Testing and Maintenance of Graphical User Interfaces

Valéria Lelli
INSA/INRIA, Rennes – France

Jury
Lydie du BOUSQUET (Rapporteur)  Francois-Xavier DORMOY (Examineur)
Philippe PALANQUE (Rapporteur)  Benoit BAUDRY (Directeur)
Pascale SÉBILLOT (Examineur)    Arnaud BLOUIN (Co-encadrant)
Context

- Graphical user interfaces (GUIs)
  - Designed for being controlled by the users
  - Composed of graphical interactive widgets
Validation of GUIs

- GUI designers concern the design and qualitative assessment of GUIs

Arrangement of widgets

Using suitable colors

TEXT EDITORS
Validation of GUIs

- Software engineers ensure that
  - GUIs react correctly to user interactions

Pressing on a button

TEXT EDITORS
Validation of GUIs

- Software engineers ensure that
  - GUIs react correctly to user interactions

produces the expected action
GUI testing

Numerous and different kinds of widgets

GUI faults are multiple and diverse

GUI testing tools

GUI failures
GUI testing techniques

- Capture and Replay tools
  - Recording user interactions to be replayed
- Monkey tools
  - Sending random events such as mouse events
- Functional GUI testing tools
  - Pre-defined libraries to write test cases
- Event-flow graphs
  - Based on the sequence of events to automatically generate test cases
  - GUI failures from the recent GUI developments
Graphical User Interfaces

Users

GUI source code

Human input device

Graphical elements

TEXT EDITORS

GUI source code

Users interact with the interface through graphical elements using a human input device (like a mouse or keyboard). The graphical user interface (GUI) is the visual and interactive component of software that allows users to interact with the computer through the interface. The GUI source code is the programming code that generates the graphical elements and processes user input.
Validation of GUIs

GUI source code

60% of the total software

Few works focus on GUI code analysis
GUI code analysis

- **GUI design smells**
  - Bad coding practices that degrade GUI source code

- **Bug finder tools**
  - FindBugs and PMD do not focus on detect problems that affect the GUI source code
  - Absence of GUI metrics/rules to detect GUI design smells
Contributions

GUI fault model

GUI source code analysis
GUI Fault Model
Recent developments of GUIs involve more advanced user interactions

Current GUI testing tools focus on finding bugs in classical GUIs

How the characteristics of recent developments of GUIs impact on GUI testing?
Post-WIMP GUIs

- Ad hoc widgets such as drawing areas
- Complex interactions: multi-touch, etc.
WIMP vs. post-WIMP GUIs

Event-based GUIs
- Standard widgets
- Mono-event interactions

Interaction-based GUIs
- Ad hoc widgets
- Multi-event interactions

✔ New problems of GUI faults
✗ Current GUI testing tools
GUI Fault Model

- **Objectives**
  - Baseline to evaluate the effectiveness of GUI testing techniques
  - Developing GUI testing techniques

GUI Fault Model

Describe how GUI faults come to be and how and why they occur as a GUI failure
- **Structure and behavior of the graphical components**
The interaction process when a user interacts with a GUI
Concrete examples of user interface faults

//Set widget properties
7. widget.setVisible(true);
8. widget.setAlignment(5);
...

- E.g. of GUI Fault: incorrect vs. correct lines of GUI code
- E.g. of GUI failure: a widget is not visible
Concrete examples of user interface faults

- *E.g. of GUI Fault*: incorrect vs. correct lines of GUI code
- *E.g. of GUI failure*: a widget is not visible
Concrete examples of user interface faults

- E.g. of GUI Fault: incorrect vs. correct lines of GUI code
- E.g. of GUI failure: a widget is not visible

... //Set widget properties
7. copyMenu.setVisible(false);
8. btnRedo.setAlignmentY(10);
...

Widgets are not aligned
Incorrect layout of widgets fault
Interaction behavior
1. figures.firstElement().onDragged(formerPt, newPt);
Concrete examples of user interaction faults

Not possible to move a shape since the drag is incorrectly processed

Interaction behavior
1. `figures.firstElement().onDragged(newPt, formerPt);`
Fault model assessment

RQ1: Is the GUI fault model relevant against real GUI failures?

