Analyzing the Effects of Refactorings on Bad Smells

Cleiton Silva Tavares
Advisor: Mariza Bigonha
Co-Advisor: Eduardo Figueiredo

March 31th, 2021

Programming Language Laboratory (LLP)
http://www.llp.dcc.ufmg.br/

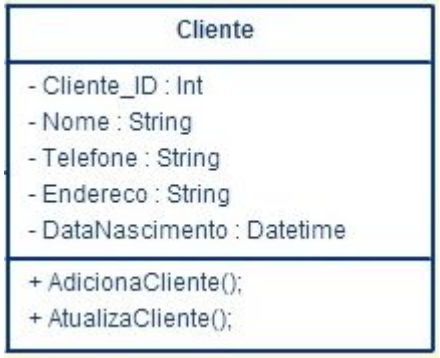
Software Engineering Lab (LabSoft)
http://labsoft.dcc.ufmg.br/

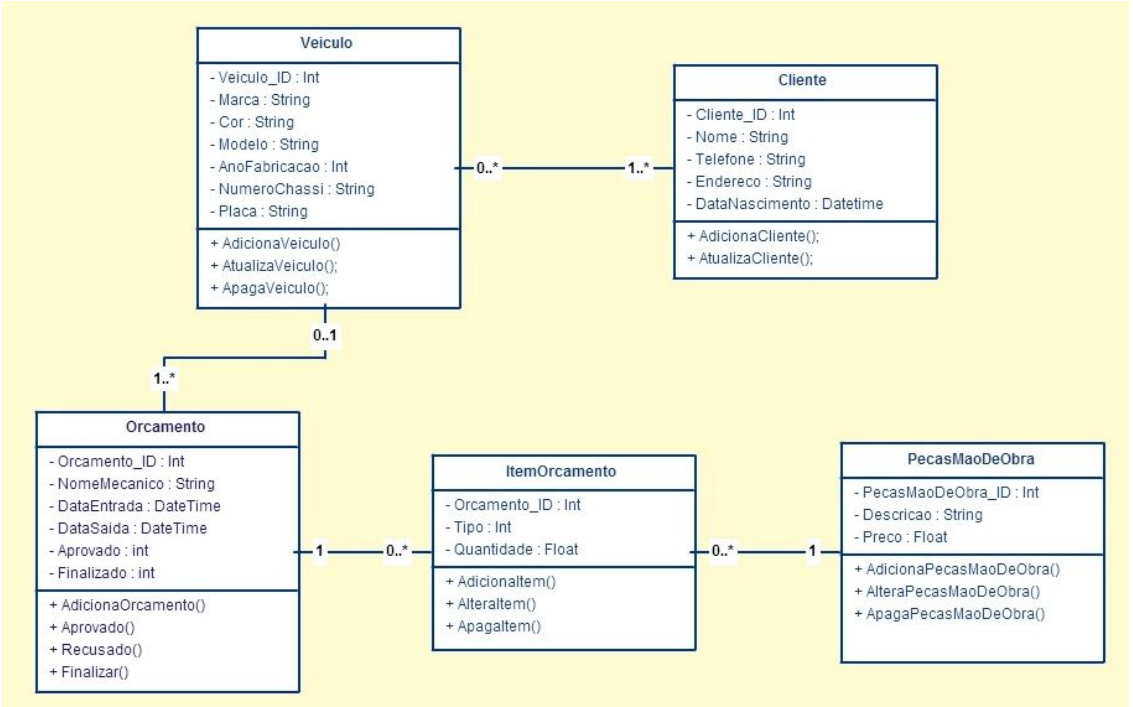
Summary

- Introduction
- Systematic Literature Review
- Empirical Study
- Comparative Analysis
- Conclusion

Motivation

- A software system is in the process of constant change and evolution
  - Complex over time



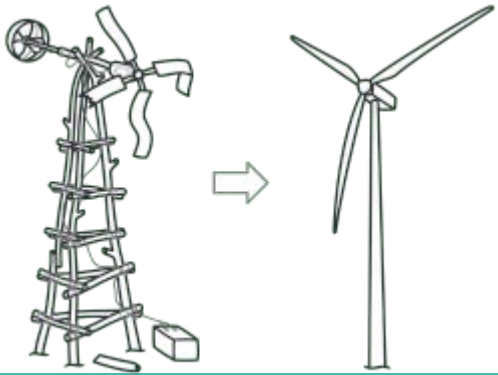


Motivation

- A software system is in the process of constant change and evolution
  - Great effort to understand and make modifications in the source code
- Bad Smell
  - Problems in the code structure

Motivation

- A software system is in the process of constant change and evolution
- ~~Bad Smell~~
  - ~~Problems in the code structure~~
- Refactoring
  - Increase the maintainability of the code by changing its internal structure without changing its behavior



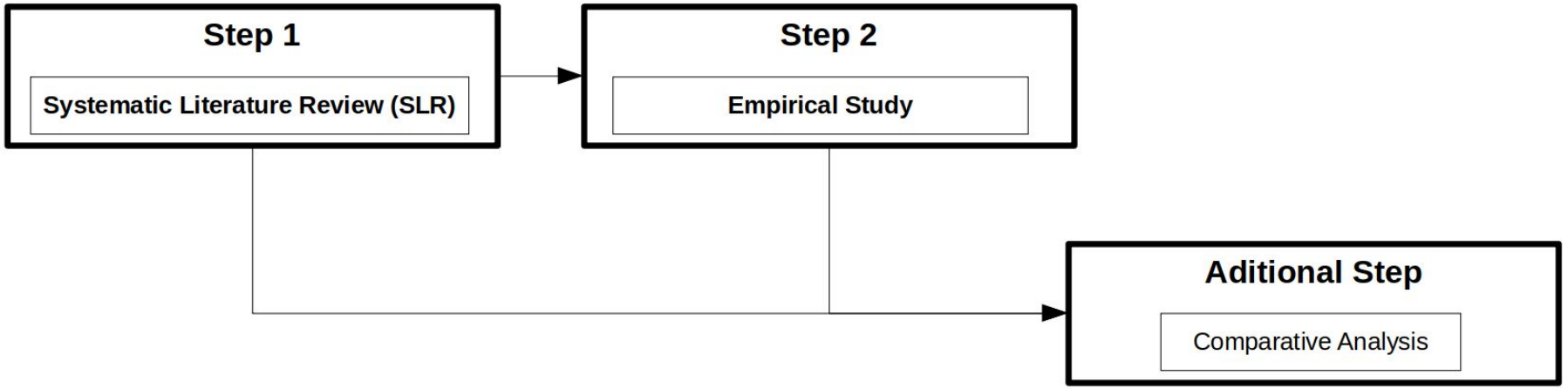
Problem Statement

- Refactoring is a tricky activity
  - Identify the code fragment to refactor
  - Operation that will solve it
  - Where allocate the refactored code
- Some studies show that refactoring may introduce new bad smells into the source code

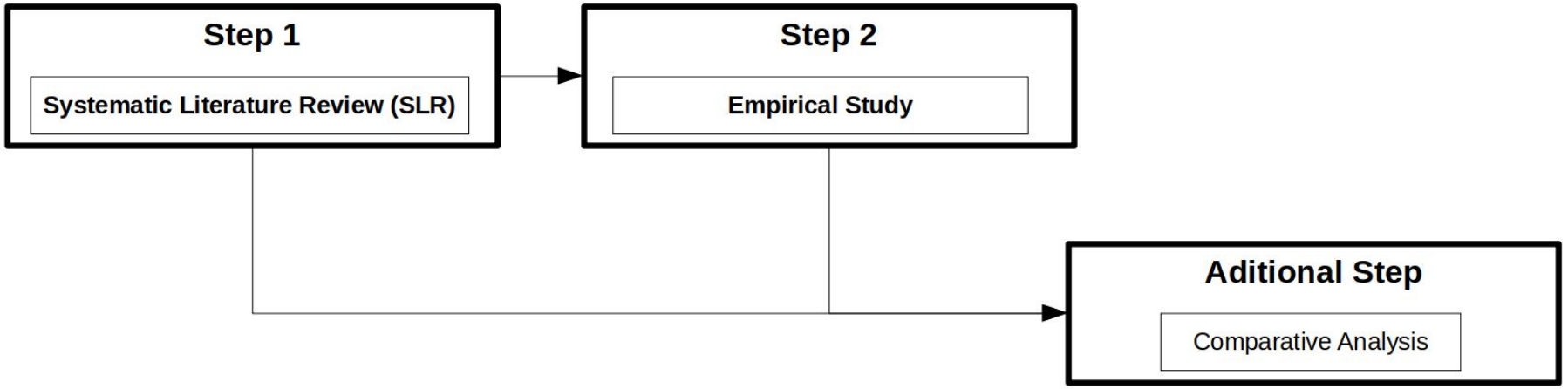
Proposed Work

- Research study to evaluate refactoring operations' impact on bad smells
  - Systematic Literature Review (SLR)
  - Empirical Study

Study Steps


Step 1
Systematic Literature Review (SLR)
Step 2
Empirical Study
Aditional Step
Comparative Analysis

Study Steps


Step 1
Systematic Literature Review (SLR)
Step 2
Empirical Study
Aditional Step
Comparative Analysis

SLR Goal

- Find a direct relationship between the bad smell and the refactoring proposed in the Fowler's catalog

SLR Research Questions

- RQ1 - Which relationships between refactoring and bad smells are the literature explicitly discussing?

