Static Detection of Brittle Parameter Typing

Michael Pradel, Severin Heiniger, and Thomas R. Gross

Department of Computer Science
ETH Zurich
Motivation

```c
void m(A a) { ... }
```
Motivation

void m(A a) { ... }

Compatible types
Motivation

```java
void m(A a) { ... }
```

Expected by method

[Diagram of a class hierarchies with nodes labeled A, B, C, D, E, F, and Object.]
Motivation

```java
void m(A a) { ... }
```

![Diagram showing the hierarchy and expected/unexpected by method]

- **Expected by method**
- **Unexpected by method**
Example: Swing API (1)

```java
JMenu.add(Component)
```
Example: Swing API (1)

```java
JMenu.add(Component)
```

Expected:

- `Component`
- `JComponent`
- `AbstractButton`
- `JTextField`
- `JCheckBox`
- `JSeparator`
- `JMenuItem`
- `JCheckBoxMenuItem`
- `JRadioButtonMenuItem`
- `JMenu`
- `JRadioButtonMenuItem`
Example: Swing API (1)

```java
JMenu.add(Component)
```

Expected

- JButton
- JCheckbox
- JRadioButton
- JMenuItem

Unexpected

- JTextField
- JSeparator
- JCheckboxMenuItem
- JMenuItem
- JMenu
- JRadioButtonMenuItem
Example: Swing API (1)

JMenu.add(Component)

Mismatch between declared type and expected types

Expected

Component

AbstractButton

JTextFile

JCheckBox

JSeparator

Unexpected

JMenuItem

JCheckBoxMenuItem

JMenu

JRadioButtonMenuItem

Mismatch between declared type and expected types
Expected argument type: JCheckBoxMenuItem

Unexpected argument type: JCheckBox

Confirmed as bug: See nTorrent issue #136
The Problem

Brittle parameter type:
Has subtypes that are not expected by callee

Compatible but unexpected arguments:
Subtle errors hidden from the type system
This Talk

Infer from API clients which parameters are brittle and search for unusual argument types
Overview

API clients

Static analysis

Argument type observations

Anomaly detection

Warnings about unexpected types
Overview

API clients

Static analysis

Argument type observations

Anomaly detection

Warnings about unexpected types
Static Analysis

Goal

■ Find types of arguments given to API methods

Two variants

■ Simple: Statically declared type
■ Points-to analysis: Use points-to set of arguments
Argument Type Observations (1)

Client:

```java
Foo foo = new Foo();
JLabel label = new JLabel();
api(foo, label);
```

API:

```java
void api(Object, Component)
```
## Argument Type Observations (1)

### Client:
```
Foo foo = new Foo();
JLabel label = new JLabel();
api(foo, label);
```

### Simple analysis:
- **Foo**
- **JLabel**

### API:
```
void api(Object, Component)
```
Argument Type Observations (2)

Client:
```
Foo foo = new Foo();
Component comp;
if (...) comp = new JLabel();
else comp = new Button();
api(foo, comp);
```

API:
```
void api(Object, Component)
```
Argument Type Observations (2)

Client:

```
Foo foo = new Foo();
Component comp;
if (...) comp = new JLabel();
else comp = new Button();
api(foo, comp);
```

Simple analysis:

API:

```
void api(Object, Component)
```
Client:

```
Foo foo = new Foo();
Component comp;
if (...) comp = new JLabel();
else comp = new Button();
api(foo, comp);
```

Simple analysis:

API:

```
void api(Object, Component)
```
Client:

```
Foo foo = new Foo();
Component comp;
if (...) comp = new JLabel();
else comp = new Button();
api(foo, comp);
```

Points-to analysis:

API:

```
void api(Object, Component)
```

```
Foo
```

```
JLabel, Button
```

```
Component
```

```
Button
```

```
JLabel
```
Client:

```
Foo foo = new Foo();
Component comp;
if (...) comp = new JLabel();
else comp = new Button();
api(foo, comp);
```

Points-to analysis: (with confidence)

API:

```
void api(Object, Component)
```

Foo (1.0)  JLabel (0.5), Button (0.5)
**Argument Type Observations (3)**

**Result:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>api(Object, ...)</td>
<td>Foo (1.0)</td>
</tr>
<tr>
<td>api(..., Component)</td>
<td>Button (0.5)</td>
</tr>
<tr>
<td></td>
<td>JLabel (0.5)</td>
</tr>
</tbody>
</table>
Focus on API Types

1. Generalize client types to API types:

   Foo (1.0) → Button (1.0)

2. Remove all non-API types
Merging Observations

Merge observations from

- Different clients of same API
- Different call sites of same API method

Example:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>api(Object, ...)</td>
<td>Button (1.0)</td>
</tr>
<tr>
<td></td>
<td>JList (1.0)</td>
</tr>
<tr>
<td></td>
<td>Button (0.25)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>api(..., Component)</td>
<td>...</td>
</tr>
</tbody>
</table>
Overview

