User Interface Design Smell: Automatic Detection and Refactoring of Blob Listeners

Arnaud Blouin, Valéria Lelli, Benoit Baudry, Fabien Coulon

DiverSE research group
Inria / IRISA, Rennes, France

On (UI) code quality

Bugs

- Crashes
- Errors
- Incorrect behaviours

```
if (g != null)
  paintScrollBars(g,colors);
g.dispose();
```

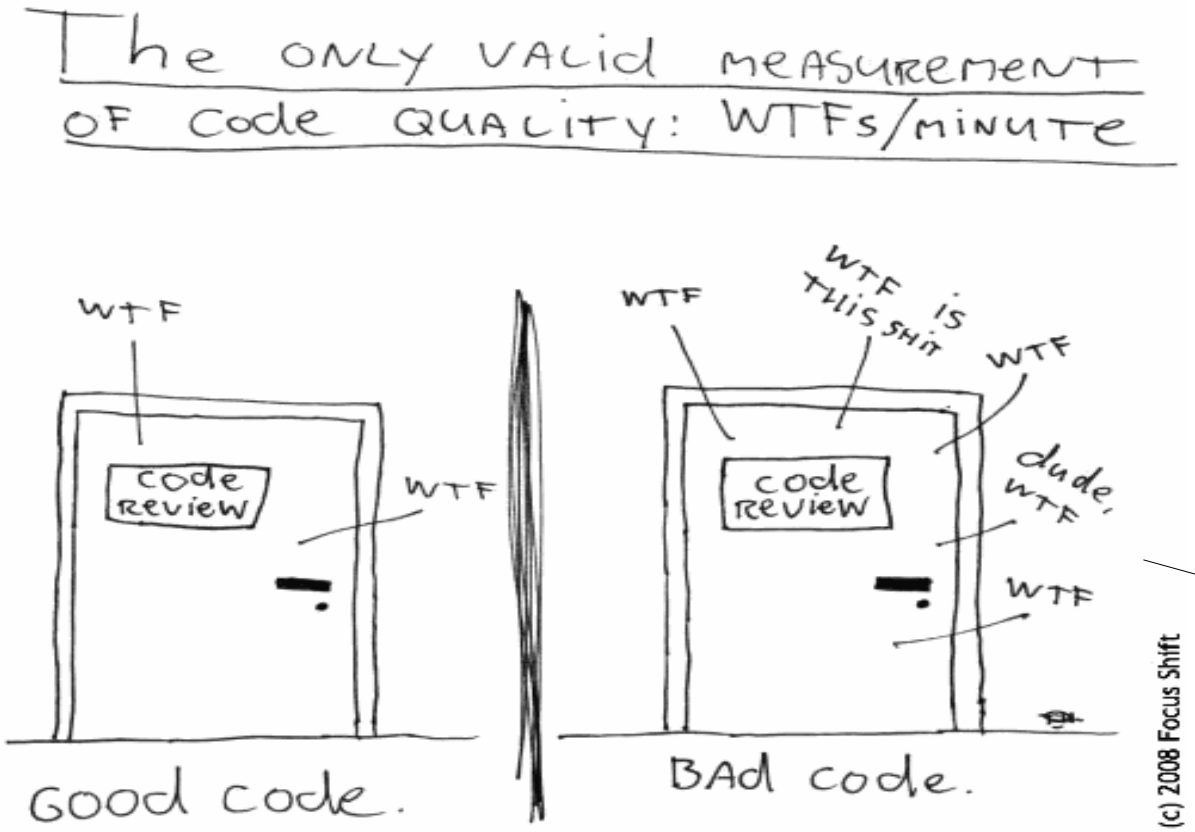
```
if (g != null) {
  paintScrollBars(g,colors);
  g.dispose();
}
```

Code/Design smells

Bad coding practices / design

- **May have negative impact**: readability, understandability, maintainability, etc.
- Examples:
  - long method (between 100 and 150 LoCs)
    Makes use of code metrics
  - Rules: “Code blocks without parentheses are forbidden”

Evaluating code quality



Code metrics

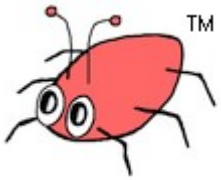
- # of methods / class
- # LoC / method
- etc

|WTF| > 2 => bad code

Automated code analysis tools

E.g. for object-oriented prog. (OOP):

Findbugs, PMD



Analysing user interface code

- Can apply Findbugs on UI code
- Can find OO design smells in UI code
- Many works on OOP code analysis, design smell, code metrics (Rasool2015)

=> **To find object-oriented issues only**

Analysing user interface code

What about specific UI elements that compose UI code?