- RQ2 - Which tools found the papers perform refactoring from bad smell detection?

SLR Research Questions

- RQ1 - Which relationships between refactoring and bad smells are the literature explicitly discussing?
  - RQ1.1 - Which are the most mentioned relationships between bad smells and refactoring found in the literature?
  - RQ1.2 - Do relationships we found are different from those that Fowler presents?

Answering RQ1

- RQ1 - Which relationships between refactoring and bad smells are the literature explicitly discussing?

  - 20 Papers found
  - 22 Fowler's Bad Smells
    - 16 different bad smells - 73%
  - 72 Fowler's Refactorings
    - 31 different refactorings - 43%

Answering RQ1.1

- RQ1.1 - Which are the most mentioned relationships between bad smells and refactoring found in the literature?
  - Relationship being discussed by the largest number of different studies
    - **Move Method and Feature Envy** presented by 14 different papers
  - Refactoring operation that relates the largest amount of bad smells
    - **Move Method** being relate to 11 types of smells
  - Bad smell related with the highest number of refactorings
    - **Long Method** related to 13 refactorings

Answering RQ1.2

- RQ1.2 - Do relationships we found are different from those that Fowler presents?

Bad Smell	Refactoring	Fowler	Literature
Data Class	Encapsulate Collection	•	•
	Encapsulate Field	•	•
	Extract Method	•	•
	Hide Method	•	•
	Move Method	•	•
	Remove Setting Method	•	•
Data Clumps	Extract Class	•	
	Introduce Parameter Object	•	
	Preserve Whole Object	•	
Divergent Change	Extract Class	•	•
	Extract Method		•
	Extract Superclass		•
	Move Field		•
	Move Method		•
	Pull Up Method		•

Answering RQ1.2

- RQ1.2 - Do relationships we found are different from those that Fowler presents?

Bad Smell	Refactoring	Fowler	Literature
Data Class	Encapsulate Collection	•	•
	Encapsulate Field	•	•
	Extract Method	•	•
	Hide Method	•	•
	Move Method	•	•
	Remove Setting Method	•	•
Data Clumps	Extract Class	•	
	Introduce Parameter Object	•	
	Preserve Whole Object	•	
Divergent Change	Extract Class	•	•
	Extract Method		•
	Extract Superclass		•
	Move Field		•
	Move Method		•
	Pull Up Method		•

Answering RQ1.2

- RQ1.2 - Do relationships we found are different from those that Fowler presents?

Bad Smell	Refactoring	Fowler	Literature
Data Class	Encapsulate Collection	•	•
	Encapsulate Field	•	•
	Extract Method	•	•
	Hide Method	•	•
	Move Method	•	•
	Remove Setting Method	•	•
Data Clumps	Extract Class	•	
	Introduce Parameter Object	•	
	Preserve Whole Object	•	
Divergent Change	Extract Class	•	•
	Extract Method		•
	Extract Superclass		•
	Move Field		•
	Move Method		•
	Pull Up Method		•

Answering RQ1.2

- RQ1.2 - Do relationships we found are different from those that Fowler presents?

Bad Smell	Refactoring	Fowler	Literature
Data Class	Encapsulate Collection	•	•
	Encapsulate Field	•	•
	Extract Method	•	•
	Hide Method	•	•
	Move Method	•	•
	Remove Setting Method	•	•
Data Clumps	Extract Class	•	
	Introduce Parameter Object	•	
	Preserve Whole Object	•	
Divergent Change	Extract Class	•	•
	Extract Method		•
	Extract Superclass		•
	Move Field		•
	Move Method		•
	Pull Up Method		•

Answering RQ1.2

- RQ1.2 - Do relationships we found are different from those that Fowler presents?

Bad Smell	Refactoring	Fowler	Literature
Data Class	Encapsulate Collection	•	•
	Encapsulate Field	•	•
	Extract Method	•	•
	Hide Method	•	•
	Move Method	•	•
	Remove Setting Method	•	•
Data Clumps	Extract Class	•	
	Introduce Parameter Object	•	
	Preserve Whole Object	•	
Divergent Change	Extract Class	•	•
	Extract Method		•
	Extract Superclass		•
	Move Field		•
	Move Method		•
	Pull Up Method		•

24 relationships not addressed in Fowler's catalog

Answering RQ2

- RQ2 - Which tools found the papers perform refactoring from bad smell detection?

Tool [Ref.]	GUI	FRA	ONL	PLG	FRE	OPS	USG	SL
Extract Method Detector [46]	Yes	-	No	Yes	Yes	Yes	No	Java
JDeodorant [8; 17; 65]	Yes	-	No	Yes	Yes	Yes	Yes	Java
JMove [64]	Yes	-	-	Yes	Yes	Yes	Yes	-
Liu's Approach [44]	-	-	-	-	-	-	-	-
Methodbook [9]	-	-	-	-	-	-	-	Java
MMRUC3 [63]	-	Yes	-	-	-	-	-	Java
Tsantalis's Methodology [79]	-	-	-	-	-	-	-	Java

GUI: graphical user interface; **FRA**: framework; **ONL**: online; **PLG**: plugin; **FRE**: free for use; **OPS**: open-source; **USG**: user guide available; **SL**: supported language; "**-**": information not available.

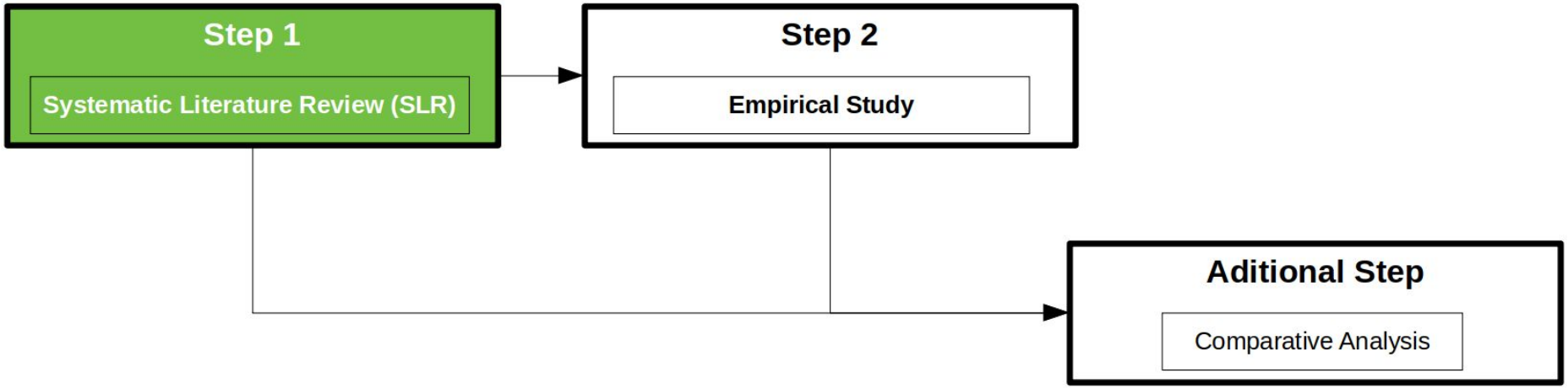
SLR Final Remarks

- 20 different papers that show the direct relationship
  - 31 refactoring types and 16 bad smells proposed by Fowler
- The most discussed relationship in the literature
  - Move Method and Feature Envy
- 16 papers mentioned the tools used
  - 07 perform refactoring from bad smell detection

SLR Final Remarks / Implication

- Most strategies defined in the Fowler's book were addressed in the literature
- There are different refactoring strategies than those discussed by Fowler to address bad smells
    - Empirical studies to validate the new relationships found

Study Steps


Step 1
Systematic Literature Review (SLR)
Step 2
Empirical Study
Aditional Step
Comparative Analysis

