AGRippin: A Novel Search Based Testing Technique for Android Applications

Domenico Amalfitano,
*Nicola Amatucci*,
Anna Rita Fasolino,
Porfirio Tramontana
Context & Motivation

- Automated GUI Testing Techniques for Mobile Applications
  - Capture&Replay, Model-Based, Model-Learning and Random Techniques
  - Model-Learning and Random Techniques don’t need information about the application under test
  - Fully automated generation of a Test Suite
- Model-Learning vs Random Techniques
  - Test-Suites generated by Random Techniques are able to reach an higher coverage of the application under test, but they are more complex
AGRippin

- Android Genetic Ripping
  - GUI Ripping Technique
    - Model-Learning Technique
  - Genetic Algorithm Implementation
- The purpose of the technique is to generate test suites that are more effective than the ones obtained by the Model-Learning technique, without increasing the test suite size

DeMobile 2015 – Bergamo, Italy – August 31
Android Ripper

- Implements a set of GUI Ripping Techniques
- The GUI showing on the screen is analyzed and a list of fireable User and System events is built
  - After the execution of each event the GUI shown is analyzed and a new list of events is built
  - Each event is performed according to a Breadth-First, Depth-First or Random strategy
    - The tool creates sequences of events by visiting the GUI
    - The GUIs visited are added to a list
- A Test Suite is built reproducing to the sequences of executed events

Genetic Algorithm

- Representation
  - Test Case Representation
- Crossover Technique
  - Single-Point Crossover
- Mutation Technique
- Fitness Evaluation
  - Global & Local Fitness
- Selection Technique
  - Rank-Based Selection
- Combination Technique
  - Further exploration of unvisited GUI Interfaces
Test Cases Representation

GUI Interface Instance

Widget

Action

Touch

GUI Interface Instance
Test Cases Representation

- A Test Case is a sequence of GUI Interface Instances and Actions
Crossover Technique

Single-Point Crossover!
Equivalence Criteria

- Two **GUI interfaces** are considered **equivalent** if they include the same set of widgets and they define the same set of event handlers.

- Two **actions** are considered **equivalent** if they are associated to the same user actions and the same event.
Crossover Technique

Single-Point Crossover!

- Equivalent interfaces
  - Try to generate executable Test Cases
- Try not to generate too similar Test Cases
  - Not equivalent actions
  - Not equivalent subsequences
  - Not empty subsequences
Crossover Technique

Single-Point Crossover!

• Generated test cases are added to the Test Suite, if they are actually executable

• Using the given equivalence criteria we try to ensure the generation of executable sequences of events
Mutation Technique

- Input values generated by fully automated techniques are usually random or constants.
- Our mutation operator changes these values randomly:
  - Addresses, Names, Numbers, Paths, Email, Dates, Time Values, ...
- Test cases are added to the test suite:
  - The original test cases are kept into the test suite.
Fitness Evaluation

• The Test Suite is executed and the code coverage of each Test Case is evaluated

• Global Fitness Function
  ◦ Test Suite Fitness
    • Percentage of LOCs covered by the Test Suite

• Local Fitness Function
  ◦ Test Case Fitness
    • Diversity between the sets of LOCs covered by each Test Case
L1, test cases that cover at least a line not covered by other test cases
L2, test cases that cover a set of lines also covered by the union of the coverage sets of other test cases
L3, test cases that cover a set of lines completely covered by at least one different test case

<table>
<thead>
<tr>
<th>L1</th>
<th>TC1</th>
<th>L2</th>
<th>TC5</th>
<th>L3</th>
<th>TC2</th>
<th>L3</th>
<th>TC3</th>
<th>L1</th>
<th>TC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td></td>
<td></td>
<td></td>
<td>l1</td>
<td></td>
<td>l1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l2</td>
<td></td>
<td></td>
<td></td>
<td>l2</td>
<td></td>
<td>l2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l3</td>
<td></td>
<td></td>
<td></td>
<td>l3</td>
<td></td>
<td>l3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l4</td>
<td></td>
<td></td>
<td></td>
<td>l4</td>
<td></td>
<td>l4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l5</td>
<td></td>
<td></td>
<td></td>
<td>l5</td>
<td></td>
<td>l5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l6</td>
<td></td>
<td></td>
<td></td>
<td>l6</td>
<td></td>
<td>l6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l7</td>
<td></td>
<td></td>
<td></td>
<td>l7</td>
<td></td>
<td>l7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l8</td>
<td></td>
<td></td>
<td></td>
<td>l8</td>
<td></td>
<td>l8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l9</td>
<td></td>
<td></td>
<td></td>
<td>l9</td>
<td></td>
<td>l9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l10</td>
<td></td>
<td></td>
<td></td>
<td>l10</td>
<td></td>
<td>l10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TCx Test Case
li Line of Code
Local Fitness Evaluation