- UI listeners, UI commands, data binding, MV* patterns, UIDL documents, widgets, etc.

Do these UI elements have specific issues, code smells?

- Is there specific errors, bad coding practices when coding undo/redo, data binding, **UI listeners, UI commands**?
- How can we detect them?
- Do they have a negative impact on the code quality?

Contributions of this work

- An **empirical study** that focuses on how developers code UI listeners
- The characterisation of **a UI design smell** that affects UI listeners: **Blob Listener**
- **A UI code analysis technique** to detect Blob Listeners deeply intertwined with the rest of the code
- **A behaviour-preserving code refactoring solution** to remove Blob listeners

What are UI listeners and commands?

```
class AController implements ActionListener {
  JButton b1;
  JButton b2;
  JMenuItem m3;

  @Override public void actionPerformed(ActionEvent e) {
     Object src = e.getSource();
     if(src==b1){
        // Command 1
     }else if(src==b2)
        // Command 2
     }else if(src instanceof AbstractButton &&
         ((AbstractButton)src).getActionCommand().equals(
         m3.getActionCommand()))
        // Command 3
     }
}}
```

UI listener method: method called on user actions

UI command: code executed in reaction of a user action on a widget

UI listeners that use “if” statements

```
public void actionPerformed(ActionEvent evt) {
   Object target = evt.getSource();
   if (target instanceof JButton) {
      //...
   } else if (target instanceof JTextField) {
      //...
   } else if (target instanceof JCheckBox) {
      //...
   } else if (target instanceof JComboBox) {
      //...
}}
```

Identification of the widget that produced the event

```
public void actionPerformed(ActionEvent event) {
   if(event.getSource() == view.moveDown) {
      //...
   } else if(event.getSource() == view.moveLeft) {
      //...
   } else if(event.getSource() == view.moveRight) {
      //...
   } else if(event.getSource() == view.moveUp) {
      //...
   } else if(event.getSource() == view.zoomIn) {
      //...
   } else if(event.getSource() == view.zoomOut) {
      //...
}}
```

These listeners manage several widgets

InspectorGuidget: a tool for detecting UI commands

- **Requires a specific code analysis technique**
  - Counting the # of “if” is not precise enough
  - Mandatory to refactor code
- **Java toolkits** supported (Swing, JavaFX, SWT)
- **Based on Spoon**, a Java code analysis framework
- **Open-source**: https://github.com/diverse-project/InspectorGuidget

Detecting UI commands

```
class AController implements ActionListener {
  JButton b1;
  JButton b2;
  JMenuItem m3;

  @Override public void actionPerformed(ActionEvent e) {
     Object src = e.getSource();
     if(src==b1){
        // Command 1
     }else if(src==b2)
        // Command 2
     }else if(src instanceof AbstractButton &&
         ((AbstractButton)src).getActionCommand().equals(
         m3.getActionCommand()))
        // Command 3
     }
}}
```

- **Detection of UI listener methods**
  - Have a UI event as a parameter
- **Command =** code block surrounded by a condition that uses (un-)directly a widget
- No condition = 1 command

Detecting UI commands: evaluation

Software system	Version	UI toolkit	kLoCs	# commits	# UI listeners
Eclipse (platform.ui.workbench)	4.7	SWT	143	10 049	259
JabRef	3.8.0	Swing	95	8567	486
ArgoUML	0.35.1	Swing	101	10 098	214
FreeCol	0.11.6	Swing	118	12 330	223

Detecting UI commands: evaluation

Software System	Successfully Detected Commands (#)	FN (#)	FP (#)	$Recall_{cmd}$ (%)	$Precision_{cmd}$ (%)
Eclipse	330	0	5	100	98.51
JabRef	510	5	7	99.03	98.65
ArgoUML	264	3	3	98.88	98.88
FreeCol	288	0	47	100	85.93
OVERALL	1392	8	62	99.43	95.73

- **Recall**: 99.43%
- **Precision**: 95.73%

Ground-truth = manual inspection of the UI listeners to identify the commands

Studying the coding practice “several commands per listener”

Does this coding practice has an negative impact on the code quality?