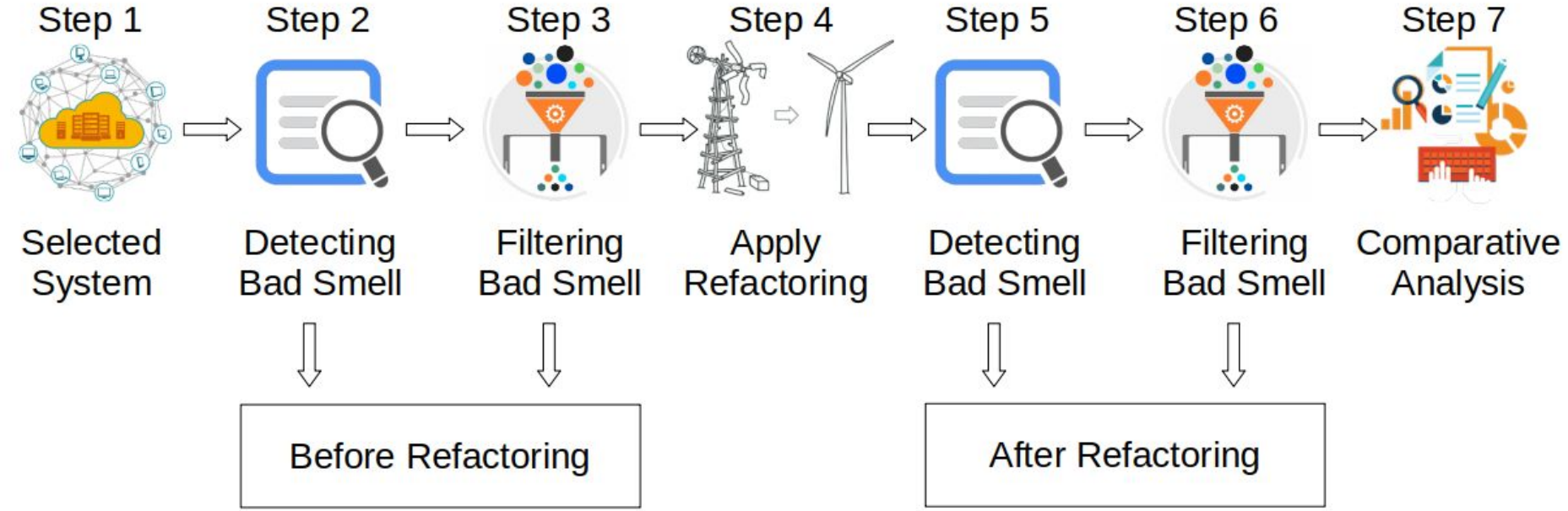
Empirical Study Goal

- Conduct empirical research to assess the impacts of automated refactoring on detecting bad smells

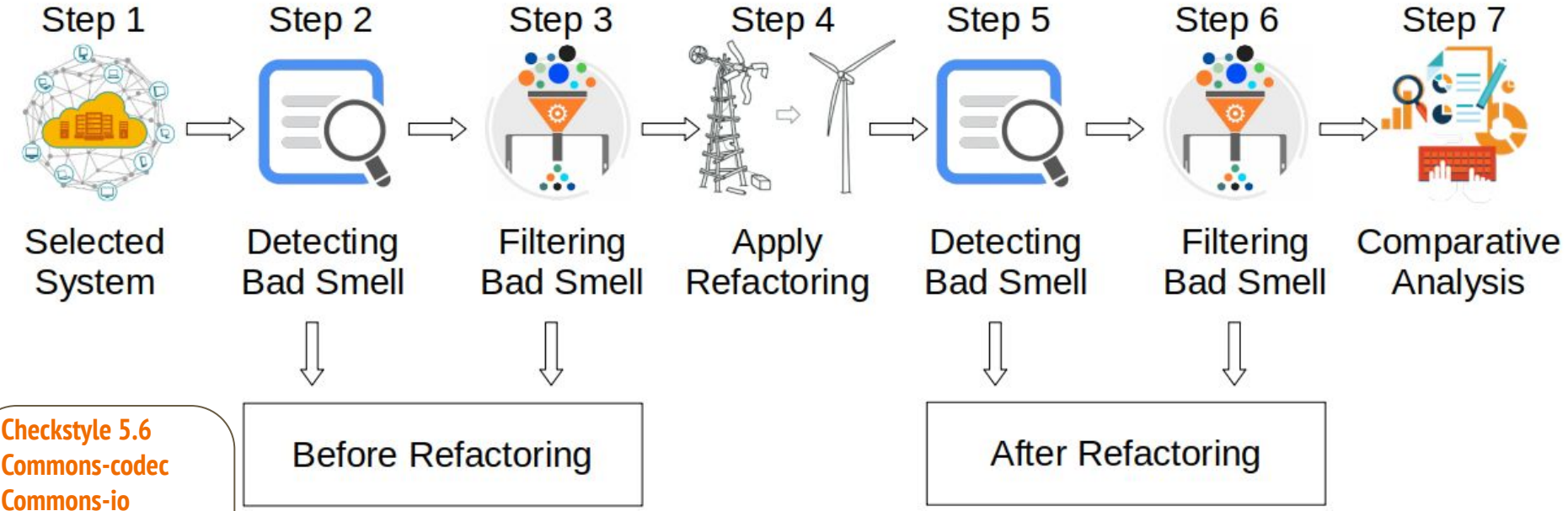
Empirical Study Research Question

- RQ - What are the impacts of automated refactoring on the detection of bad smells?
  - RQ1 - Does the automated refactoring process remove bad smells?
  - RQ2 - Does the automated refactoring process introduce bad smells?

