Design Smells in the Context of Agile Software Development

Abdou Maiga

DIRO, Université de Montréal, Québec, Canada
Ph.D. Defense

Nov. 2012
Agile Software Development (ASD) Context

- ASD is a set of **collaborative, iterative, and incremental** development methods [Beck, 2001]
- In general we have **inadequation** between **documentation** and **source code**

⇒

- The **source code** is the most important **source of information** to **evolve, change and maintain** the system.
- **Maintenance activities** have become very **expensive**. They reach more than **70%** of the overall costs of software development [Pressman, 2001]
Context

- **Procedural thinking in object oriented language**
Context

- **Procedural thinking in object oriented language**
- **Impact** of this design on comprehension?
Context

- **Procedural** thinking in object oriented language
- **Impact** of this design on comprehension?
- **Impact** of this design on fault-fixing activities?
Context

- **Procedural** thinking in object oriented language
- **Impact** of this design on comprehension?
- **Impact** of this design on fault-fixing activities?
- How to **accurately detect** this design in ASD context?
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
Outline

1. Introduction

2. Related Works and Contributions

3. Impact of Design Smells on Comprehension

4. Impact of Design Smells on Fault-fixing

5. SMURF

6. Conclusion and Perspectives
Design Smells in Software Systems

Design smells are poor solutions to recurring design or implementation problems [Webster, 1995]

- They are generally the result of misuse of the object-oriented paradigm and–or design patterns [Brown, 1998]

Design smells are present in software systems due to

- The time-to-market
- The lack of understanding
- The developers’ experience

Developers cannot always follow standard designing and coding techniques
Motivation

Quantitative evidences on the relation between design smells, comprehension, and fault-fixing activities

- are important for practitioners to help developers reduced change costs
- help to reduce maintenance costs and improve software quality
- help practitioners to take rational decisions about how dealing with design smells
- help practitioners justify the removal of design smells

⇒ Empirical software engineering thesis
There is an impact of design smells on source code comprehension and fault-fixing activities and we can provide a tool for accurate incremental design smells detection in ASD context.
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
## Related Works – Design Smells Impact

<table>
<thead>
<tr>
<th>Work</th>
<th>Limitations</th>
</tr>
</thead>
</table>
| [Deligiannis, 2003]  
Blob affect evolution of design structures and use of inheritance | No evidence of impact of DS on comprehension |
| [Du Bois, 2006]  
Decomposition of God classes into collaborating classes using refactorings can improve comprehension | No evidence of impact of DS on comprehension |
| [Olbrich, 2009]  
Blob and Shotgun Surgery are more change-prone than others | No evidence of impact of DS on Fault-fixing activities |
| [Abbes, 2011]  
Blob, Spaghetti Code design smells impedes developers’ performance | Could not find any impact from one occurrence of blob or spaghetti code |
| [Khomh 2012]  
Design smell classes are more change-prone and fault-prone than others | No link between fault-fixing activities and design smells |
Related Work – Limitations

Design Smells Impact

- No evidence of the impact of DS on comprehension
  ⇒ Controlled experiments on impact of DS on comprehension
Related Work – Limitations

**Design Smells Impact**

- No evidence of the impact of DS on comprehension
  ⇒ Controlled experiments on impact of DS on comprehension

- No impact found for single occurrence of DS on comprehension
  ⇒ Experiments with combination of DS
Related Work – Limitations

**Design Smells Impact**

- **No evidence** of the impact of DS on **comprehension**
  - ⇒ **Controlled experiments** on impact of DS on comprehension
- **No impact** found for **single occurrence** of DS on comprehension
  - ⇒ **Experiments with combination** of DS
- **No evidence** of impact of DS on fault-fixing activities
  - ⇒ **Empirical study** on the impact of DS on fault-fixing activities
### Related Work – Design Smells Detection