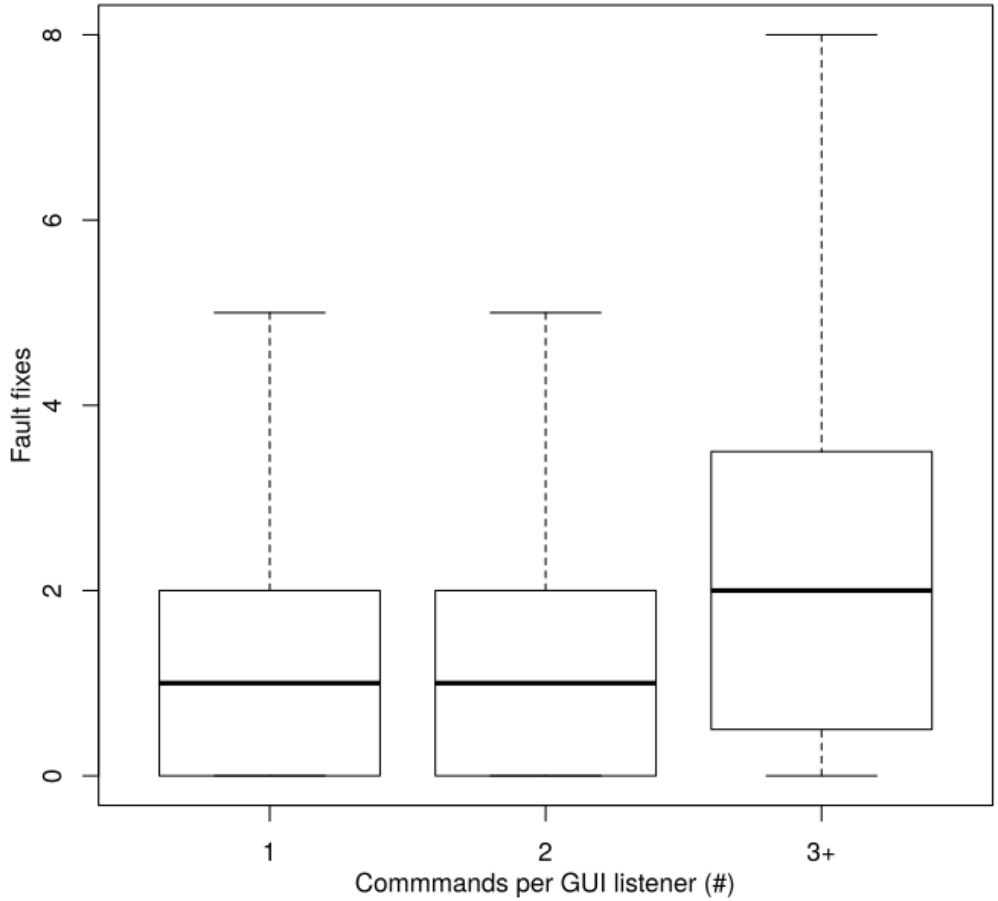
- RQ1: has an **impact on fault-proneness** of the UI listener code?
- RQ2: has an **impact on change-proneness** of the UI listener code?
- RQ3: Does **a threshold value** that can characterize a UI design smell exist?