L1, test cases that cover at least a line not covered by other test cases

L2, test cases that cover a set of lines also covered by the union of the coverage sets of other test cases

L3, test cases that cover a set of lines completely covered by at least one different test case

\[ F_2(t) = \sum_{l \in \text{Cov}(t)} w(l) \]

- \( \text{Cov}(t) \) the set of lines \( l \) covered by \( t \)
- \( w(l) = \frac{1}{\sum_{u \in T} c(u)} \) the relative weight of the coverage of the line \( l \)
- \( c(u) \in \{0, 1\} \), 0 if \( l \notin \text{Cov}(u) \)
Selection Technique

- Test Cases are ranked using the Local Fitness Function

<table>
<thead>
<tr>
<th>RANK</th>
<th>t</th>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TC4</td>
<td>L1</td>
<td>2.98</td>
</tr>
<tr>
<td>2</td>
<td>TC1</td>
<td>L1</td>
<td>2.23</td>
</tr>
<tr>
<td>3</td>
<td>TC5</td>
<td>L2</td>
<td>1.23</td>
</tr>
<tr>
<td>4</td>
<td>TC6</td>
<td>L2</td>
<td>0.91</td>
</tr>
<tr>
<td>5</td>
<td>TC2</td>
<td>L3</td>
<td>0.98</td>
</tr>
<tr>
<td>6</td>
<td>TC3</td>
<td>L3</td>
<td>0.65</td>
</tr>
</tbody>
</table>

turnover ratio = 1/3

- A set of Test Cases is deleted according to a «turnover ratio»
Combination Technique

- GUI Interface Instances not visited by the GUI Ripping Technique can be discovered by test cases
  - It’s probably possible to generate a new set of Test Cases
- The discovered GUIs are used as starting points for the GUI Ripping Technique
  - The generated Test Cases are added to the Test Suite
Experimentation

- **RQ:** Are the test suites generated by the proposed technique more effective that the ones generated by the considered GUI Ripping Technique?

- **Effectiveness**
  
  $\eta(T) = 100 \times \frac{|\bigcup_{t \in T} Cov(t)|}{|LOC|}$

DeMobile 2015 – Bergamo, Italy – August 31
## Subjects & Configuration

<table>
<thead>
<tr>
<th>AUT</th>
<th>Application</th>
<th>LOCs</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUT1</td>
<td>AardDict 1.4.1</td>
<td>2308</td>
<td>7</td>
</tr>
<tr>
<td>AUT2</td>
<td>TomDroid 0.7.1</td>
<td>4167</td>
<td>10</td>
</tr>
<tr>
<td>AUT3</td>
<td>OmniDroid 0.2.1</td>
<td>6770</td>
<td>16</td>
</tr>
<tr>
<td>AUT4</td>
<td>AlarmClock 1.7</td>
<td>2320</td>
<td>5</td>
</tr>
<tr>
<td>AUT5</td>
<td>BookWorm 1.0.18</td>
<td>3190</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossover ratio</td>
<td>20%</td>
</tr>
<tr>
<td>Mutation Ratio</td>
<td>5%</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>30</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>AUT</th>
<th>GUI Ripper</th>
<th>AGRippin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\eta(t)$</td>
<td>$\mu(\eta(t))$</td>
</tr>
<tr>
<td>AUT1</td>
<td>43.07%</td>
<td>67.10%</td>
</tr>
<tr>
<td>AUT2</td>
<td>28.08%</td>
<td>32.61%</td>
</tr>
<tr>
<td>AUT3</td>
<td>51.58%</td>
<td>58.31%</td>
</tr>
<tr>
<td>AUT4</td>
<td>66.90%</td>
<td>68.00%</td>
</tr>
<tr>
<td>AUT5</td>
<td>40.34%</td>
<td>47.22%</td>
</tr>
</tbody>
</table>

- We repeated each experiment with AGRippin six times
  - Six different random seeds
- For each application we noticed an increase in the effectiveness $\eta(T)$
Conclusions

- We proposed a novel search based testing technique
- We evaluated its effectiveness by carrying out a case study involving five Android Applications comparing it to a GUI Ripping Technique

Future Works

- Extend the experimentation with a larger set of applications and techniques
- Study the influence of the configuration parameters on the effectiveness of the generated Test Suite
- Implement variants of the Genetic Algorithm Implementation we proposed
Thank you !!!