<table>
<thead>
<tr>
<th>Approach</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Travassos, 99] Manual inspection</td>
<td>Not applicable on large systems</td>
</tr>
<tr>
<td>[Marinescu, 2004] Detection strategies based on metrics</td>
<td>Validated on one system</td>
</tr>
<tr>
<td>[Munro, 2005] Heuristics based on metrics</td>
<td>Specific to code smells</td>
</tr>
<tr>
<td>[Langelier, 2005], [Dhambri, 2008] Visualization approach</td>
<td>High human intervention</td>
</tr>
<tr>
<td>[AliKacem, 2006] Rules based on metrics detect violations quality principles</td>
<td>Not directly applicable to design smells detection</td>
</tr>
<tr>
<td>[Moha, 2010] Based on set of metrics rules and relations between classes</td>
<td>Definition of rules cards are manual</td>
</tr>
<tr>
<td>[Khomh, 2011] Probabilistic approach based on Bayesian Belief Networks</td>
<td>Extensive knowledge</td>
</tr>
</tbody>
</table>

- Manual inspection
- Detection strategies based on metrics
- Heuristics based on metrics
- Visualization approach
- Rules based on metrics detect violations quality principles
- Based on set of metrics rules and relations between classes
- Probabilistic approach based on Bayesian Belief Networks
Related Work – Limitations

Design Smells Detection

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require extensive knowledge of DS
- ⇒ Machine learning technique: SVM
Related Work – Limitations

Design Smells Detection

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require extensive knowledge of DS
  ⇒ Machine learning technique: SVM
- The choice of thresholds
  ⇒ Derived automatically from the application of SVM
Related Work – Limitations

**Design Smells Detection**

The previous approaches are mostly based on the use of code/design quality metrics and thresholds to identify DS.

- They require **extensive knowledge** of DS
  - ⇒ **Machine learning** technique: SVM

- The choice of **thresholds**
  - ⇒ Derived **automatically** from the application of SVM

- They did not integrate the **subjectivity** of the practitioners
  - ⇒ Integrating user’s **feedback**
Contributions

1. Quantitative evidences of impact of DS on source code comprehension of systems
   ⇒ Help developers to take rational decisions about their design quality

2. Quantitative evidences of impact of DS on fault-fixing activities
   ⇒ Help developers to deal with DS

3. Incremental detection approach based on SVM and using users’ feedback
   ⇒ Make developers proactive and help to reduce maintenance costs
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
Comprehension: central element for the effectiveness of software maintenance and evolution

Design smells are conjectured to negatively impact comprehension

Following the design in [Abbes, 2011], we conducted two experiments to assess the impact of co-occurrences of DS

In [Abbes, 2011], single occurrence of DS did not impact comprehension but co-occurrences of Blob and Spaghetti Code impact comprehension

We conducted a quasi quasi-replication to analyse whether co-occurrences of same DS, Blob or Spaghetti Code, in a single system, impact comprehension.
Study Design

Our design is like [Abbes, 2011]: 2 * 3 factorial design

- Three different systems (per experiment), each with two possibilities (with or without co-occurrences of DS)
- For each combination, we prepare a set of comprehension questions, making up treatments.
Research Questions

<table>
<thead>
<tr>
<th>RQ1</th>
<th>What is the impact of co-occurrences of Blob on code source comprehension?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2</td>
<td>What is the impact of co-occurrences of Spaghetti Code on code source comprehension?</td>
</tr>
</tbody>
</table>

Hypothesis

For RQs, we formulate following null hypothesis

- $H_{01_{Blobs}}$: No statistically significant difference between the subjects’ average correct answers
- $H_{02_{Blobs}}$: No statistically significant difference between the subjects’ average time spent
- $H_{03_{Blobs}}$: No statistically significant difference between the subjects’ average effort
Objects of the Study