RQ2: Are GUI testing tools able to detect the failures classified in our fault model?
Experiment (RQ1): relevance

- GUI bug reports of 5 open-source software systems
  - Sweet Home 3D
  - File-roller
  - JabRef
  - Inkscape
  - Firefox Android

- **Manual analysis** of the real GUI bug reports
  - Source forge, bugzilla, etc.
  - Root cause: description, patches, comments, or stack traces

Several kinds of widgets, interactions and platforms
Experiment Results

- All GUI failures (279) were classified into the fault model
- User interface (41%) and user interaction (59%) (Post-WIMP)
  - 25% (user interface)
  - 18% (user interaction)
Experiment Results

- 43% into **Action** and
- 27% into **GUI structure and aesthetics**
1% of GUI failures classified into **Feedback**
Several “failures” were considered by developers as improvements
RQ1: Is the GUI fault model relevant against real GUI failures?

✓ All GUI related faults of 5 large scale case studies can be classified
  ✓ 279 GUI bug reports

✓ All the 6 fault categories are covered

✓ Faults concern WIMP and post-WIMP GUIs
  ✓ Ad hoc widgets (59 faults)
Experiment (RQ2): GUI testing tools

- **JabRef**: selected 11 out of 15 GUI faults
Experiment (RQ2): GUI testing tools

- **GUITAR**
  - Most popular academic tool in GUI testing
  - Automated test cases generation
  - Event-flow graph is built by reverse engineering

- **Jubula**
  - Partially manual generation of test cases
  - Reuse pre-defined libraries to create manually test cases

---


3http://www.eclipse.org/jubula
Experiment Results

- **GUITAR** detected
  - 3 out of 11 GUI faults
- **Jubula** detected
  - 9 out of 11 GUI faults

- GUI failures detected
  - ✔ Properties of standard widgets
  - ✔ Crashes
  - ➔ Oracle for standard widgets
Experiment Results

- **GUITAR**
  - Missed 8 out of 11 GUI faults reported in JabRef

- GUITAR builds the event-flow graph by
  - Extracting the sequence of events behind standard widgets
  - Collecting the information in the properties of standard widgets as event logs

- ✗ User interface failures into properties of standard widgets
- ✗ Complex data in ad hoc widgets
- ✗ Events are both widgets and their underlying interactions
- ✗ Ad hoc widgets and their multi-event interactions
Real example of a GUI failure
Real example of a GUI failure

Incorrect Feedback
RQ2: Are GUI testing tools able to detect the classified failures?

✓ Most of GUI faults concern standard widgets

✗ Faults that concern the interactive features such as feedback and reversibility
Conclusion

- An empirical study of real GUI failures
  - 279 GUI-related bug reports

- Evaluation of two GUI testing tools against
  - Real GUI failures into standard and ad hoc widgets
  - 65 GUI mutants derived from our fault model
    - 43 GUI mutants were not killed

- A precise analysis of standard GUI testing frameworks
  - Why GUI failures that stem from GUI faults described in our fault model were not detected?
Contributions

GUI fault model

GUI source code analysis
GUI Design Smells: The case of Blob Listener
Objectives

- Identify and characterize design smells that degrade the GUI code quality
- Develop a novel static analysis to detect GUI design smells
GUI implementations

View

Represent of the GUI elements

Controller

Receive the events

Model

Data model of an interactive system

Listener methods

events

View

Controller

Model
GUI implementations

- Specific architectural design patterns
  - Organize the GUI components
  - Describe how the components interact with each other

- Mode-View*
  - Model-View Controller (MVC)
  - Mode-View Presenter (MVP)
  - Model-View-ViewModel (MVVM), etc.
AController manages events produced by three widgets (b1, b2, and m3)
A GUI listener produces **several** GUI commands

```java
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
        }
        //...
    }
}
```