API clients

Static analysis

Argument type observations

Anomaly detection

Warnings about unexpected types
Overview

API

clients

Static analysis

Argument type observations

Anomaly detection

Warnings about unexpected types
Anomaly Detection

Goal:
Find unexpected arguments given to likely brittle parameters

For each parameter:
1. Build type histogram
2. Search anomalies
**Type Histograms**

\[ \text{JComponent.add(} \text{Component, Object}) \]
Type Histograms

`JComponent.add(Component, Object)`

Not a brittle parameter
Type Histograms (2)

Container.add(Component, Object)
Type Histograms (3)

```java
JMenu.add(Component)
```
Type Histograms (3)

Brittle parameter, two unexpected arguments

```
JMenu.add(Component)
```
Type Histograms (4)

JScrollPane.<init>(Component, int, int)
Type Histograms (4)

`JScrollPane.<init>(Component, int, int)`

Little information
Finding Anomalies

- Initial assumption: Each observation = potential anomaly
- Filters to remove false warnings
- If all filters passed: Warning about unexpected argument
Filtering
Filtering

Initial assumption:
All observations are anomalies
Filtering
Filtering

Whole-histogram filter:
Minimum number of observations
Filtering
Whole-histogram filter: Maximum number of types
Filtering
Filtering

Per-type filter:
Minimum Confidence Drop
(Does the argument type deviate from an otherwise accepted rule?)
Filtering

Whole-histogram filter:
Maximum percentage of anomalies
Filtering

Result:
Two anomalies
(both are bugs)
Summary

API clients

Static analysis

 Argument type observations

Anomaly detection

Warnings about unexpected types
Evaluation

- 21 programs (650 kLoC)
- AWT/Swing API
- Bugs - code smells - false positives
Example: Bug in jEdit

class FilteredListModel extends AbstractListModel {
    void setFilter(String filter) {
        Runnable runner = new Runnable() {
            public void run() {
                fireContentsChanged(this, 0, getSize() - 1);
            }
        };
    }
}

Confirmed as a bug and fixed within a day. See bug #3477759.
Example: Bug in jEdit

class FilteredListModel extends AbstractListModel {
    void setFilter(String filter) {
        Runnable runner = new Runnable() {
            public void run() {
                fireContentsChanged(this, 0, getSize() - 1);
            }
        };
    }
}

Confirmed as a bug and fixed within a day. See bug #3477759.
Example: Bug in jEdit

```java
class FilteredListModel extends AbstractListModel {
    void setFilter(String filter) {
        Runnable runner = new Runnable() {
            public void run() {
                fireContentsChanged(this, 0, getSize()-1);
            }
        };
    }
}
```

Confirmed as a bug and fixed within a day. See bug #3477759.
Example: Bug in jEdit

Declared: Object
Expected: ListModel
Here: Runnable

Confirmed as a bug and fixed within a day. See bug #3477759.

```java
class FilteredListModel extends AbstractListModel {
    void setFilter(String filter) {
        Runnable runner = new Runnable() {
            public void run() {
                fireContentsChanged(this, 0, getSize() - 1);
            }
        };
    }
}
```
Example: Bug in JFtp

...

JScrollPane scrollPane = new JScrollPane(list);
container.add(new JScrollPane(scrollPane));

...

Confirmed as a bug and fixed. See bug #3484625.
Example: Bug in JFtp

... JScrollPane scrollPane = new JScrollPane(list);
container.add(new JScrollPane(scrollPane));
...

- **Declared:** Component
- **Expected:** JList, JTextArea, ...
- **Here:** JScrollPane

Confirmed as a bug and fixed. See bug #3484625.
Characteristics of Issues

Common to all issues found:

- Subtle problems (no exception etc.)
- Visual glitches
- GUI misbehavior

Hard to find with traditional testing
Precision

Default filtering: 47%

5 bugs  4 code smells  10 false positives
Precision

Default filtering: 47%

- 5 bugs
- 4 code smells
- 10 false positives

Recall-focused filtering: 11%

- 11 bugs
- 4 code smells
- 140 false positives
Recall

Randomly seeded bugs
(for known brittle parameters)

- Default filtering: 83%
- Recall-focused filtering: 94%
Benefits of using points-to analysis

- Find two more bugs
  (recall-focused configuration, original programs)

- Increased precision: 76% → 83%
  (default configuration, seeded bugs)
Influence of Points-to Analysis

Benefits of using points-to analysis

- Find two more bugs
  (recall-focused configuration, original programs)

- Increased precision: 76% → 83%
  (default configuration, seeded bugs)

Beneficial but not crucial
Performance

Good performance for automatic analysis

- All 21 programs: 23 minutes
- 99.9% of time: Static analysis

Intel Core 2 Duo with 3.16 GHz, 2.5 GB memory
Conclusion

Powerful analysis that finds subtle errors where traditional testing fails

Lessons learned:

- **Brittle parameters**: Real problem that deserves attention
- **Simple analysis**: Effective in practice
- **Many-client analysis**: Key to success
Implementation and experimental data:
http://mp.binaervarianz.de/issta2012/

Static Detection of Brittle Parameter Typing
Michael Pradel, Severin Heiniger, and Thomas R. Gross