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Systems</th>
<th>Classes</th>
<th>SLOCs</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Azureus v2.3.0.6</td>
<td>1,449</td>
<td>191,963</td>
<td>2005</td>
</tr>
<tr>
<td></td>
<td>iTrust v11.0</td>
<td>565</td>
<td>21,901</td>
<td>2010</td>
</tr>
<tr>
<td></td>
<td>SIP v1.0</td>
<td>1,771</td>
<td>486,966</td>
<td>2010</td>
</tr>
<tr>
<td>2</td>
<td>ArgoUml v0.20</td>
<td>1,230</td>
<td>113,017</td>
<td>2006</td>
</tr>
<tr>
<td></td>
<td>JHotDraw v5.4b2</td>
<td>484</td>
<td>72,312</td>
<td>2004</td>
</tr>
<tr>
<td></td>
<td>Rhino v1.6R5</td>
<td>108</td>
<td>48,824</td>
<td>2009</td>
</tr>
</tbody>
</table>

Table: Object Systems

Subjects of the Study

- **59** anonymous subjects
- M.Sc. and Ph.D. students at ÉPM and UdeM
- M.Sc. and Ph.D. students at Carleton University
- Subjects were volunteers and could withdraw at any time, for any reason
### Independent Variables

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$Has2b$ : Boolean (presence or not) of co-occurences of Blob</td>
</tr>
<tr>
<td>2</td>
<td>$Has2SC$ : Boolean (presence or not) of co-occurences of Spaghetti Code</td>
</tr>
</tbody>
</table>

### Mitigating Variables

Three mitigating variables could possibly impact the measures of the dependent variables:

- Subject’s knowledge level in **Java**.
- Subject’s knowledge level of **Eclipse**.
- Subject’s knowledge level in **software engineering**.
Dependent Variables

Dependent variables measure subjects’ performance

- Subjects’ **effort** using the NASA Task Load Index (TLX)
- **Time spent** to answer the comprehension questions
- Percentage of **correct answers**
Analysis Method

- Non-parametric Mann-Whitney test for RQ1 and RQ2
- Non-parametric effect size measure Cliff’s d
- Kruskal-Wallis one-way analysis of variance to analyse the impact of the mitigating variables
## Results

### Table: Mann-Whitney tests and Cliff’s $d$ results for 2 Blob

<table>
<thead>
<tr>
<th></th>
<th>M.-W. $p$</th>
<th>Cliff’s $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$&lt;0.01$</td>
<td>0.87 (Large)</td>
</tr>
<tr>
<td>Correct Answers</td>
<td>$&lt;0.01$</td>
<td>0.53 (Large)</td>
</tr>
<tr>
<td>Effort</td>
<td>$&lt;0.01$</td>
<td>0.73 (Large)</td>
</tr>
</tbody>
</table>

### Table: Mann-Whitney tests and Cliff’s $d$ results for 2 SC

<table>
<thead>
<tr>
<th></th>
<th>M.-W. $p$</th>
<th>Cliff’s $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$&lt;0.01$</td>
<td>0.93 (Large)</td>
</tr>
<tr>
<td>Correct Answers</td>
<td>$&lt;0.01$</td>
<td>0.93 (Large)</td>
</tr>
<tr>
<td>Effort</td>
<td>$&lt;0.01$</td>
<td>0.55 (Large)</td>
</tr>
</tbody>
</table>
Impact of Mitigating Variables

Kruskal-Wallis test shows no effect on the results of the mitigating variables

1. Java Knowledge
2. Eclipse Knowledge
3. Software Engineering Knowledge
Subjects working with systems with **co-occurrences** of DS

- Put **more effort** in doing comprehension tasks
- Have **few percentages** of correct answers
- Spend **more time**

**Co-occurrences** of DS are sufficient to impact developer’s comprehension of source code
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
Context

We know that

- Design smells are prevalent in systems
- Several studies have shown that design smells negatively impact maintenance

However, none of them empirically investigated the impact of design smells on fault-fixing activities

