Making Software Refactorings Safer

Anna Maria Eilertsen
@Sowhow

Supervised by: Anya Bagge\textsuperscript{1} Volker Stolz \textsuperscript{2}

\textsuperscript{1}Inst. for Informatikk, Universitetet i Bergen

\textsuperscript{2}Inst. for Data- og Realfag, Høgskolen i Bergen
Norway

July 12, 2016

The results of this thesis has been accepted to the ISOLA\textsuperscript{1} conference as a paper.

\textsuperscript{1}http://www.isola-conference.org/isola2016/
"Behaviour preserving program transformation"
Software Refactoring Tools

Ambiguous return value: Selected block modifies more than one local variable used in subsequent code. Affected variables are:

String s
String s1

Making Software Refactorings Safer

July 12, 2016
Unsafe Refactorings

“The primary risk is regression, mostly from misunderstanding subtle corner cases in the original code and not accounting for them in the refactored code.”

– interviewee, Microsoft developer, Kim et al., 2012
Unsafe Refactoring Example

Extract Local Variable

In Java/Eclipse:

Before

```java
public void f() {
    x.n();
    setX();
    x.n();
}
```

After

```java
public void f() {
    X temp = x;
    temp.n();
    setX();
    temp.n();
}
```
An analysing problem

```java
public void f() {
    X temp = x;
    temp.n();
    setX();
    temp.n();
}
```

Solution:

```java
assert temp == x;
```
Extract Local Variable

Simplified example:

```java
public class C {
    public X x = new X();
    { //initializer
        x.myC = this;
    }

    public void f(){
        x.n();
        x.m();
        x.n();
    }
}
```

```java
public class X{
    public C myC;

    public void m(){
        myC.x = new X();
    }

    public void n(){
        System.out.println(
            this.hashCode());
    }
}
```

Output:
1735600054
21685669
Extract Local Variable

Refactored:

```java
public class C {
    public X x = new X();
    { //initializer
        x.myC = this;
    }

    public void f(){
        X temp = x;
        temp.n();
        temp.m();
        temp.n();
    }
}
```

```java
public class X{
    public C myC;

    public void m(){
        myC.x = new X();
    }

    public void n(){
        System.out.println(this.hashCode());
    }
}
```

Output:
1735600054
1735600054
Extract Local Variable

With dynamic checks:

```java
public class C {
    public X x = new X();
    { //initializer
        x.myC = this;
    }
    public void f(){
        X temp = x;
        assert temp == x;
        temp.n();
        assert temp == x;
        temp.m();
        assert temp == x;
        temp.n();
    }
}

public class X{
    public C myC;
    public void m(){
        myC.x = new X();
    }
    public void n(){
        System.out.println(this.hashCode());
    }
}
```

Output:
1735600054
Exception in thread "main" java.lang.AssertionError
Extract And Move Method

A similar problem:

```java
public class C {  
    public X x = new X();  
    { //initializer  
        x.myC = this;  
    }  
    public void f(){  
        x.bar(this);  
    }  
}
```

```java
public class X{  
    ...  
    void bar(C c){  
        this.n();  
        assert this == c.x;  
        this.m();  
        assert this == c.x;  
        this.n();  
    }  
}
```

Similar how?
Evaluate `x` once.
Refer to that value by `this`
Substitute every occurrence of `x` with `this`
Experiment: Case study

Case: Eclipse JDT UI source code

Experiment:

- Execute our modified refactorings on Eclipse JDT UI project
- Run Eclipse test suite
- Look for triggered asserts
- Profit!!

Need custom automated refactoring tool.
## Experiment: Results

<table>
<thead>
<tr>
<th></th>
<th>Extract Local Variable</th>
<th>Extract and Move Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executed refactorings</td>
<td>4538</td>
<td>755</td>
</tr>
<tr>
<td>Total number of asserts</td>
<td>7665</td>
<td>610</td>
</tr>
<tr>
<td>Resulting compile errors ✗</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>Successful Tests ✔</td>
<td>2392</td>
<td>2151</td>
</tr>
<tr>
<td>Unsuccessful Tests ✗</td>
<td>4</td>
<td>245</td>
</tr>
<tr>
<td>Asserts triggered</td>
<td>2 / 136*</td>
<td>0</td>
</tr>
</tbody>
</table>

* 136 instances of the same 2 assert statements
**Discussion**

<table>
<thead>
<tr>
<th></th>
<th>Extract Local Variable</th>
<th>Extract and Move Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executed refactorings</td>
<td>4538</td>
<td>755</td>
</tr>
<tr>
<td>Total number of asserts</td>
<td>7665</td>
<td>610</td>
</tr>
<tr>
<td>Resulting compile errors</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>Successful Tests</td>
<td>2392</td>
<td>2151</td>
</tr>
<tr>
<td>Unsuccessful Tests</td>
<td>4</td>
<td>245</td>
</tr>
<tr>
<td>Asserts triggered</td>
<td>2 / 136</td>
<td>0</td>
</tr>
</tbody>
</table>

**Take-away and questions:**

- Dynamic preconditions can be useful!
- Assert statements are incomplete.
- Show or hide the asserts from the programmer?
- Is reference equivalence too strict?

Thank you!

---

* Art with face is from Hyperbole and a Half

Anna Maria Eilertsen @sowhow Making Software Refactorings Safer July 12, 2016 13 / 14
Experiment: Development

Eclipse refactoring plug-in

- Modify Eclipse’s refactorings to introduce asserts
  - Extract Local Variable
  - Extract And Move Method
- Automate refactoring process
  - Execute on Java project
  - One refactoring per method
- Custom heuristic for finding refactoring targets