Empirical Study Design

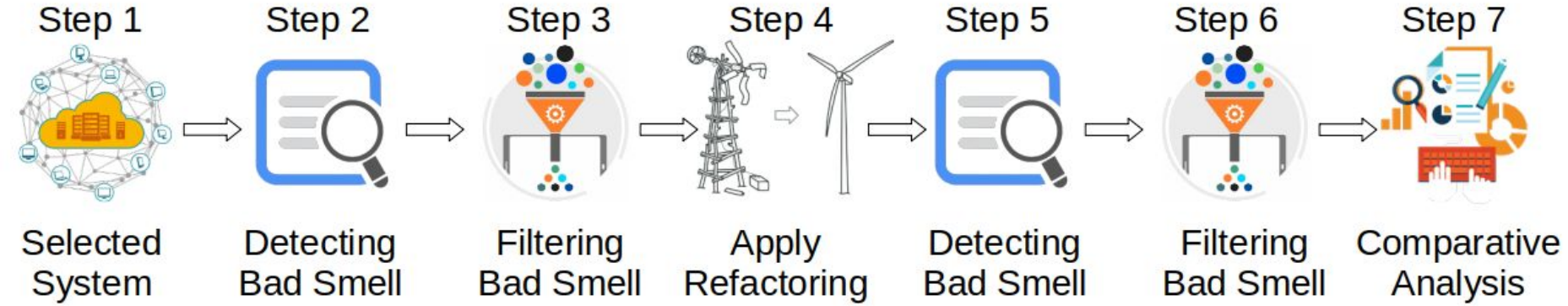


Empirical Study Design



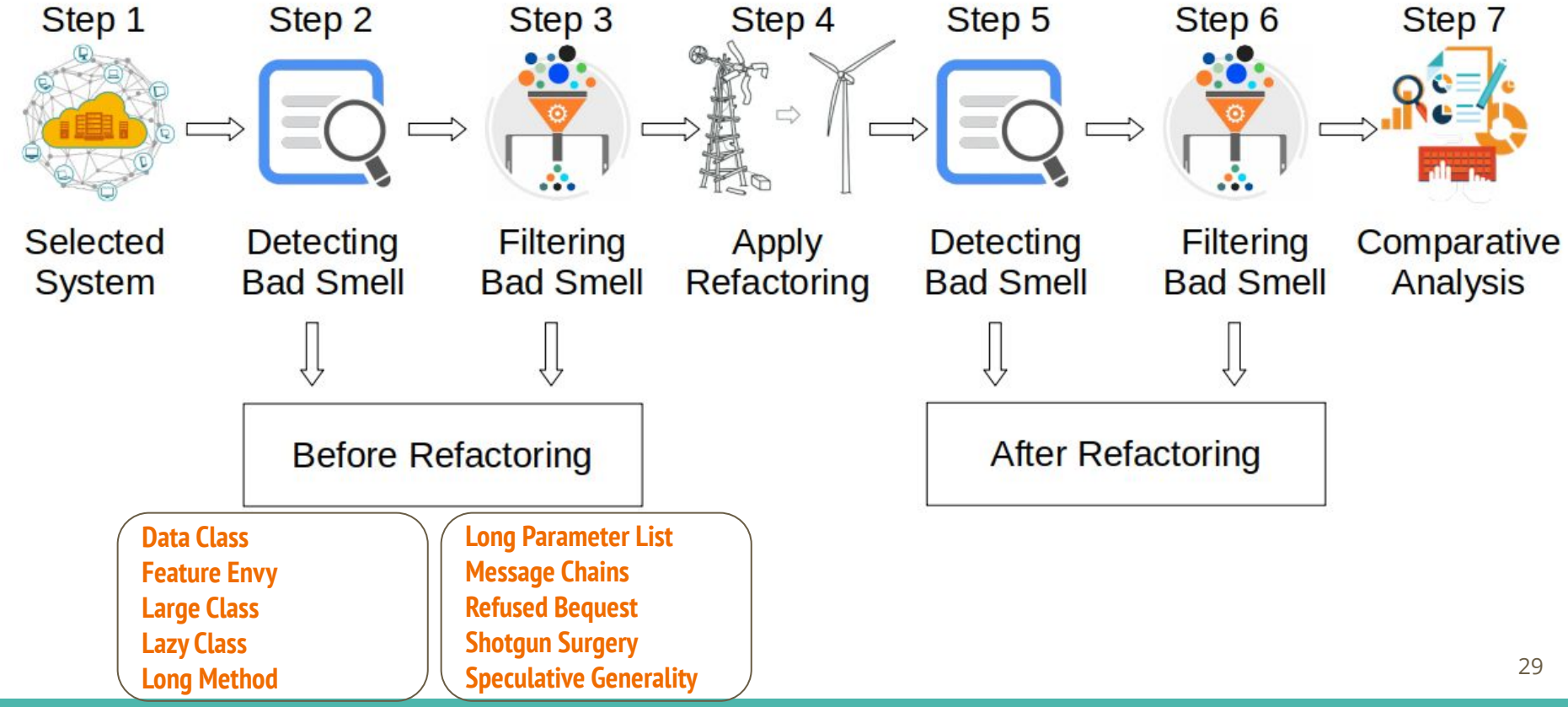
- Checkstyle 5.6
- Commons-codec
- Commons-io
- Commons-lang
- Commons-logging
- JHotDraw 7.5.1
- Quartz 1.8.3
- Squirrel_sql 3.1.2