Empirical study

Evaluate negative impact: # of commands managed by a listener

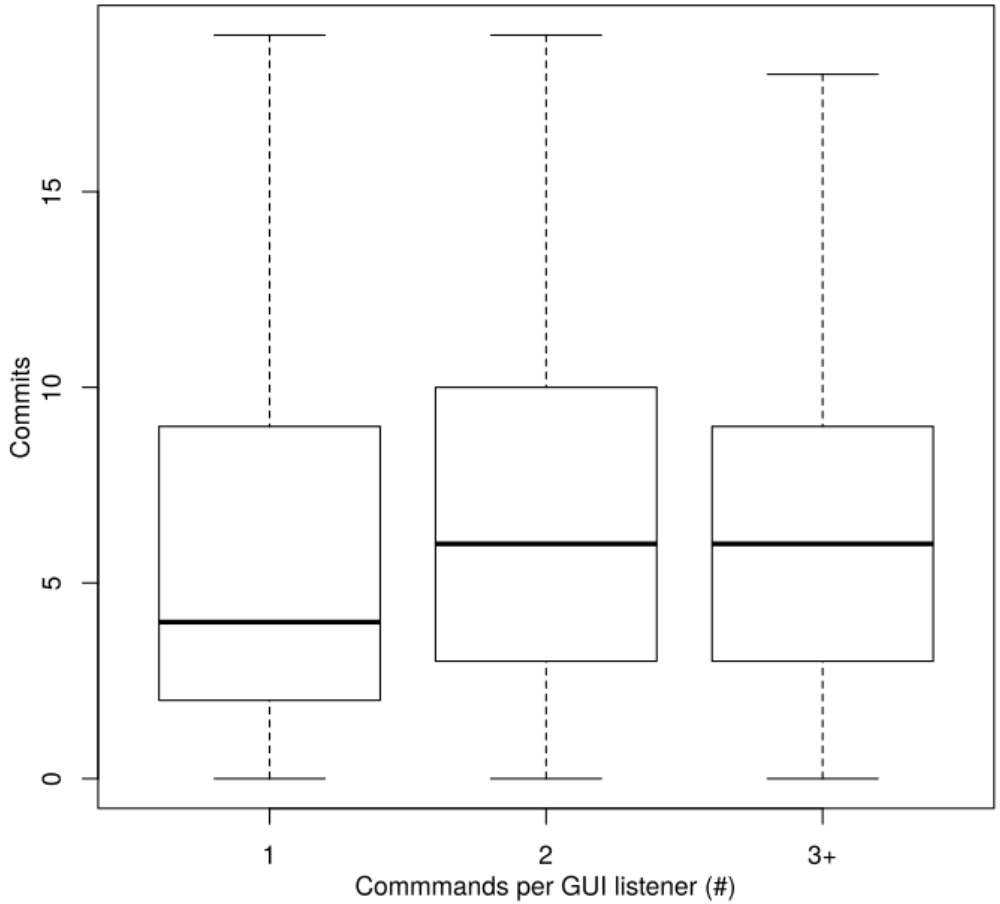
- **Change-proneness**: # of commits per UI listener
- **Fault-proneness**: # of fault fixes per UI listener
  - Bug fixes: commits that contains specific words (“fix”, etc.)
  - Done manually
- Requires apps with **code history (Git)**

RQ1: # of commands per listener has an impact on **fault-proneness**?



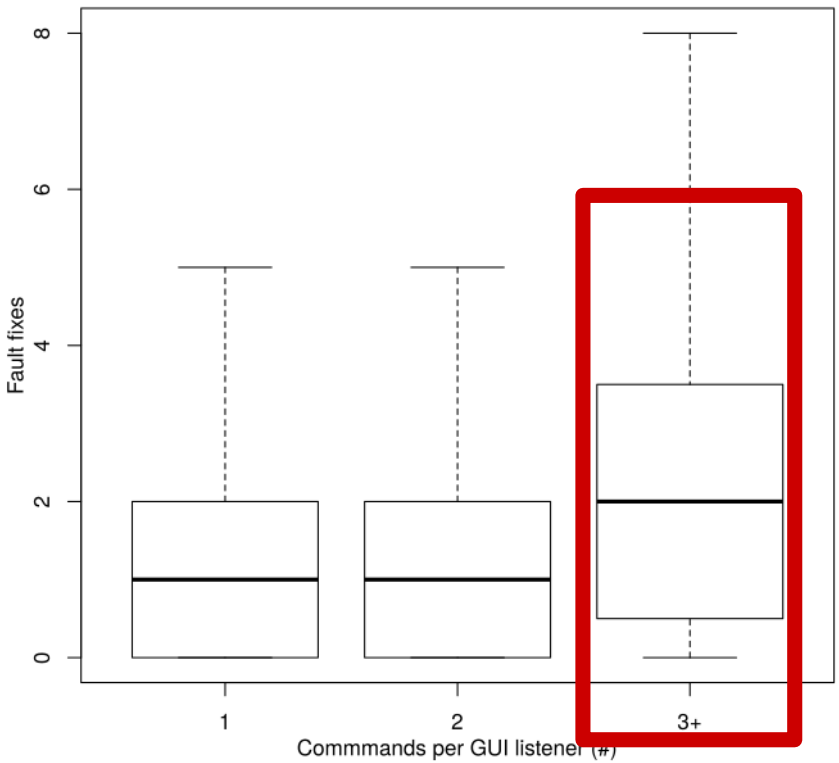
- **Significant** increase of fault fixes for **3+ commands** per listener
  - Cohen's *d* = 0.81 (large), p-value<0.001
- Moderate correlation (0.43)

