Detecting Anomalies in the Order of Equally-typed Method Arguments

Michael Pradel, Thomas R. Gross

Department of Computer Science
ETH Zurich
Image to have two different keys...
Image to have two different keys...

Which key fits which keyhole?
int high = ...
int low = ...

void setEndPoints(int, int)
int high = ...
int low = ...

void setEndPoints(int, int)

Image to have two int variables...

Which argument fits which position?
Image to have two int variables...

Which argument fits which position?
Image to have two 
int variables...

Which argument fits 
which position?
Equally-typed Method Arguments

Ordered arguments

Caller → Callee

Problem: Type system doesn’t help
What Can Go Wrong? (1)

```java
int high = ..;
int low = ..;

setEndPoints(?, ?);

void setEndPoints(int i, int j) {
    ...
}
```
What Can Go Wrong? (1)

```java
int high = ..;
int low = ..;

setEndPoints(?, '?);

void setEndPoints(int i, int j) {
    ...
}
```

Bad names for formal parameters

→ Understandability problem
int high = ..;
int low = ..;

setEndPoints(low, high);

void setEndPoints(int high, int low) {
    ...
}

What Can Go Wrong? (2)
int high = ..;
int low = ..;

setEndPoints(low, high);

void setEndPoints(int high, int low) {
    ...
}

Arguments passed in wrong order
→ Correctness problem
What Can Go Wrong? (3)

```c
int high = ..;
int low = ..;

setEndPoints(low, high); // invert end points

void setEndPoints(int high, int low) {
    ...
}
```
What Can Go Wrong? (3)

```java
int high = ..;
int low = ..;

setEndPoints(low, high);  // invert end points

void setEndPoints(int high, int low) {
    ...
}
```

Unexpected but correct argument order → Maintainability problem
Is This Important?

11% of all call sites have two or more equally-typed arguments

I.e., 1 unchecked call per 20 LOC

(DaCapo benchmarks, 1.6 MLOC Java code)
null
Static Anomaly Detection

Program → Argument Naming → Anomalies Examples

Simple yet effective

```c
void m(int a, int b) {} 
m(a, b); 
m(b, a); 
```

(a,b) (a,b) (b,a)

Reverse?!
Static Anomaly Detection

Program → Argument Naming → Anomalies

Program:

```c
void m(int a,
       int b) {}
```

```c
m(a, b);
m(b, a);
```

Examples:

```
(a,b)
(a,b)
(b,a)
```

Reverse?!
Goal: Find examples for argument names

```java
int highEP;
int[] lowEP;
Data data;

setEndPoints(highEP, lowEP[i]);

setEndPoints(data.h, data.low());
```
Goal: Find examples for argument names

```c
int highEP;
int[] lowEP;
Data data;

setEndPoints(highEP, lowEP[i]);

setEndPoints(data.h, data.low());
```

local variable: "highEP"
Argument Name Extraction

Goal: Find examples for argument names

```java
int highEP;
int[] lowEP;
Data data;

setEndPoints(highEP, lowEP[i]);
setEndPoints(data.h, data.low());
```

**local variable:** "highEP"

**array access:** "lowEP"
Argument Name Extraction

Goal: Find examples for argument names

```c
int highEP;
int[] lowEP;
Data data;

setEndPoints(highEP, lowEP[i]);
setEndPoints(data.h, data.low());
```

- **local variable:** "highEP"
- **array access:** "lowEP"
- **field access:** "h"
Goal: Find examples for argument names

```java
int highEP;
int[] lowEP;
Data data;

setEndPoints(highEP, lowEP[i]);

setEndPoints(data.h, data.low());
```

- **local variable:** "highEP"
- **array access:** "lowEP"
- **field access:** "h"
- **method call:** "low"
Parameter Name Extraction

More examples: Formal parameter names

```java
void setEndPoints(int high, int low) {
    ...
}

"high"   "low"
```
Static Anomaly Detection

void m(int a, int b) {}
m(a, b);
m(b, a);

Program → Argument Naming → Anomalies

(a,b) (a,b) (b,a)

Reverse?!
Static Anomaly Detection

Program → Argument Naming Examples → Anomalies

void m(int a, int b) {}
m(a, b);
m(b, a);

(a, b)
(a, b)
(b, a)

m(b, a);
Reverse?!
**Anomaly Detection**

What is an anomaly?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>
Anomaly Detection

What is an anomaly?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
<td>An unusual name is no anomaly</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
<td></td>
</tr>
<tr>
<td>low</td>
<td>high</td>
<td></td>
</tr>
</tbody>
</table>
What is an anomaly?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>

An unusual permutation is an anomaly.
Anomaly Detection

Which permutation is more normal?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
<th>vs.</th>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
<td></td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
<td></td>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td></td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
<td></td>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
Anomaly Detection

How normal is this permutation?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
Anomaly Detection

How normal is this permutation?

$$\text{fitPos}(\text{high}, 1) = \text{similarity}(\text{high}, 1) - \text{similarity}(\text{high}, 2)$$

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
Anomaly Detection

How normal is this permutation?