Empirical Study Design

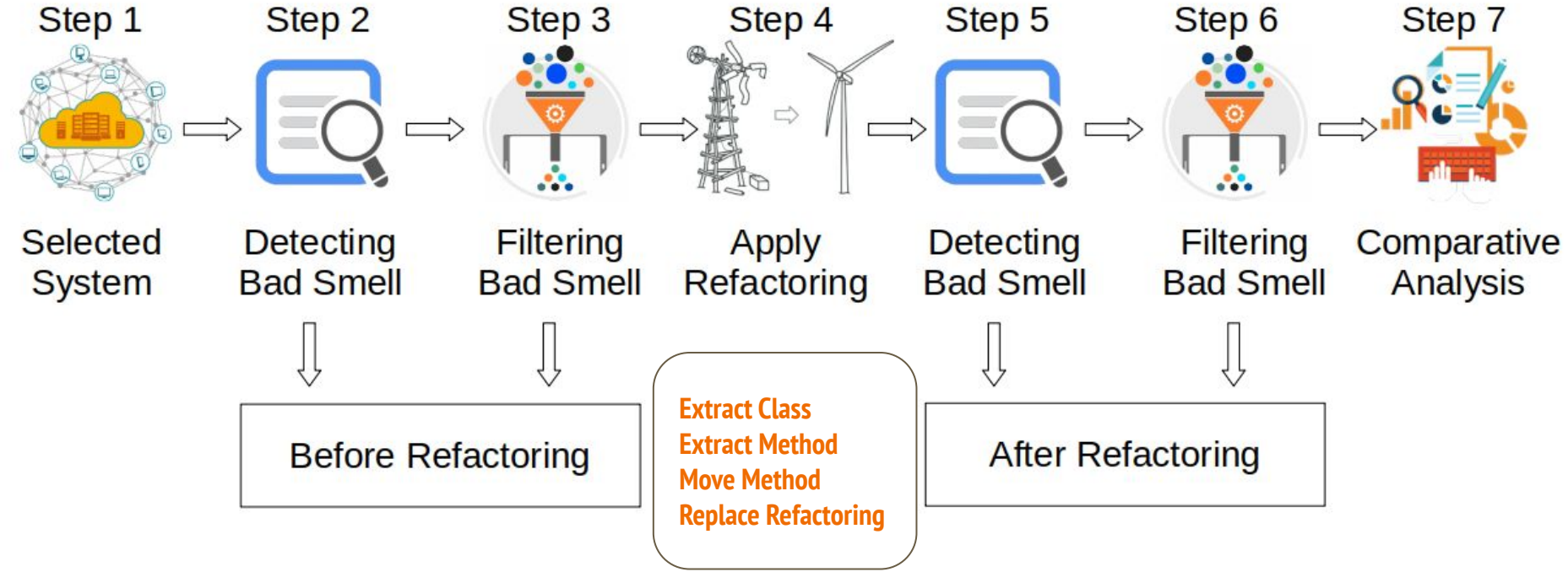


- Decor
- Designite
- JDeodorant
- JSpIRIT
- Organic

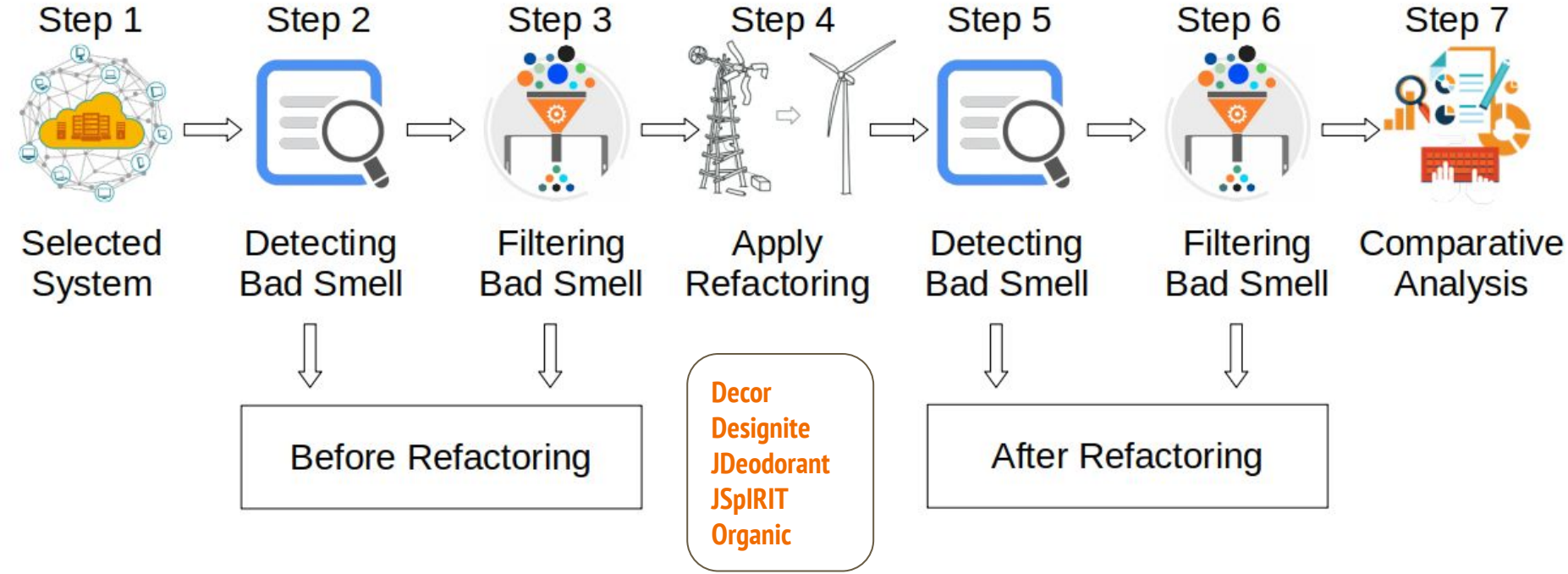
Empirical Study Design



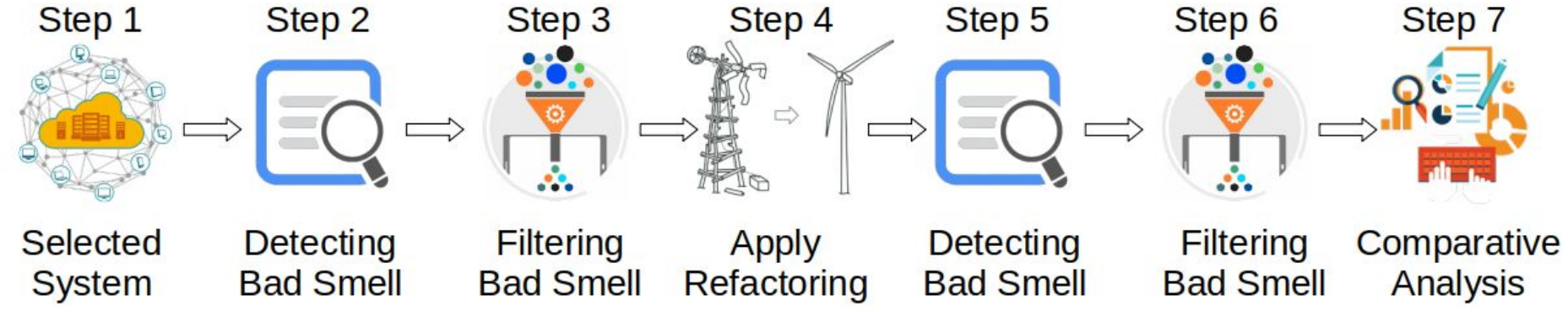
Empirical Study Design



Empirical Study Design



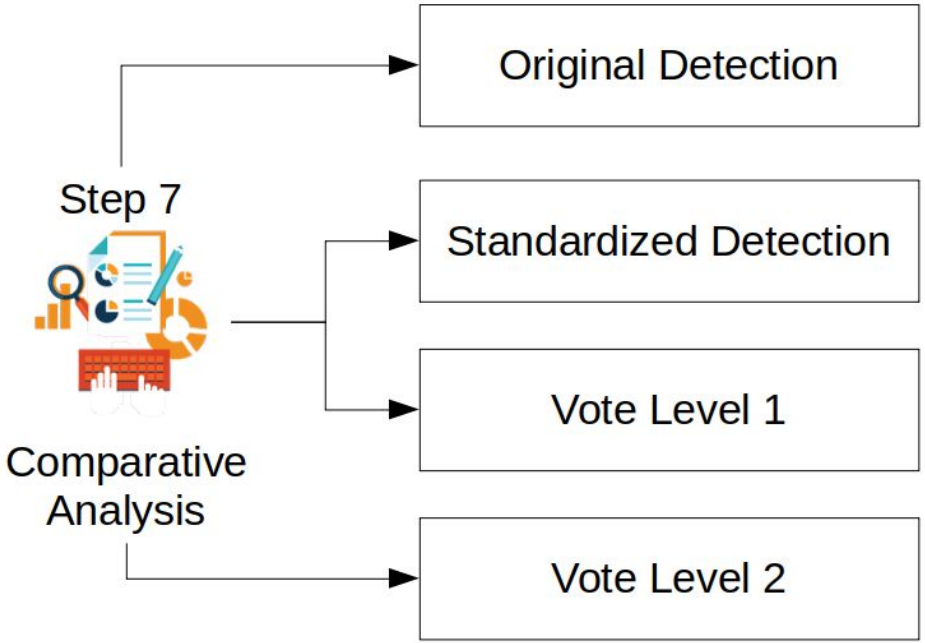
Empirical Study Design



- Data Class
- Feature Envy
- Large Class
- Lazy Class
- Long Method

- Long Parameter List
- Message Chains
- Refused Bequest
- Shotgun Surgery
- Speculative Generality

Comparative Analysis Perspectives



Answering RQ1.1

- RQ1.1 - Does the automated refactoring process remove bad smells?

Refactoring	Bad Smell	% of system
Extract Class	Feature Envy	**12.5 %**
	Lazy Class	**12.5 %**
	Long Parameter List	**12.5 %**
Extract Method	Long Method	**50.0 %**
	Refused Bequest	12.5 %
Move Method	Feature Envy	**62.5 %**
	Long Method	12.5 %
Replace Refactoring	-	-

Answering RQ1.2

- RQ1.2 - Does the automated refactoring process introduce bad smells?

Refactoring	Bad Smell	% of system
Extract Class	Feature Envy	**37.5 %**
	Lazy Class	12.5 %
	Long Method	25.0 %
	Long Parameter List	**37.5 %**
Extract Method	Feature Envy	**25.0 %**
	Lazy Class	12.5 %
	Long Parameter List	**25.0 %**
Move Method	-	-
Replace Refactoring	Refused Bequest	**25.0 %**

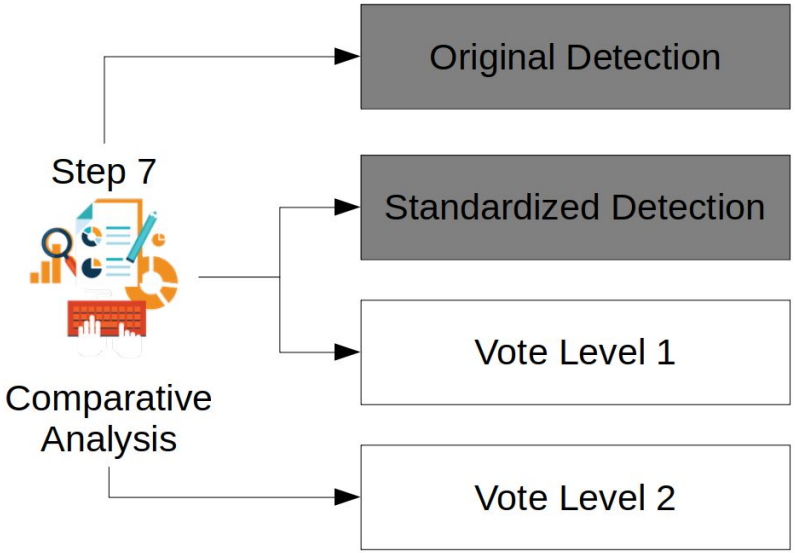
Answering RQ

- RQ - What are the impacts of automated refactoring on the detection of bad smells?

Refactoring	Decrease	Increase	Neutral
Extract Class	3.75 %	**11.25 %**	23.75 %
Extract Method	6.25 %	6.25 %	8.75 %
Move Method	**7.50 %**	0.00 %	**25.00 %**
Replace Refactoring	0.00 %	2.50 %	22.50 %

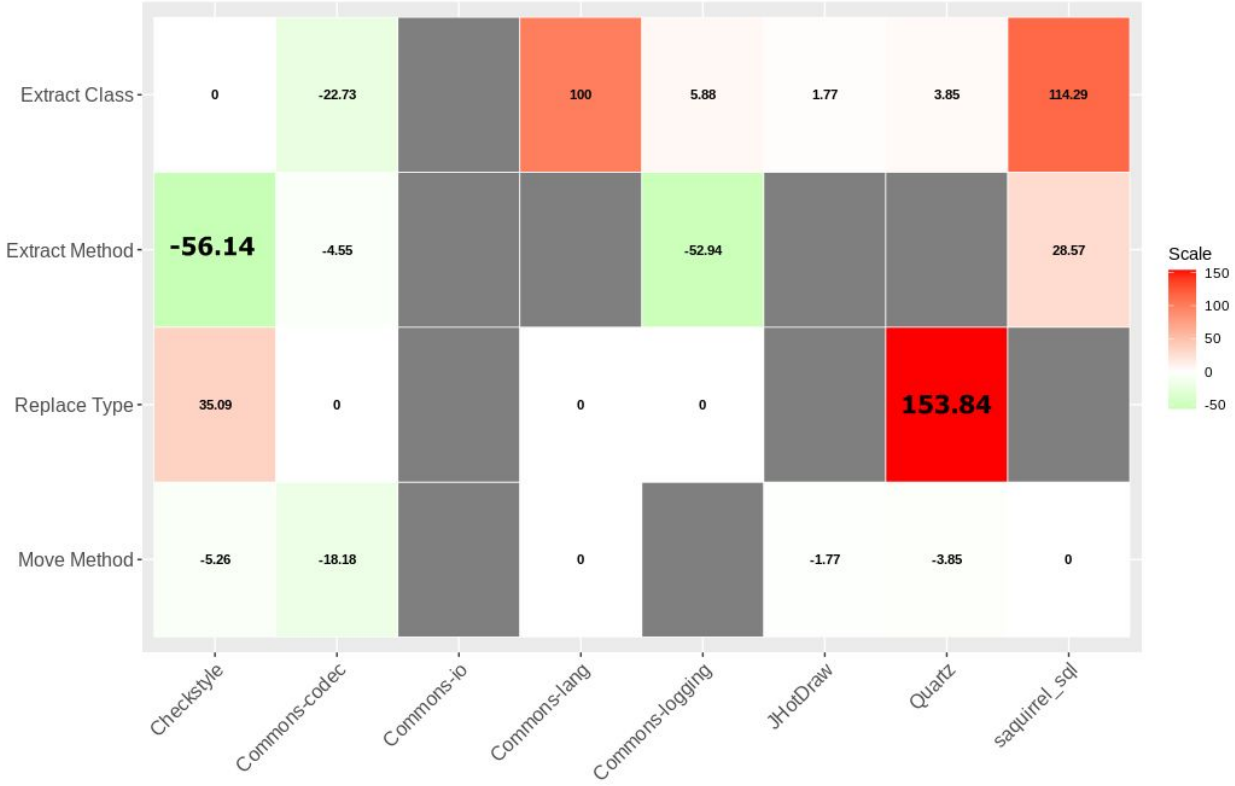
* 80 situations

Aggregated Analysis



System	Original Version	Extract Class	Extract Method	Move Method	Replace Refactoring
Checkstyle-5.6	57	57	25	54	77
Quartz-1.8.3	26	27	-	25	66