⇒

Gather quantitative evidence on the relation between design smells, faults, and the developers’ effort to fix the faults
### Research Questions

<table>
<thead>
<tr>
<th>RQ1</th>
<th>Relation between number of DS in a class and its fault-proneness?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ2</td>
<td>Relation between duration of fixing period and the presence of DS?</td>
</tr>
<tr>
<td>RQ3</td>
<td>Relation between number of elements (fields and methods) impacted by fault-fixes and the presence of DS?</td>
</tr>
<tr>
<td>RQ4</td>
<td>Relation between entropy of fault-fixes and the presence of DS?</td>
</tr>
<tr>
<td>RQ5</td>
<td>Relation between fault-fixes and the number of occurrences of DS before and after fault-fixes?</td>
</tr>
</tbody>
</table>
### Independent Variables

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td><code>HasF</code>: (Boolean) presence or not of fault in a class</td>
</tr>
<tr>
<td>(RQ2, RQ3, RQ4)</td>
<td><code>HasDS_t</code>: (Boolean) presence or not of DS in a fault $t$</td>
</tr>
<tr>
<td>RQ5</td>
<td>$DS_{t,b}$: Number of occurrences of DS in a fault $t$ before the fault-fix</td>
</tr>
</tbody>
</table>
## Dependent Variables

<table>
<thead>
<tr>
<th>RQ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>$DS_i$: Number of occurrences of DS in class $i$</td>
</tr>
<tr>
<td>RQ2</td>
<td>$D_t$: Duration of fixing period (number of days between date of report of the fault day of the fix)</td>
</tr>
<tr>
<td>RQ3</td>
<td>$Nbr_{Elts}$: Number of elements (methods and fields) impacted by a fault-fix</td>
</tr>
<tr>
<td>RQ4</td>
<td>$E_t$: Entropy of fault-fix</td>
</tr>
<tr>
<td>RQ5</td>
<td>$DS_{t,a}$: Number of occurrences of DS in a fault $t$ after the fault-fix</td>
</tr>
</tbody>
</table>
Analysis Methods

We use the following analysis methods:

- Non-parametric Mann-Whitney test
- Non-parametric effect size measure Cliff’s $d$
- Multivariate regression analyses
### Results

<table>
<thead>
<tr>
<th>RQ</th>
<th>ArgoUML</th>
<th>Eclipse</th>
<th>Mylyn</th>
<th>Rhino</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1 (Fault-prone)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RQ2 (Duration)</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>RQ3 (Elements)</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RQ4 (Entropy)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RQ5 (Nbr DS)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Results

Quantitative evidence of impact of DS on fault-fixing activities

- Faulty classes have higher number of DS
- Faults involving DS take more time to be fixed and their fixes impact more fields and methods with higher entropy
- Fixing faults reduces the number of DS

⇒ Help developers to deal with DS
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
Context

We know from the study above that

- DS impact code source comprehension
- DS impact fault-fixing activities
- Automatically remove DS can improve the quality of source code and reduce maintenance cost

However previous approaches have some limitations

- They have **limited** precision and recall
- They require **extensive knowledge** of DS
- They cannot be applied on **subsets of systems**
- They did not integrate the **subjectivity** of the practitioners
- They are not **incremental**.
Contributions: SMURF

- SMURF approach to detect DS using SVM and practitioners’ feedback
  - Classical learning systems with same precision as neural networks or higher
  - Can integrate users’ feedback
  - SVM is robust to noises [Taylor and Cristianini, 2004;Rychetsky, 2001]
Research Questions

<table>
<thead>
<tr>
<th>RQ1,RQ2</th>
<th>How does the accuracy of SMURF compare with that of DETEX and BDTEX?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ3</td>
<td>How does the accuracy of SMURF change when used in inter-system configuration?</td>
</tr>
<tr>
<td>RQ4</td>
<td>How does the accuracy of SMURF change when using users’ feedback?</td>
</tr>
</tbody>
</table>
### Objects