\[ \text{normality} = \]
\[ \text{fitPos}(\text{high}, 1) = \]
\[ \text{similarity}(\text{high}, 1) \]
\[ - \text{similarity}(\text{high}, 2) \]

\[ \text{fitPos}(\text{high}, 1) = \text{similarity}(\text{high}, 1) - \text{similarity}(\text{high}, 2) \]

\[ \text{fitPos}(\text{low}, 1) = \text{similarity}(\text{low}, 1) - \text{similarity}(\text{low}, 2) \]

\[ \text{fitPos}(\text{low}, 2) = \text{similarity}(\text{low}, 1) - \text{similarity}(\text{low}, 2) \]

\[ \text{fitPos}(\text{high}, 1) = \text{similarity}(\text{high}, 1) - \text{similarity}(\text{high}, 2) \]

Pos. 1 | Pos. 2
---|---
high | low
h | Low
high | low
highEP | lowEP
high | low
## Anomaly Detection

Which permutation is more normal?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>

*normality = 0%*

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>

*normality = 86%*
Anomaly Detection

Which permutation is more normal?

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>

normality = 0%

vs.

<table>
<thead>
<tr>
<th>Pos. 1</th>
<th>Pos. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>h</td>
<td>Low</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>highEP</td>
<td>lowEP</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>

normality = 86%

Anomaly!
Summary of Approach

Program → Argument Naming → Examples → Anomalies
Summary of Approach

No specifications

Program \rightarrow Argument \rightarrow Naming \rightarrow Examples \rightarrow Anomalies
Summary of Approach

Program

No specifications

Argument

Naming

Examples

Anomalies

Programmer-provided hints on argument semantics
Evaluation

- Effectiveness in finding anomalies?
- Anomalies in mature programs?

Setup:
DaCapo benchmarks, 1.6 MLOC Java
Effectiveness

for each call site with equally-typed arguments
for each permutation of the arguments
seed anomaly and try to find it
Effectiveness

For each call site with equally-typed arguments for each permutation of the arguments, seed anomaly and try to find it.

Precision = \frac{\text{# true pos.}}{\text{# true pos.} + \text{# false pos.}}

Recall = \begin{cases} 
1 & \text{if seeded anomaly found} \\
0 & \text{otherwise} 
\end{cases}
Effectiveness

Average over 49K seeded anomalies

Precision: 72%
Recall: 38%
Anomalies in Real Programs

29 anomalies

22 relevant

7 false positives
Anomalies in Real Programs

29 anomalies

22 relevant

1 correctness problem

7 false positives

11 maintainability problems

10 understandability problems
Anomalies in Real Programs

Correctness problem in Eclipse

// call
createAlignment(name, mode,
    Alignment.R_INNERMOST, count, sourceRestart,
    adjust);

// called method
Alignment createAlignment(String name, int mode,
    int count, int sourceRestart, int continuationIndent,
    boolean adjust) { ... }
Anomalies in Real Programs

Correctness problem in Eclipse

// call
createAlignment(name, mode,
    Alignment.R_INNERMOST, count, sourceRestart,
    adjust);

// called method
Alignment createAlignment(String name, int mode,
    int count, int sourceRestart, int continuationIndent,
    boolean adjust) { ... }
Anomalies in Real Programs

Correctness problem in Eclipse

// call
createAlignment(name, mode,
    Alignment.R_INNERMOST, count, sourceRestart, adjust);

// called method
Alignment createAlignment(String name, int mode,
    int count, int sourceRestart, int continuationIndent,
    boolean adjust) { ... }
Anomalies in Real Programs

Understandability problem in Jython

// called method
PyFloat _pow(double value, double iw, PyObject modulo)
Anomalies in Real Programs

Understandability problem in Jython

// called method
PyFloat _pow(double value, double iw, PyObject modulo)

Exponentiation: What is base and what is exponent?
Anomalies in Real Programs

Understandability problem in Jython

// call
_pow(coerce(left), value, null)

// called method
PyFloat _pow(double value, double iw, PyObject modulo)

Exponentiation: What is base and what is exponent?
Anomalies in Real Programs

Maintainability problem in Eclipse

// call
generateOptimizedBoolean(
    currentScope, codeStream,
    falseLabel, trueLabel, valueRequired)

// called method
void generateOptimizedBoolean(
    BlockScope currentScope, CodeStream codeStream,
    Label trueLabel, Label falseLabel, boolean valueRequired)
Anomalies in Real Programs

Maintainability problem in Eclipse

// call
genrateOptimizedBoolean(
    currentScope, codeStream,
    falseLabel, trueLabel, valueRequired)

// called method
void generateOptimizedBoolean(
    BlockScope currentScope, CodeStream codeStream,
    Label trueLabel, Label falseLabel, boolean valueRequired)

Unexpected but correct
Summary of Evaluation

- Effectiveness in finding anomalies?
  72% precision, 38% recall

- Anomalies in mature programs?
  22 relevant among 29 reported
What Does It Cost?

Only input:
Source code of program

Time to analyze 1.6 MLOC:
Less than two minutes
Conclusion

- Cheap technique to find anomalies involving equally-typed arguments
- Simple but effective
- Try it out!
  http://mp.binaervarianz.de/issta2011
Thank you!

michael@binaervarianz.de

http://mp.binaervarianz.de/issta2011