AUTOMATED GUI LAYOUT REFACCTORING TO IMPROVE MONKEY TESTING OF ANDROID APPLICATIONS

Samad Paydar
s-paydar@um.ac.ir
AUTOMATED GUI LAYOUT REFACTORED TO IMPROVE MONKEY TESTING OF ANDROID APPLICATIONS

Samad Paydar

I’ve received my Ph.D. in Computer Engineering from Ferdowsi University of Mashhad (FUM), Iran in 2015. Since then, I’ve been an assistant professor of the Computer Engineering Department at FUM. I have always been interested in programming and software engineering. My main research interest includes software testing in general, and specifically testing Android applications.
OUTLINE

Problem Statement
Related Work
Proposed Approach
Evaluation
conclusion
PROBLEM STATEMENT
**PROBLEM STATEMENT**

Android operating system has the highest market share of mobile operating systems.

A large number of Android apps on app stores like Google Play and F-Droid.

Emphasizes the need for effective testing techniques for Android apps.
**PROBLEM STATEMENT**

*Monkey Testing* is a random testing based technique

The official Android SDK includes a monkey tool for monkey testing of Android apps

- It automatically generates a stream of pseudo-random events on the application’s GUI to see if the app crashes or shows any undesirable behavior, e.g. issuing Application Not Responding (ANR) errors

- Can detect robustness and responsiveness faults
Despite it’s benefits, there exist concerns about the effectiveness of Google’s monkey

**Widget-Obliviousness**

- Monkey has no knowledge about the widgets
  - e.g. location of widgets
- Many events are invalid or useless
  - Clicking on an empty space of GUI
  - Click on a static GUI widget
- No attention is paid to the importance of GUI elements

**State-Obliviousness**

- Monkey does not remember visited states of the app
- Generates events that repeatedly change the state of the application between a few states

**Goal:** to improve effectiveness of monkey testing by tackling widget-obliviousness
The main line of research

- To propose new monkey testing tools or random testing methods
  - Employing new event generation strategies
  - Utilizing knowledge of the GUI
- For instance,
  - EHBDroid [1] directly calls event handlers, instead of indirectly calling them through firing events on widgets
  - Sapienz [2] uses search-based techniques to minimize the length of the test sequence, while maximizing code coverage and fault detection.
  - Dynodroid [3] enhances random exploration strategy of monkey by providing frequency strategy and biased strategy
  - [4] Supports a wider range of user and system events
    - Dynamically builds GUI transition graph, and uses this graph in a biased random-walk exploration algorithm
RELATED WORK

All the existing approaches are implicitly based on the idea that it’s only the monkey that is responsible for the lack of test effectiveness.

We propose a totally different approach by focusing on the application under test.
PROPOSED APPROACH
UNDERLYING IDEA

If monkey testing is not as effective as expected
- Monkey is not the only one to put the blame on
- The application under test is also responsible

Monkey testing is performed through application GUI

GUI is designed to be user-friendly
- How the user likes to see the GUI

Not monkey-friendly
- How the monkey likes to see the GUI
Why not have a second version of the app, specifically for monkey testing

▪ A monkey-friendly version

Our proposal:
▪ A fully automated technique to refactor a given app to create a monkey-friendly version, by conducting GUI refactoring, preserving the application behavior
**TECHNIQUE**

**Input:** Source code of the AUT

**Output:** Refactored version of the AUT
TECHNIQUE :: PROJECT ANALYSIS

To identify the XML layout files that define AUT’s GUI

Each identified XML layout file is passed to the next step
TECHNIQUE :: LAYOUT ANALYSIS

Uses static analysis of both XML and Java files

To identify *interactive GUI elements*, i.e. elements that
- Have an event handler, e.g. onClick listener
- Of serve as an input field, the value of which is accessed by another event handler (either directly or indirectly)
  - We call that event handler an *accessor method*

**Output**: a list of interactive elements along with their associated event handler or accessor methods
For 1st type of interactive elements

- Identification of interactive elements is challenging
- Due to different methods for setting event handlers

- Method 1
  - The event handler is specified in the XML layout file
  - Where the GUI element is defined

- This case can be handled by simply parsing the XML layout file, to find the name of the event handler
For 1st type of interactive elements

- Identification of interactive elements is challenging
- Due to different methods for setting event handlers

- Method 2
  - Event handler is set in the Java code, not in the XML file
  - Using specific annotations

- To handle this case, both static analysis of Java code and parsing XML layout files is required.
For 1\textsuperscript{st} type of interactive elements

- Identification of interactive elements is challenging
- Due to different methods for setting event handlers

- Method 3
  - Event handler is set in the Java code using setter methods

- To handle this case, more advanced static analysis of Java code and parsing XML layout files is required.
For 2nd type of interactive elements

