A Study on the Relation Between Antipatterns and the Cost of Class Unit Testing

A. Sabané, M. Di Penta, G. Antoniol, Y-G. Guéhéneuc

École Polytechnique de Montréal, Québec, Canada
University of Sannio, Benevento, Italy

March 7th, 2013
Outline

- Motivations
- Goal
- Empirical Study
- Results
- Other Findings
- Conclusion
- Ongoing Project
What are Antipatterns?

Identify poor solutions to recurring design problems [Brown et al.]

Some causes

- Time-market pressure
- Insufficient knowledge and or experience
- Misapplication of some design patterns
Exemple of Antipatterns

- Blob (God Class): Large and complex class that centralises the behaviour of a portion of a system
- Lazy Class: Class with few responsibilities
- Spaghetti Code: Class declaring long methods with no parameters and using global variables
Motivations

Impact of Antipatterns...

- Understandability (Abbes et al.)
- Fault proneness (Khomh et al.)
- Maintainability (Deligiannis et al.)
- Change proneness (Olbrich et al.)
Motivations

What about Testability and Testing?

- Understandability
  - Abbes et al.
- Maintainability
  - Deligiannis et al.
- Change proneness
  - Olbrich et al.
  - Khomh et al.
- Fault proneness
  - Khomh et al.

Testability?
Goal of the Study

Study the impact of antipatterns on class testability
Class Testability Measure

What is Testability?

- Degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met [IEEE Glossary]
- Testing effort [Bache et al.]

How to Measure Testability?

- Metric-Based approaches [Bruntink et al.]
- number of test cases required to satisfy a given coverage criterion [Bache et al.]
MaDUM

What is it?

- Unit testing technique for object oriented programs [Bashir et al.]
- Based on data slices
- Core idea: The correctness of a class is equivalent to the correctness of all its slices
- Focus on the test of methods interactions
MaDUM

Empirical Study

Class Testability Measure

MaDUM

Objects

Subjects

RQ1: Assess AP class impact on testing effort?
RQ2: Which APs are the most expensive
RQ3: Cost-effectiveness of APs testing

MaDUM Process Description

For each slice, test:

- reporters
- constructors
- interactions among transformers
  in each constructor context ([cl][ct]c)
- other methods

```java
class Person{
    String name;
    int age;

    public Person()
    {
        this.age=0;
        this.name="";
    }

    public Person(String aName, int anAge)
    {
        this.age=anAge;
        this.name=aName;
    }

    public getAge()
    {
        return this.age;
    }

    public getName()
    {
        return this.name;
    }

    public void print()
    {
        System.out.println("Name: "+this.name+" Age: "+this.age);
    }

    public update(String aName, int anAge)
    {
        this.age=anAge;
        this.name=aName;
    }
}
```
## MaDUM

**Introduction**

**Empirical Study**

**Results**

**Conclusion**

**Class Testability Measure**

**MaDUM**

**Objects**

**Subjects**

RQ1: Assess AP class impact on testing effort?

RQ2: Which APs are the most expensive

RQ3: Cost-effectiveness of APs testing

<table>
<thead>
<tr>
<th>Method</th>
<th>name</th>
<th>age</th>
<th>getName()</th>
<th>getAge()</th>
<th>update(String, int)</th>
<th>print()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person()</td>
<td>c</td>
<td>c</td>
<td>r</td>
<td>t</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Person(String, int)</td>
<td>c</td>
<td>c</td>
<td>r</td>
<td>t</td>
<td>o</td>
<td></td>
</tr>
</tbody>
</table>

A. Sabané, M. Di Penta, G. Antoniol, Y-G. Guéhéneuc

March 7th, 2013
Why MaDUM?