**GUI command** is a set of statements executed in reaction of a user interaction
Empirical Study on GUI listeners

- 13 open-source software systems
  - Github repository that use an issue-tracking system
  - Large Java systems
  - GUI size: 858 GUI listeners

- Metrics
  - Average commits
  - Average fault fixes
  - Number of commands
The number of commands per GUI listeners has a negative impact on fault-proneness of listeners code.
Results

✓ Establish a threshold value to **at least three commands per listener**
✓ 21% of the analyzed GUI listeners are *Blob listeners*

*Blob Listener* is a GUI listener that produces **more than two** GUI commands
Blob Listener detection
GUI listeners are analyzed to identify GUI listeners that have at least one conditional statement.
Blob Listener detection

Source code + libraries

```java
public class JWhiteBoard extends ReceiverAdapter {
    implements ActionListener, ChannelListener {
        //...
        clearButton=new JButton("Clean");
        clearButton.addActionListener(this);
        leaveButton=new JButton("Exit");
        leaveButton.addActionListener(this);
        //...more than 150 lines of code

        @Override public void actionPerformed(ActionEvent e) {
            String command=e.getActionCommand();
            if("Clear".equals(command)) {//GUI fault in Command #1
                if(noChannel) {
                    clearPanel();
                    return;
                }
                sendClearPanelMsg();
            }
            else if("Leave".equals(command)) {//GUI fault in Command #2
                stop();
            } //...
        }
    }
}
The conditionals are analyzed to detect any reference to a GUI event or widget.
The conditionals are analyzed to detect any reference to a GUI event or widget.
Blob Listener detection

```java
@Override public void actionPerformed(ActionEvent e) {
    Object src = e.getSource();
    if (src instanceof JMenuItem || src instanceof JButton) {
        String cmd = e.getActionCommand();
        if (cmd.equals("Copy")) { // Command #1
            if (selectedText) output.copy();
        } else if (cmd.equals("Cut")) { // Command #2
            output.cut();
        } else if (cmd.equals("Paste")) { // Command #3
            output.paste();
        }
    } // etc.
}
```

✓ The nested commands are removed
GUI listeners that contain more than two GUI commands are marked as Blob Listener
Blob Listener detection

```java
public class CalculatorLayout extends JFrame implements ActionListener{
    private JButton bOne = new JButton("1");
    private JButton bTwo = new JButton("2");
    private JButton bSin = new JButton("sin");
    private JButton bCos = new JButton("cos");
    private JTextField tfDisplay = new JTextField(); //result displaying screen
    private JTextField tfRawInput = new JTextField();
    //... more 41 declarations/instantiation of swing widgets variables