Aggregated Analysis



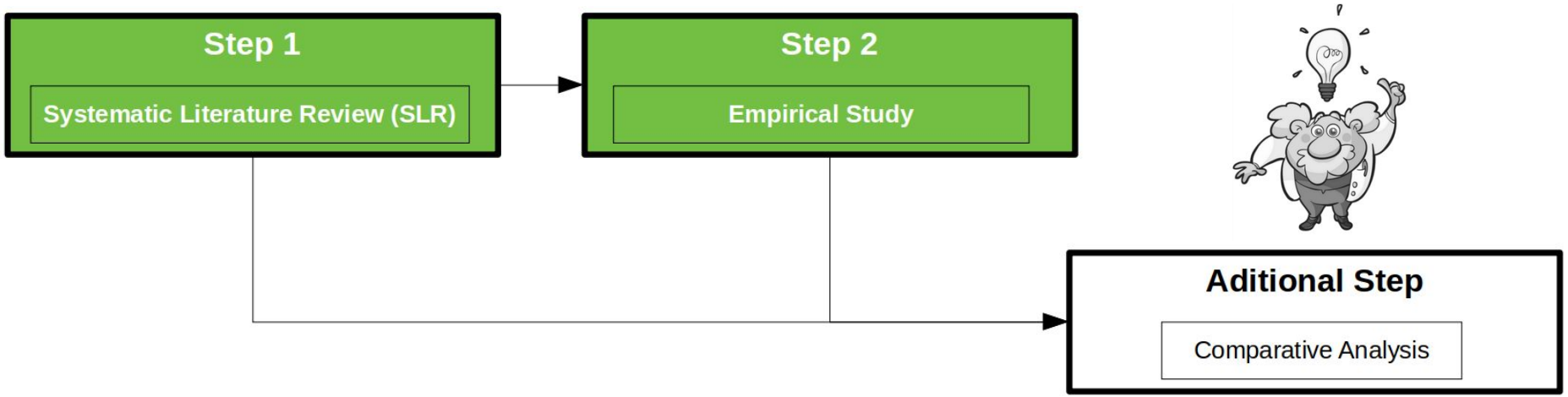
Empirical Study Final Remarks

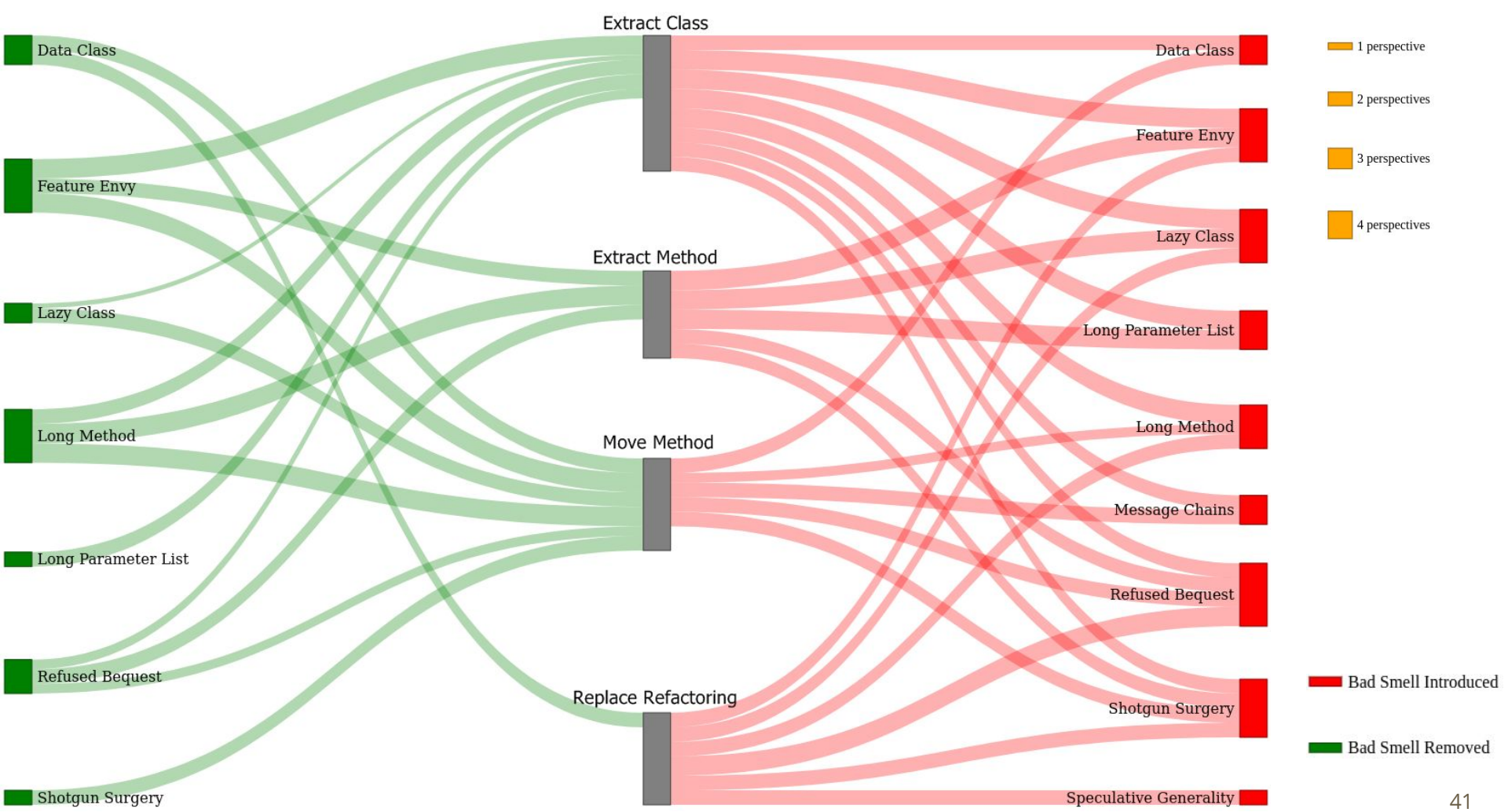
- Empirical research analyzing the impact of four refactorings on ten bad smells