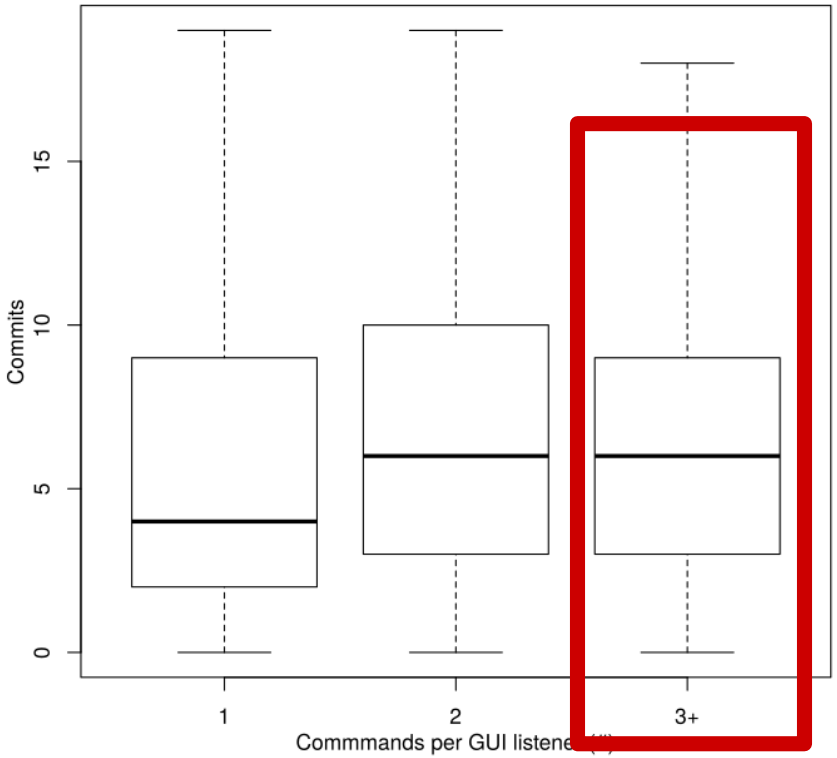
RQ2: # of commands per listener has an impact on **change-proneness**?



- Increase of changes at **3+ commands** per listener
  - Cohen's *d* = 0.5323 (medium), not significant (*p*-value = 0.0564)
- Moderate correlation (0.35)

RQ3: Is there a **threshold value** that can characterize a **UI design smell**?





- Feedback from developers:
  - They confirm the bad coding practice
  - 2 seems a good threshold, but may raise many false positives

=> We define the threshold at 3 commands per listener

(of course: the threshold value is customisable)

Refactoring Blob listeners

Blob Listener: A UI design smell for listener methods that can produce 3 or more UI commands

```java
class B implements ActionListener {
   JButton but1;
   JButton but2;

   B() {
       but1 = new JButton( text: "button1");
       but1.setActionCommand("BUTTON1_ACTION_CMD");
       but1.addActionListener( l: this);

       but2 = new JButton( text: "button2");
       but2.setActionCommand("BUTTON2_ACTION_CMD");
       but2.addActionListener( l: this);
   }

   @Override
   public void actionPerformed(final ActionEvent e) {
       if(e.getActionCommand().equals( anObject: "BUTTON1_ACTION_CMD")) {
           System.out.println( x: "Command 1");
           return;
       }
       if(e.getActionCommand().equals( anObject: "BUTTON2_ACTION_CMD")) {
           System.out.println( x: "Command 2");
           return;
       }
   }
}
```

```java
class B {
    JButton but1;
    JButton but2;

    B() {
        but1 = new JButton( text: "button1");
        but1.addActionListener(e -> System.out.println( x: "Command 1"));
        but2 = new JButton( text: "button2");
        but2.addActionListener(e -> System.out.println( x: "Command2 "));
    }
}
```