**Table: Description of the objects of the study**

<table>
<thead>
<tr>
<th>Names</th>
<th>Versions</th>
<th># Lines of Code</th>
<th># Classes</th>
<th># Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgoUML</td>
<td>0.19.8</td>
<td>113,017</td>
<td>1,230</td>
<td>67</td>
</tr>
<tr>
<td>Azureus</td>
<td>2.3.0.6</td>
<td>191,963</td>
<td>1,449</td>
<td>546</td>
</tr>
<tr>
<td>Xerces</td>
<td>2.7.0</td>
<td>71,217</td>
<td>513</td>
<td>162</td>
</tr>
</tbody>
</table>
The subjects of our study are the following four DS

- Blob
- Functional Decomposition (FD)
- Spaghetti Code (SC)
- Swiss Army Knife (SAK)

We use these four DS because they well known and previously studied [Moha, 2010; Kohmh 2011]
Subsets of System: RQ1

Table: Precision of SMURF in subsets

<table>
<thead>
<tr>
<th>DS</th>
<th>ArgoUML</th>
<th>Azureus</th>
<th>Xerces</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>97.09</td>
<td>97.32</td>
<td>95.51</td>
<td>96.64</td>
</tr>
<tr>
<td>FD</td>
<td>70.68</td>
<td>72.01</td>
<td>66.93</td>
<td>69.87</td>
</tr>
<tr>
<td>SC</td>
<td>85.00</td>
<td>88.00</td>
<td>86.00</td>
<td>86.33</td>
</tr>
<tr>
<td>SAK</td>
<td>75.46</td>
<td>84.54</td>
<td>80.76</td>
<td>80.25</td>
</tr>
</tbody>
</table>

Table: Recall of SMURF in subsets

<table>
<thead>
<tr>
<th>DS</th>
<th>ArgoUML</th>
<th>Azureus</th>
<th>Xerces</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>84.09</td>
<td>91.33</td>
<td>95.29</td>
<td>90.24</td>
</tr>
<tr>
<td>FD</td>
<td>57.50</td>
<td>84.28</td>
<td>70.00</td>
<td>70.59</td>
</tr>
<tr>
<td>SC</td>
<td>71.00</td>
<td>89.00</td>
<td>86.00</td>
<td>82.00</td>
</tr>
<tr>
<td>SAK</td>
<td>77.14</td>
<td>85.71</td>
<td>75.50</td>
<td>79.45</td>
</tr>
</tbody>
</table>
**Subsets of System: RQ1**

Figure: Trends of precision and recall when increasing the size of subset for SC in Xerces
Study Results

Complete System: RQ12

Table: Total recovered true occurrences of Blob by DETEX and SMURF

<table>
<thead>
<tr>
<th></th>
<th>DETEX</th>
<th>SMURF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgoUML</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>Azureus</td>
<td>38</td>
<td>48</td>
</tr>
<tr>
<td>Xerces</td>
<td>39</td>
<td>55</td>
</tr>
<tr>
<td>Total</td>
<td>102</td>
<td>143</td>
</tr>
</tbody>
</table>

- DETEX is not applicable on subsets of systems while SMURF has good precision and recall.
- On entire systems, SMURF detects more true occurrences of Blob than DETEX.
The accuracy of SMURF compared with that of BDTeX

Figure: Trends in the increase of precision and recall when decreasing the probability of being a design smell for Blob in Xerces
The accuracy of SMURF in inter-systems configuration

Table: Precision of SMURF in inter-systems

<table>
<thead>
<tr>
<th></th>
<th>ArgoUML (%)</th>
<th>Azureus (%)</th>
<th>Xerces (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>92.00</td>
<td>96.00</td>
<td>89.00</td>
</tr>
<tr>
<td>FD</td>
<td>57.00</td>
<td>62.00</td>
<td>36.00</td>
</tr>
<tr>
<td>SC</td>
<td>77.00</td>
<td>74.00</td>
<td>91.00</td>
</tr>
<tr>
<td>SAK</td>
<td>56.00</td>
<td>73.00</td>
<td>90.00</td>
</tr>
</tbody>
</table>