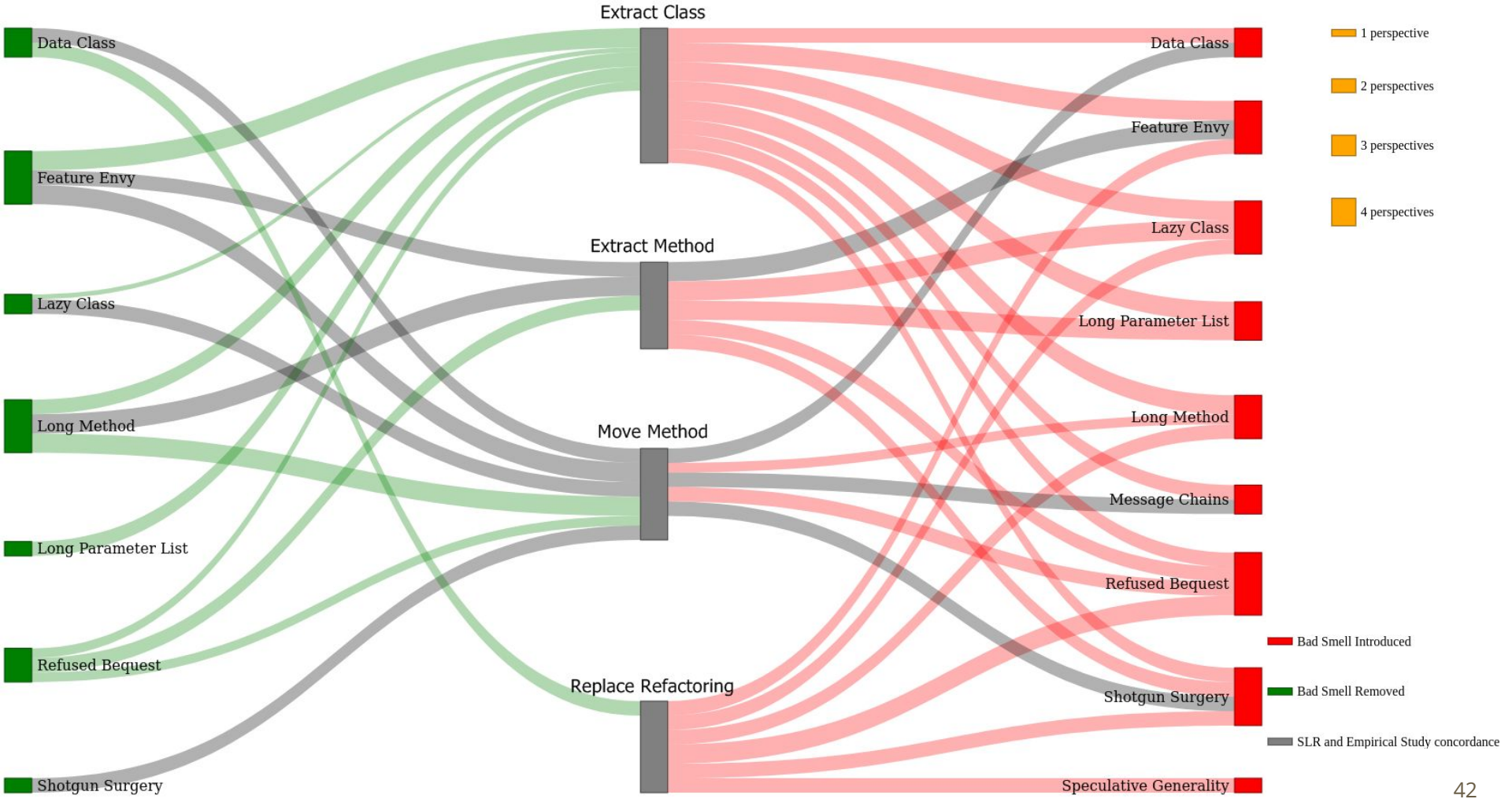
- We present our results in four different perspectives

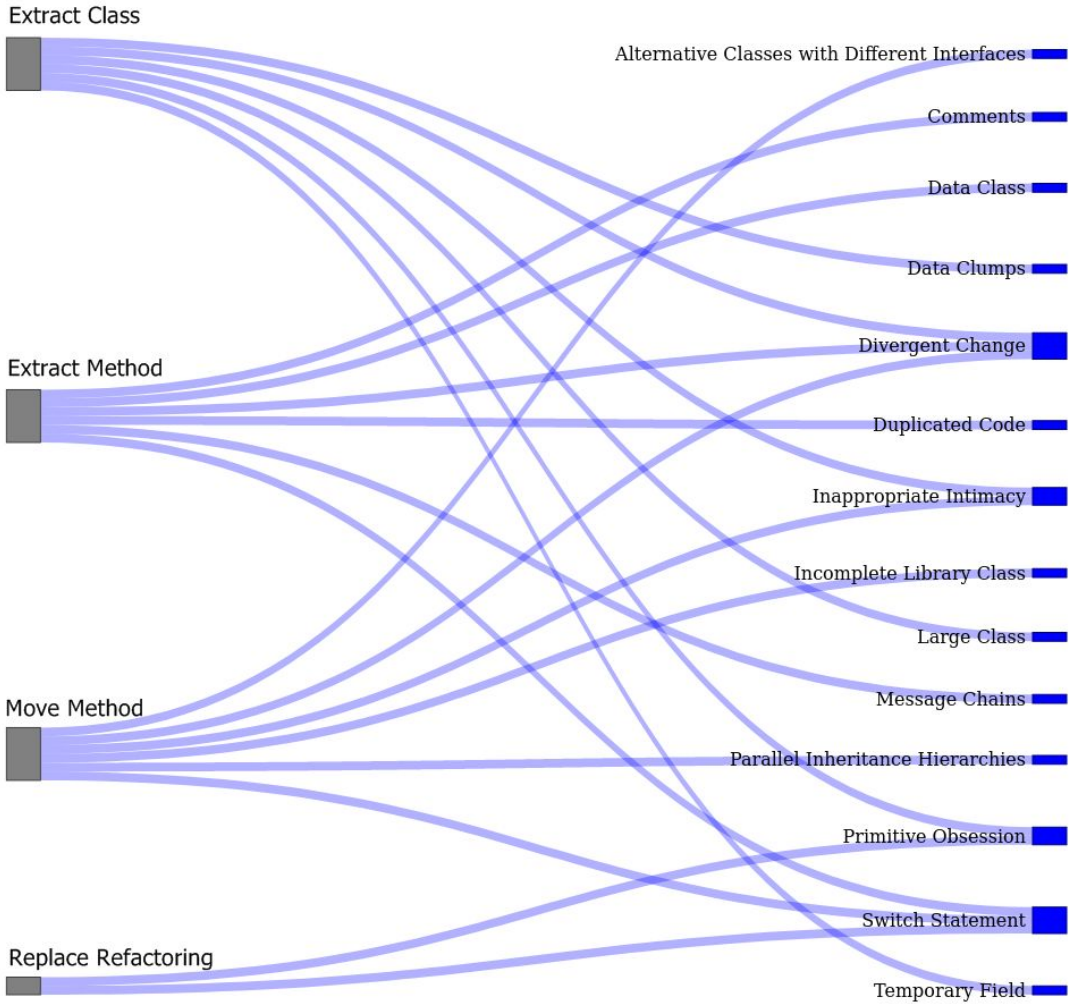
- Surprisingly, the number of decrease cases was the lowest compared to the others
  - Except in Move Method refactoring applied

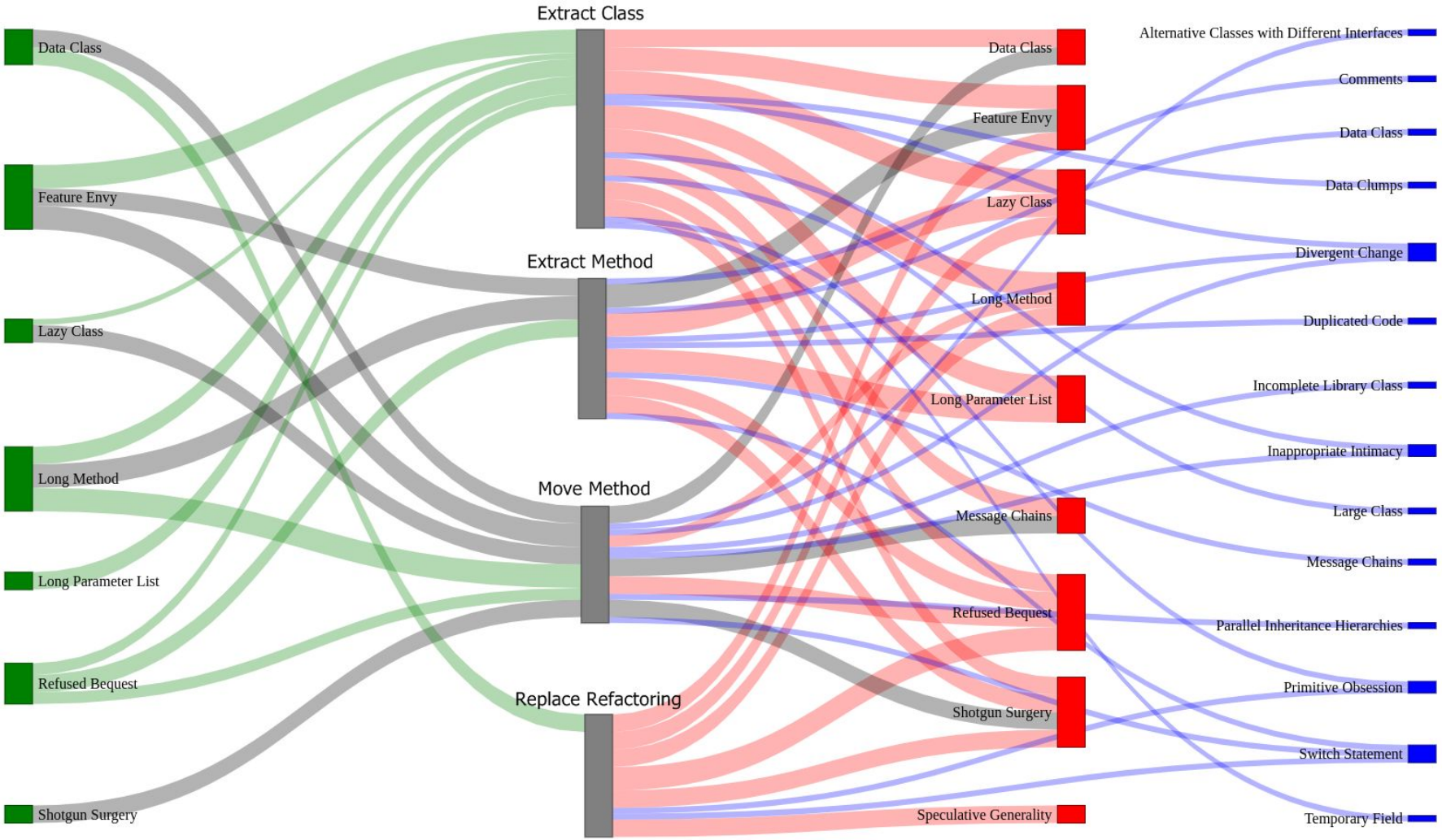
Study Steps


Step 1
Systematic Literature Review (SLR)
Step 2
Empirical Study
Aditional Step
Comparative Analysis