Refactoring Blob listeners

Specific code analysis to find the widgets associated to UI commands

```java
class B implements ActionListener {
   JButton but1;
   JButton but2;

   B() {
       but1 = new JButton( text: "button1");
       but1.setActionCommand("BUTTON1_ACTION_CMD");
       but1.addActionListener( l: this);

       but2 = new JButton( text: "button2");
       but2.setActionCommand("BUTTON2_ACTION_CMD");
       but2.addActionListener( l: this);
   }

   @Override
   public void actionPerformed(final ActionEvent e) {
       if(e.getActionCommand().equals( anObject: "BUTTON1_ACTION_CMD")) {
           System.out.println( x: "Command 1");
           return;
       }
       if(e.getActionCommand().equals( anObject: "BUTTON2_ACTION_CMD")) {
           System.out.println( x: "Command 2");
           return;
       }
   }
}
```

```java
class B {
    JButton but1;
    JButton but2;

    B() {
        but1 = new JButton( text: "button1");
        but1.addActionListener(e → System.out.println( x: "Command 1"));
        but2 = new JButton( text: "button2");
        but2.addActionListener(e → System.out.println( x: "Command2 "));
    }
}
```

Widget identification: using string literals, variables, etc. used in (nested) conditional statements

Refactoring Blob listeners

Code refactoring solution to move the UI commands

```java
class B implements ActionListener {
   JButton but1;
   JButton but2;

   B() {
       but1 = new JButton( text: "button1");
       but1.setActionCommand("BUTTON1_ACTION_CMD");
       but1.addActionListener( l: this);

       but2 = new JButton( text: "button2");
       but2.setActionCommand("BUTTON2_ACTION_CMD");
       but2.addActionListener( l: this);
   }

   @Override
   public void actionPerformed(final ActionEvent e) {
       if(e.getActionCommand().equals( anObject: "BUTTON1_ACTION_CMD")) {
           System.out.println( x: "Command 1");
           return;
       }
       if(e.getActionCommand().equals( anObject: "BUTTON2_ACTION_CMD")) {
           System.out.println( x: "Command 2");
           return;
       }
   }
}
```

```java
class B {
    JButton but1;
    JButton but2;

    B() {
        but1 = new JButton( text: "button1");
        but1.addActionListener(e → System.out.println( x: "Command 1"));
        but2 = new JButton( text: "button2");
        but2.addActionListener(e → System.out.println( x: "Command2 "));
    }
}
```

Refactoring Blob listeners

Not possible for all UI commands (49 % of the Blob listeners refactored)

```
public void keyPressed(KeyEvent e) {
  //...
  if (e.isControlDown()) {
    switch (e.getKeyCode()) {
      case KeyEvent.VK_UP:
        frame.getGroupSelector().moveNodeUp(node);
        break;
      case KeyEvent.VK_DOWN:
        frame.getGroupSelector().moveNodeDown(node);
        break;
      //...
}
```

```
InputMap im = textfield.getInputMap();
ActionMap a = textfield.getActionMap();

im.put(KeyStroke.getKeyStroke(KeyEvent.VK_UP,
       InputEvent.CTRL_MASK), "up");
a.put("up", e -> frame.getGroupSelector().moveNodeUp(node));

im.put(KeyStroke.getKeyStroke(KeyEvent.VK_DOWN,
       InputEvent.CTRL_MASK), "down");
a.put("down",e->frame.getGroupSelector().moveNodeDown(node));
```

Feedback from developers regarding Blob listeners

Patches submitted to JabRef, Eclipse, Freecol, and ArgoUML

Patches accepted and merged: JabRef, Freecol

Eclipse: discussions on the patches started, positive feedback, and then… no news

ArgoUML: a dead project?

Feedback from developers regarding Blob listeners

“I like it when the code for defining a UI element and the code for interacting with it are close together. So hauling code out of the action listener routine and into a lambda next to the point a button is defined is an obvious win for me.”

“It does not strictly violate the MVC pattern. […] Overall, I like your solution”

"there might be situations where this can not be achieved fully, e.g. due to limiting implementations provided by the framework."

“It depends, if you refactor it by introducing duplicated code, then this is not suitable and even worse as before”

Software System	LoC (#)	CC (#)	DUP (#)
Eclipse	−40	−45	11
JabRef	−49	−21	0
ArgoUML	−35	−47	13
FreeCol	−146	−37	1
OVERALL	−270	−150	25

Conclusions

- The characterisation of the **Blob Listener** design smell: 3+ commands per listener
- **InspectorGuidget**: an open-source tool that detects and refactors Blob listeners

  https://github.com/diverse-project/InspectorGuidget
- Empirical studies on UI code are not easy to conduct:
  - have to find relevant software systems with UIs (+1 for JabRef and Freecol)
  - UI testing: no or small UI test suites
  - Code analyses may strongly depend on the UI toolkits

Research Agenda

- Static code analyses to:
  - improve UI testing techniques
  - Amplify UI test suites
- Design smell in data bindings