- Finding GUI elements that their value is accessed in some event handler method

- To handle this case, more advanced static analysis of Java code and parsing XML layout files is required.

```java
public void someMethod(View view) {
    EditText usernameField = (EditText) findViewById(R.id.username);
    String username = usernameField.getText();
    ...
}
```
For each interactive element identified in the previous step

- The associated event handler or accessor method is processed to compute a complexity value for
We have proposed a complexity measure to compute complexity of a given event handler or accessor method.

\[ \text{cmp}(m) = \text{cycCmp}(m) + \text{callCmp}(m) + \text{apiCmp}(m) + \text{intentCmp}(m) + \text{asyncCmp}(m) \]
We have proposed a complexity measure to compute complexity of a given event handler or accessor method:

\[ \text{cmp}(m) = \text{cycCmp}(m) + \text{callCmp}(m) + \text{apiCmp}(m) + \text{intentCmp}(m) + \text{asyncCmp}(m) \]
TECHNIQUE :: LAYOUT REFACTORIZATION

For each XML layout file L

- First a backup is taken from L
- Then L is given to our refactoring algorithm
- The result XML layout file L’ replaces L
Two LinearLayouts are created

One for the interactive elements
- All the available GUI space is dedicated to this layout
- The interactive elements are sized based on the complexity of their associated event handler or accessor method
- The more complex is the method associated with an interactive element, the more space is given to that element

The other for the rest of the elements
IMPLEMENTATION

We have implemented the proposed technique as a plugin for Android Studio IDE.
CASE STUDY :: SATRAP

An Android application developed at Ferdowsi University of Mashhad

- To be used by the people in the Protective Services Department
- For registering, monitoring and reporting the driving fines issued for traffic violations.
- Many network-related functionalities.
- Many database-related functionalities

- The application size is about 7K SLOC
- There are 38 interactive elements in the GUI, and 42 methods associated with these elements.
- There are a total of 418 methods.
**METRICS**

**Hit Count (HC)**
- The number of times a method is executed in response to the events fired by monkey

**Method Coverage (MC)**
- The percentage of the methods of AUT that are executed, at least once, during the monkey testing session
PROCEDURE

1. Execute a monkey testing session with on the original version of the AUT.
2. Compute the evaluation metrics for this session.
3. Apply the proposed technique on the original version of the AUT to get the refactored version.
4. Execute a monkey testing session on the refactored version, with the same setting as the first one.
5. Compute the evaluation metrics for the second session.
6. Compute the changes in the evaluation metrics in the two sessions.
REFACTORING EXAMPLE

Original version of the Main activity
REFACTORING EXAMPLE

Original version of the Main activity

- App Title and Version Info.
- App Logo
- ‘Register Fine’ Button
- ‘Report’ Button
- ‘Change Password’ Button
- ‘About’ Button
- ‘Exit’ Button
- Copyright Info.

Refactored version of the Main activity
EXPERIMENT SETTINGS

Average over 100 executions

Monkey is executed with $n = 10K$ events

We have considered two sets of target methods
- The set of 42 methods associated with interactive elements
- The set of all 418 methods in the AUT
RESULTS

For the first set

- HC is increased by a factor of 48%
- MC is increased by a factor of 12%

For the second set

- HC is increased by a factor of 69%
- MC is increased by a factor of 9%
## RESULTS

<table>
<thead>
<tr>
<th>Method Name</th>
<th>HC1</th>
<th>HC2</th>
<th>HC Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFine</td>
<td>1</td>
<td>7</td>
<td>600</td>
</tr>
<tr>
<td>remoteSearch</td>
<td>3</td>
<td>13</td>
<td>333</td>
</tr>
<tr>
<td>localSearch</td>
<td>4</td>
<td>14</td>
<td>250</td>
</tr>
<tr>
<td>showReport</td>
<td>4</td>
<td>13</td>
<td>225</td>
</tr>
<tr>
<td>editFine</td>
<td>5</td>
<td>14</td>
<td>180</td>
</tr>
</tbody>
</table>

Top-5 methods with the highest increase in HC

<table>
<thead>
<tr>
<th>Method Name</th>
<th>HC1</th>
<th>HC2</th>
<th>HC Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>exit</td>
<td>9</td>
<td>5</td>
<td>44</td>
</tr>
<tr>
<td>showAboutUs</td>
<td>4</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>backToMain</td>
<td>4</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>cancelAddFine</td>
<td>4</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>cancelEditFine</td>
<td>9</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Top-5 methods with the highest decrease in HC
CONCLUSION

A new technique is proposed to improve effectiveness of monkey testing of Android apps.

The main idea is automated GUI layout refactoring.

The technique is implemented as a plugin for Android Studio IDE.

A case study is conducted as an initial evaluation of the technique, demonstrating that the technique is promising.
THANK YOU FOR YOUR ATTENTION!
REFERENCES
REFERENCES


