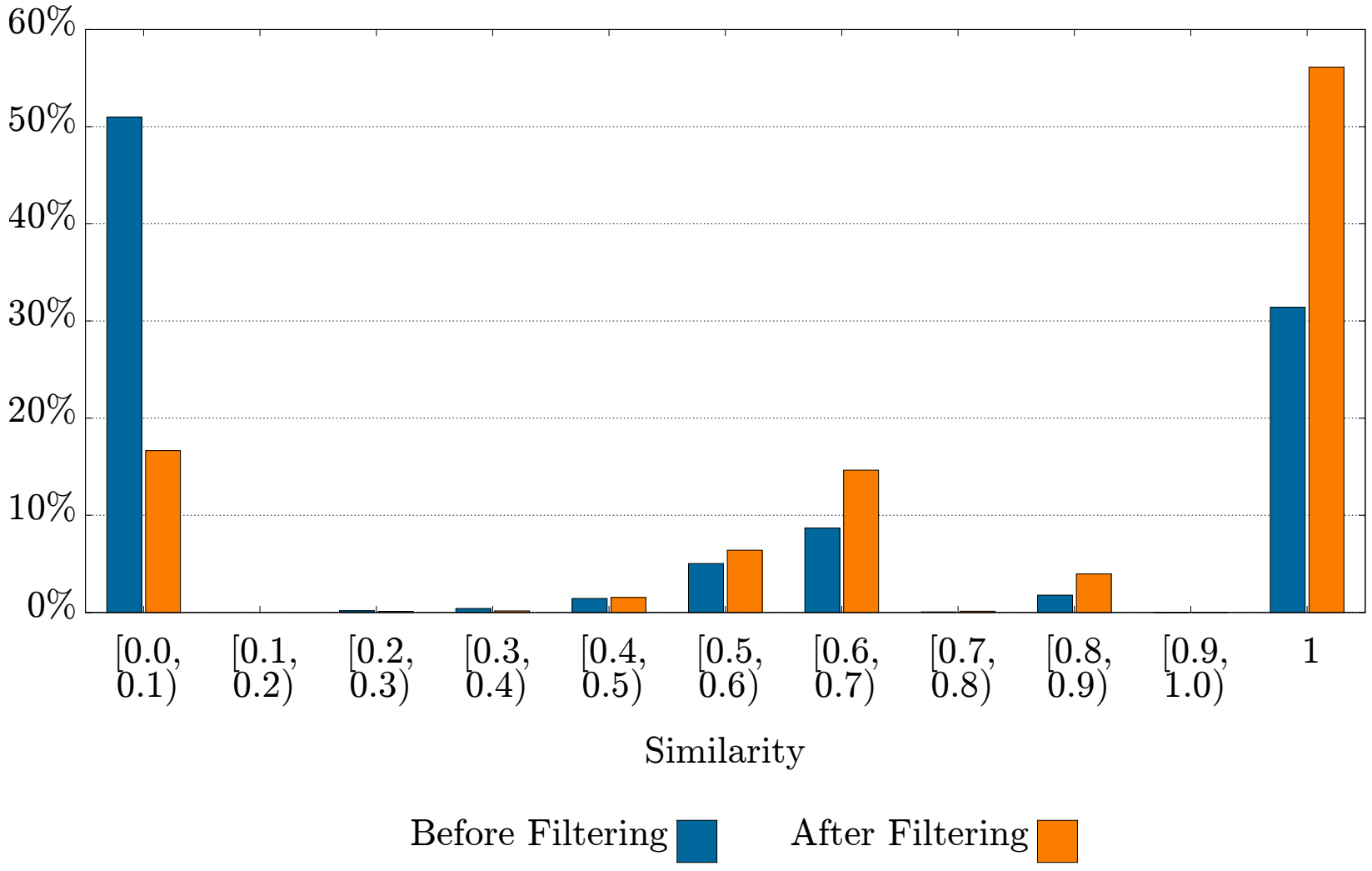
```
public int maxValue(int array[]){
  List<Integer> list = new ArrayList<Integer>();
  for (int i = 0; i < array.length; i++) {
    list.add(array[i]);
  }
 return Collections.max(list);
}
```

Idea

Use a corpus of programs to **infer parameters names** that are likely to appear in dissimilar pairs.

[1] stackoverflow, question 1806816

Pruning Low-Similarity Parameters



Do Developers Pick the Most Similar Arguments?

Compare argument with **potential alternatives**.

Do Developers Pick the Most Similar Arguments?

Compare argument with **potential alternatives**.

Findings

50% of the arguments have no alternatives
13.5% are **strictly** more similar than any other alternative
if filtering out is applied, this number increases two times
6.9% have a more similar alternative

Application 1: Anomaly Detection

Idea

An anomaly is a low-similarity pair that **has a potential alternative that would significantly increase the similarity**.

Application 1: Anomaly Detection

Idea

An anomaly is a low-similarity pair that **has a potential alternative that would significantly increase the similarity**.

Issue in Lightweight Java Game Library:

```
void writeVersionFile(File file, float version) {
    ...
}
File versionFile;
...
writeVersionFile(dir, latestVersion);
```

Application 1: Anomaly Detection

Idea

An anomaly is a low-similarity pair that **has a potential alternative that would significantly increase the similarity**.

Issue in Lightweight Java Game Library:

```
void writeVersionFile(File file, float version) {
    ...
}
File versionFile;
...
writeVersionFile(dir, latestVersion);
```

Anomaly Detection: Results

Ground truth: 14 bugs in the history of the subject programs

Approach detected:

- 6 / 14 and three additional ones
- 127 renaming opportunities

Average precision: 80%

Application 2: Arguments Recommendation

```
private static void compute(int min, int max) {
}

public static void main(String[] args) {
    int min = 5;
    int max = 10;
    compute(m);
}
```

max : int
min : int

Application 2: Arguments Recommendation

```
private static void compute(int min, int max) {
}

public static void main(String[] args) {
    int min = 5;
    int max = 10;
    compute(m);
}
```

max : int
min : int

Idea

Suggest the most similar potential alternative.

Arguments Recommendation: Results

Analyzed arguments in four applications.

Recommended 1,588 arguments with a **precision of 83%**.

Missing recommendations:

- complex expressions
- literals
- typecasts

Conclusions

Empirical evidence that:

- names of arguments and parameters are similar
- short and generic parameters cause dissimilarity
- dissimilar names can be filtered out
- developers tend to pick the most similar arguments

Two applications:

- anomaly detection
- arguments recommendation

Conclusions

Empirical evidence that:

- names of arguments and parameters are similar
- short and generic parameters cause dissimilarity
- dissimilar names can be filtered out
- developers tend to pick the most similar arguments

Two applications:

- anomaly detection
- arguments recommendation

Promising opportunities for more name-based techniques

Conclusions

Empirical evidence that:

- names of arguments and parameters are similar
- short and generic parameters cause dissimilarity
- dissimilar names can be filtered out
- developers tend to pick the most similar arguments

Two applications:

- anomaly detection
- arguments recommendation

Promising opportunities for more name-based techniques