- Specific to OO programs
- Not require specific design documentation
- Possibility to compute the number of required test cases
- Upper-bound for testing cost
Objects

- 4 systems: Ant, ArgoUML, Checkstyle & JFreechart
- Systems belong to different application domains, have bug-fixing data available and have been used in previous studies [Abbes et al.], [Bruntink et al.]
Subjects

- 13 antipatterns
- Studied in previous studies [Khomh et al.], [Abbes et al.],
- Detection tool: Decor [Moha et al.]

<table>
<thead>
<tr>
<th>Name (Abbr)</th>
<th>Ant</th>
<th>ArgoUML</th>
<th>CheckStyle</th>
<th>JFreeChart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antipattern classes</td>
<td>452</td>
<td>901</td>
<td>161</td>
<td>245</td>
</tr>
<tr>
<td>No Antipattern (None)</td>
<td>297</td>
<td>376</td>
<td>99</td>
<td>233</td>
</tr>
</tbody>
</table>
RQ1: Assess AP class impact on testing effort?

How large is the MaDUM test suite for classes participating in APs compared to that of other classes?
RQ2: Which APs are the most expensive

How does the size of the MaDUM test suite vary among classes participating in different kinds of APs?
RQ3: Cost-effectiveness of APs testing

What is the potential cost-benefit achieved when focusing testing on APs, as opposed to other classes?
Results
RQ1: How large is the MaDUM test suite for classes participating in APs compared to that of other classes?
RQ1: How large is the MaDUM test suite for classes participating in APs compared to that of other classes?

<table>
<thead>
<tr>
<th>System</th>
<th>Mean TCs AP</th>
<th>Mean TCs NAP</th>
<th>p-Value</th>
<th>Cliff’s d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>18</td>
<td>9</td>
<td>&lt; 0.01</td>
<td>0.23 (Small)</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>10</td>
<td>3</td>
<td>&lt; 0.01</td>
<td>0.35 (Medium)</td>
</tr>
<tr>
<td>CheckStyle</td>
<td>9</td>
<td>6</td>
<td>= 0.01</td>
<td>NA</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>26</td>
<td>13</td>
<td>&lt; 0.01</td>
<td>0.22 (Small)</td>
</tr>
</tbody>
</table>
RQ2: How does the size of the MaDUM test suite vary among classes participating in different kinds of APs?
RQ2: How does the size of the MaDUM test suite vary among classes participating in different kinds of APs?
RQ3: What is the potential cost-benefit achieved when focusing testing on APs, as opposed to other classes?

It is cost-effective to analyze/test APs classes with a higher priority than other classes.
Other Findings: Refactoring for Reducing Testing Cost

Effect of traditional refactoring on testing cost?
### Other Findings: Refactoring for Reducing Testing Cost

<table>
<thead>
<tr>
<th>Class (system)</th>
<th>Type</th>
<th>TRS</th>
<th>Before refactoring TCs</th>
<th>TRS</th>
<th>After refactoring TCs</th>
<th>TRS</th>
<th>TCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenFilter (Ant)</td>
<td>CDSBP</td>
<td>5</td>
<td>263</td>
<td>2</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PropPanel (ArgoUML)</td>
<td>Blob</td>
<td>5</td>
<td>271</td>
<td>3</td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BooleanExpressionComplexityCheck</td>
<td>LPL</td>
<td>6</td>
<td>732</td>
<td>5</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Checkstyle)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AxisState (JFreeChart)</td>
<td>NAP</td>
<td>5</td>
<td>248</td>
<td>1</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamicTimeSeriesCollection (JFreeChart)</td>
<td>Blob</td>
<td>4</td>
<td>208</td>
<td>2</td>
<td>122</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Findings

- Most of APs classes are expensive for unit testing
- APs classes with excess of responsibilities require more testing effort than other
- Prioritize the testing of APs classes can be more cost-effective
- Specific refactoring can help to reduce testing cost
Future Work

- Study more systems
- Investigate the symptoms that make APs classes more expensive
- Study the impact of APs on testing effort from other aspects
- Investigate specific refactoring for testability
Thank you very much for your attention!