Table: Recall of SMURF in inter-systems

<table>
<thead>
<tr>
<th></th>
<th>ArgoUML (%)</th>
<th>Azureus (%)</th>
<th>Xerces (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>62.00</td>
<td>48.00</td>
<td>94.00</td>
</tr>
<tr>
<td>FD</td>
<td>40.00</td>
<td>100.00</td>
<td>20.00</td>
</tr>
<tr>
<td>SC</td>
<td>96.00</td>
<td>88.00</td>
<td>91.00</td>
</tr>
<tr>
<td>SAK</td>
<td>68.00</td>
<td>84.00</td>
<td>56.00</td>
</tr>
</tbody>
</table>
The accuracy of SMURF with feedback

Figure: Trends in the increase of precision and recall when integrating incremental feedback
Results

Incremental detection approach based on SVM and using users’ feedback

- We test SMURF on 3 systems and 4 DS
- The accuracy of SMURF is greater than that of DETEX
- SMURF is more stable than the probabilistic approach BDTEX
- SMURF is an incremental detection approach: could be applied in continuous integration context

⇒ Make developers proactive and help to reduce maintenance costs
Outline

1. Introduction
2. Related Works and Contributions
3. Impact of Design Smells on Comprehension
4. Impact of Design Smells on Fault-fixing
5. SMURF
6. Conclusion and Perspectives
Conclusion 1/3

Quantitative evidences of impact of DS on source code comprehension of systems

⇒ More time to understand system with co-occurrences of Blob or Spaguetti Code with large effect size
⇒ More effort to understand system with co-occurrences of Blob or Spaguetti Code with large effect size
⇒ Lower percentage of good answers when performing task on system with co-occurrences of Blob or Spaguetti Code with large effect size

Developers should take rational decisions about their design quality to avoid the negative impact of DS on code source comprehension
Conclusion 2/3

Quantitative evidences of impact of DS on fault-fixing activities

⇒ The duration of the fixing period is longer for faults involving classes with DS
⇒ Fixing faults in classes with DS impacts more files, more fields/methods with higher entropy
⇒ After a fault is fixed, the number of occurrences of design smells in the classes involved in the fault decreases

Help developers to deal with DS when fixing a fault
Conclusion 3/3

1. **Incremental** detection approach based on SVM and using users’ feedback
   - The accuracy of SMURF is greater than that of DETEX and BDTEX
   - SMURF can be applied in both intra-system and inter-system configurations
   - SMURF accuracy improves when using practitioners’ feedback

Make developers proactive and help to reduce maintenance costs
Short Term Perspectives

- Replicate the studies in other contexts, with other subjects, questions, kind of design smells, and other systems
- Investigate the relation between the number of occurrences of design smells and the level of comprehension
- Study the impact of design pattern on source code comprehension and fault-fixing activities
- Use our incremental detection tool, SMURF, in real-world environments
Long Term Perspectives

- How long does a design smell Survive?
- What are the factors of introduction and propagation of design smells in software system?
- What are the factors of extinction of design smells?
- How to automatically remove and propose refactoring for design smells?
**Thesis**

There is an impact of design smells on source code comprehension and fault-fixing activities and we can provide a tool for accurate incremental design smells detection in ASD context.

**Contributions**

1. **Quantitative** evidences of impact of DS on source code comprehension of systems
   - Help developers to take rational decisions about their design quality

2. **Quantitative** evidences of impact of DS on fault-fixing activities
   - Help developers to deal with DS

3. **Incremental** detection approach based on SVM and using users’ feedback
   - Make developers proactive and help to reduce maintenance costs