Data Class
Feature Envy
Lazy Class
Long Method
Long Parameter List
Refused Bequest
Shotgun Surgery
Extract Class
Extract Method
Move Method
Replace Refactoring
Data Class
Feature Envy
Lazy Class
Long Parameter List
Long Method
Message Chains
Refused Bequest
Shotgun Surgery
Speculative Generality
1 perspective
2 perspectives
3 perspectives
4 perspectives
Bad Smell Introduced
Bad Smell Removed


Data Class
Feature Envy
Lazy Class
Long Method
Long Parameter List
Refused Bequest
Shotgun Surgery
Extract Class
Extract Method
Move Method
Replace Refactoring
Data Class
Feature Envy
Lazy Class
Long Parameter List
Long Method
Message Chains
Refused Bequest
Shotgun Surgery
Speculative Generality
1 perspective
2 perspectives
3 perspectives
4 perspectives
Bad Smell Introduced
Bad Smell Removed
SLR and Empirical Study concordance


Extract Class
Extract Method
Move Method
Replace Refactoring
Alternative Classes with Different Interfaces
Comments
Data Class
Data Clumps
Divergent Change
Duplicated Code
Inappropriate Intimacy
Incomplete Library Class
Large Class
Message Chains
Parallel Inheritance Hierarchies
Primitive Obsession
Switch Statement
Temporary Field
Bad Smell Introduced
Bad Smell Removed
SLR and Empirical Study concordance
SLR Relationship (Smell, Refactoring)


Extract Class
Extract Method
Move Method
Replace Refactoring
Data Class
Feature Envy
Lazy Class
Long Method
Long Parameter List
Refused Bequest
Shotgun Surgery
Data Class
Feature Envy
Lazy Class
Long Method
Long Parameter List
Message Chains
Refused Bequest
Shotgun Surgery
Speculative Generality
Alternative Classes with Different Interfaces
Comments
Data Class
Data Clumps
Divergent Change
Duplicated Code
Incomplete Library Class
Inappropriate Intimacy
Large Class
Message Chains
Parallel Inheritance Hierarchies
Primitive Obsession
Switch Statement
Temporary Field

Conclusion

Conclusion

- Systematic Literature Review (SLR)
  - 20 papers found
  - Relationships between 31 refactorings and 16 bad smells
- Empirical Study
  - Analyzed the effects of four refactorings on ten bad smells
- Comparative Analysis with the SLR and the Empirical Study

Contributions

- Catalog presenting the relationship between bad smells and refactoring discussed in the literature and a contrast with Fowler’s catalog

- Catalog showing which bad smells tend to be introduced and removed by the automatic refactoring strategy

Dissertation Research

Tavares et al. **Quantifying the Effects of Refactorings on Bad Smells**. WTDSOFT 2020.

Systematic Literature Review (SLR)

Silva et al. **Revisiting the bad smell and refactoring relationship: A systematic literature review**. ESELAW 2020.

Empirical Study

Tavares et al. **Analyzing the impact of refactoring on bad smells**. SBES 2020.

Implications

- Developers are better informed of which bad smells may be introduced or removed by the refactoring operation

- Perform the most efficient and robust refactoring tools so as not to introduce new bad smells in the source code

Future Work

- Investigate the new Fowler's catalog
  - Others refactorings and smells

- Conduct the Empirical Study with manually refactoring
  - Contrast with our automated refactoring

Thank You!

Programming Language Laboratory (LLP)
http://www.llp.dcc.ufmg.br/

CAPES

Software Engineering Lab (LabSoft)
http://labsoft.dcc.ufmg.br/

Analyzing the Effects of Refactorings on Bad Smells

Cleiton Silva Tavares
Advisor: Mariza Bigonha
Co-Advisor: Eduardo Figueiredo

March 31th, 2021

Programming Language Laboratory (LLP)
http://www.llp.dcc.ufmg.br/

Software Engineering Lab (LabSoft)
http://labsoft.dcc.ufmg.br/

Extra Slide

- Extract Class Introducing Long Parameter List

  Original Version

  JHotDraw

JSheetProductRefactored.java | JSheet.java

```
527
528         fireOptionSelected(pane, option, value, pane.getInputValue());
529     }
530
531     /**
532      * Notify all listeners that have registered interest for
533      *   notification on this event type.  The event instance
534      *   is lazily created using the parameters passed into
535      *   the fire method.
536      */
537     protected void fireOptionSelected(JOptionPane pane, int option, Object value, Object inputValue) {
538         SheetEvent sheetEvent = null;
539         // Guaranteed to return a non-null array
540         Object[] listeners = listenerList.getListenerList();
541         // Process the listeners last to first, notifying
542         // those that are interested in this event
543         for (int i = listeners.length - 2; i >= 0; i -= 2) {
544             if (listeners[i] == SheetListener.class) {
545                 // Lazily create the event:
546                 if (sheetEvent == null) {
547                     sheetEvent = new SheetEvent(this, pane, option, value, inputValue);
548                 }
549                 ((SheetListener) listeners[i + 1]).optionSelected(sheetEvent);
550             }
551         }
552     }
553
554     /**
555      * Notify all listeners that have registered interest for
556      *   notification on this event type.  The event instance
557      *   is lazily created using the parameters passed into
558      *   the fire method.
559      */
560     protected void fireOptionSelected(JFileChooser pane, int option) {
561         SheetEvent sheetEvent = null;
562         // Guaranteed to return a non-null array
563         Object[] listeners = listenerList.getListenerList();
564         // Process the listeners last to first, notifying
565         // those that are interested in this event
566         for (int i = listeners.length - 2; i >= 0; i -= 2) {
567             if (listeners[i] == SheetListener.class) {
568                 // Lazily create the event:
```

Extra Slide

- Extract Class Introducing Long Parameter List

  Refactored Version

  JHotDraw

JSheetProductRefactored.java | JSheet.java

```
18          listenerList.add(SheetListener.class, l);
19      }
20
21      /**
22       * Removes a sheet listener.
23       */
24      public void removeSheetListener(SheetListener l) {
25          listenerList.remove(SheetListener.class, l);
26      }
27
28      /**
29       * Notify all listeners that have registered interest for notification on this event type.
30       */
31      public void fireOptionSelected(JOptionPane pane, int option, Object value,
32              Object inputValue, JSheet jSheet) {
33          SheetEvent sheetEvent = null;
34          Object[] listeners = listenerList.getListenerList();
35          for (int i = listeners.length - 2; i >= 0; i -= 2) {
36              if (listeners[i] == SheetListener.class) {
37                  if (sheetEvent == null) {
38                      sheetEvent = new SheetEvent(jSheet, pane, option, value,
39                              inputValue);
40                  }
41                  ((SheetListener) listeners[i + 1]).optionSelected(sheetEvent);
42              }
43          }
44      }
45
46      /**
47       * Notify all listeners that have registered interest for notification on this event type.
48       */
49      public void fireOptionSelected(JFileChooser pane, int option, JSheet jSheet) {
50          SheetEvent sheetEvent = null;
51          Object[] listeners = listenerList.getListenerList();
52          for (int i = listeners.length - 2; i >= 0; i -= 2) {
53              if (listeners[i] == SheetListener.class) {
54                  if (sheetEvent == null) {
55                      sheetEvent = new SheetEvent(jSheet, pane, option, null);
56                  }
57                  ((SheetListener) listeners[i + 1]).optionSelected(sheetEvent);
58              }
59          }
```