    @Override public void actionPerformed(ActionEvent e){
        Object src = e.getSource();
        if (e.getSource() == bOne){//Command #1
            if (operation == '-='){
                sDisplay = "1";
                sRawInput = "1";
                tfRawInput.setText(sRawInput);
                operation = '';
            }
            else{
                sDisplay = sDisplay + "1";
                sRawInput = "1";
                tfRawInput.setText(sRawInput);
            }
        } //... more 15 GUI commands to handle buttons bTwo, etc.
        else if (e.getSource() == bEqual && !sDisplay.equals("")){//Command #18
            double number2 = Double.parseDouble(sDisplay);
            if (operation == '+'){
                result = number1 + number2;
            }
            else if (operation == '-='){
                result = number1 - number2;
            }
            else if (operation == '*=')
                result = number1 * number2;
            /* more 3 "else if" conditional statements */
            String temp = "";
            if (isPoint operation == '/'{tfDisplay.setTex(temp = ""+result; temp = ""+result;"; sDisplay = ""; number1 = result; isPlus = true; isPoint = false; isOperation = true; sRawInput = ""; tfRawInput.setText(sRawInput); sRawInput = temp; operation = '=';
        }
        //... more 21 GUI commands to handle buttons bsin, bcos, etc.
    }
}
```

1 Blob listener with 39 GUI commands

Blob Listeners Detection

two GUI commands
InspectorGuidget

- Open-source tool as an Eclipse plug-in dedicated to Java GUI systems

<table>
<thead>
<tr>
<th>Software System</th>
<th>Successfully Detected Blob listeners (#)</th>
<th>FN (#)</th>
<th>FP (#)</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastPhotoTagger</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>GanttProject</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>JaxoDraw</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>100.00</td>
<td>87.50</td>
</tr>
<tr>
<td>Jmol</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>91.67</td>
<td>81.82</td>
</tr>
<tr>
<td>TerPaint</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>TripleA</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>37</strong></td>
<td><strong>1</strong></td>
<td><strong>1</strong></td>
<td><strong>97.59</strong></td>
<td><strong>97.37</strong></td>
</tr>
</tbody>
</table>

37 out of 38 Blob listeners were detected

7https://github.com/diverse-project/InspectorGuidget
Conclusion

- **A new type of GUI design smell**
  - *Blob listener* has an negative impact on fault-proneness of GUI listeners

- **A novel static analysis approach**
  - InspectorGuidet dedicated to Java systems
  - 37 out of 38 instances of *Blob listeners* on six real-world GUI systems

- **Good coding practices** to avoid the presence of *Blob listeners*
Conclusions & Perspectives
Conclusions

- **GUI fault model**
  - 279 GUI-related bug reports of five interactive open-source systems
  - Evaluation of GUI testing frameworks against real GUI failures and GUI mutants

- **An automatic detection of a new type of GUI design smell**
  - *Blob listener* that degrades the GUI code quality
  - InspectorGuidget detected 37 out 38 instances of *Blob listeners*
Conclusions

- **Experiment and tools**
  - A complete data set
  - GUI systems that have several interactive features
  - Empirical studies of GUI implementations
### Perspectives

- **Domain-specific mutants**
  - Mapping between GUI faults and specific GUI toolkits

#### Java Swing mutants

```java
public class CalculatorLayout extends JFrame implements ActionListener {
    private JButton button1 = new JButton("1");
    private JButton button2 = new JButton("2");
    private JButton button3 = new JButton("3");
    private JButton button4 = new JButton("4");
    private JButton button5 = new JButton("5");
    private JButton button6 = new JButton("6");
    private JTextField display = new JTextField();
    private JLabel result = new JLabel();

    public CalculatorLayout() {
        super("Calculator");
        setLayout(new GridLayout(5, 2));
       add(button1);
        add(button2);
        add(button3);
        add(button4);
        add(button5);
        add(button6);
        add(display);
        add(result);
    }

    public void actionPerformed(ActionEvent e) {
        String input = display.getText();
        switch (e.getActionCommand()) {
            case "1":
                display.setText(input + "1");
                break;
            case "2":
                display.setText(input + "2");
                break;
            case "3":
                display.setText(input + "3");
                break;
            case "4":
                display.setText(input + "4");
                break;
            case "5":
                display.setText(input + "5");
                break;
            case "6":
                display.setText(input + "6");
                break;
        }
    }
}
```

#### GUI Fault Model

![GUI Fault Model Diagram](image-url)
**Perspectives**

- GUI design smells
  - A set of checking rules to check automatically for potential defects in GUI code

```java
public class Listener implements ActionListener{
    public Listener(ActionEvent event){
        // Do the initialization
        // Register the listener on a buttonA
        buttonA.addActionListener(this);
    }
    public void actionPerformed(ActionEvent event){
        Object source = event.getSource();
        if(source == buttonA){
            // Manage the event...
        }
    }
}
```
GUI design smells

Bug finders
- Findbugs
- PMD, etc.
Publications

- Valéria Lelli, Arnaud Blouin, and Baudry Benoit. **Classifying and qualifying GUI defects.** In Software Testing, Verification and Validation (ICST), 2015 IEEE Eighth International Conference, pages 1–10, April 2015.

- Valéria Lelli, Arnaud Blouin, Baudry Benoit, and Fabien Coulon. **On model-based testing advanced GUIs.** In 11th Workshop on Advances in Model Based Testing (A-MOST), pages 1–10, April 2015. *Best paper award.*
