Collaborative bug finding and bug-fixing for Android Apps

Shin Hwei Tan
Southern University of Science and Technology
Accepted in ICSE2020 with Ziqiang Li
"I have not found any easy-to-use testing solutions for Android"

"it’s hard to write the useful test case for current project, because the requirement is changed very often, and the schedule is very tiny. So we still prefer to hire some tester to do manually testing. In the other hand, the android test framework is not good enough yet, I tried study roboelectric, it’s a little bit hard to understand."

"A lot of what I do is related to how the app looks and feels. Therefore a lot of my testing is done manually..."

- Developers mostly rely on manual testing and unit testing.
- Developers prefer automatically generated test cases in natural language

from “How do Developers Test Android Applications?” [ICSME’17]
Many bug reports exist in repositories like GitHub.

App developers who developed PocketHub and ForkHub found similar bugs across two apps.

<table>
<thead>
<tr>
<th>Bug report’s title in CameraColorPicker</th>
<th>Bug report’s title in Gnucash</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do I get left top color.</td>
<td>When an account is edited, its color is lost.</td>
</tr>
<tr>
<td>publish to F-Droid</td>
<td>Publishing on F-Droid</td>
</tr>
<tr>
<td>Make gradlew executable</td>
<td>Make gradlew executable</td>
</tr>
</tbody>
</table>

There exists one-to-one correspondence for the bug report’s title in apps of different categories!
Crafting test scenario from bug report

- A test scenario includes:
  - steps to reproduce
  - test data (e.g., an image for image processing app)
  - the expected behavior
- Could solve the test oracle problem
How does collaborative bug finding works?

- Emulate the role of a competent pair programmer via developers of other similar applications

**PAIR PROGRAMMING**

- Designed 3 settings to model different interactions between coders
  - Coders-vs-Coders
  - Coder-vs-Manual-Issues
  - Coder-vs-Auto-Issues (Bugine)
Setup of the study for the first 2 settings

- Use GitHub Classroom for assignments
  - All students for a class belongs to the same organization

- Each student selects 1 app
- <3 teams could select the same app
  - Selected 20 apps based on:
    - Ease of use
    - Contains existing tests
    - Popularity on GitHub/Google Play
    - Actively Maintained
    - Likelihood of finding new bugs (# existing bug reports)

29 students
Setting 1: Coders-vs-Coders

Pair sharing:

\( i \): original shared issue

\( j \): derived issue.

1. Bug Report for App A
   - Developer A
   - on xyz • edited

2. Members of xyz
   - I found similar bugs as A

   - Write
   - Preview
   - I found a bug that is similar to App A for App B
   - Attach files by dragging & dropping, selecting or pasting them.
Motivating Example

$i$: original shared issue

Prevalent problems for many apps

5 pair sharings

View are not immediately update, will only update the view after restarting the app
Setting 2: Coder-vs-Manual-Issues

- Coder A needs to manually perform the steps below:
  1. Select the relevant issue
  2. Reproduce the steps in
  3. Check if the same bug described in applies for the app by Coder A
Effectiveness of these two settings

- Collaborative bug finding finds 17 new bugs
- Automatic testing tool like Monkey only finds 2 bugs
  - Crashes found are hard to reproduce
Types of Bugs found

- Our approach finds:
  - Prevalent problems (outdated view, certain names, certain mode, change of language)
  - Specific problems (29%)
  - Types of bugs are mostly non-crash related
  - Our approach complements existing automated testing approaches (mostly focus on finding crashes).
What do students think about collaborative bug finding?

“Because many functions in the app in the same category are similar even totally same ... and others' report will also inspire the mind to find bugs which I never considered.”

“Collaborative bug finding takes more time to review different apps and search useful issues...but is more likely to find new bug”

Searching for issues could be time-consuming

➢ Need automation for collaborative bug finding!
Setting 3: Coder-vs-Auto-Issues (Bugine)

- GitHub issues pre-processing
- Given an $App_{query}$
  1. Extracts its UI components to get its app description file
  2. Use similarities between app description file for $App_{query}$ and app description files for apps in database to search for similar apps
  3. Rank issues based on quality
Finding the similarities between apps

Android App 1 Shares similar GUI components My Android App

Has some bug reports that mentioned the “shared” GUI components, recommend these bug reports to my app

Are these two apps similar?
UI elements is usually declared in XML files
Android UI is defined via the hierarchy of View and ViewGroup objects
Names of Android resource following certain conventions

From: https://abhiandroid.com/ui/xml
Naming information for extracting description files

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
<th>Extracted Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
<td>Resource Name</td>
<td>android:id=&quot;@+id/my_btn&quot;</td>
<td>my_btn</td>
</tr>
<tr>
<td>View Name</td>
<td>Name for the type of UI component</td>
<td>&lt;Button android:id=&quot;@+id/my_btn&quot; /&gt;</td>
<td>Button</td>
</tr>
<tr>
<td>XML File Names</td>
<td>Layout name</td>
<td>main_layout.xml</td>
<td>main_layout</td>
</tr>
</tbody>
</table>

For each XML file, convert each view and each resource in $App_{query}$ to the query of the form:

$XML$ file name $\land$ View Name $\land$ Resource name
## Ranking GitHub Issues

<table>
<thead>
<tr>
<th>Factors</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue length</td>
<td>Word count of issue body (int)</td>
<td>Longer issue is better</td>
</tr>
<tr>
<td>Issue Status</td>
<td>Closed or opened (binary)</td>
<td>Issue is more important if it is:</td>
</tr>
<tr>
<td></td>
<td> Closed</td>
<td>✓ Closed</td>
</tr>
<tr>
<td>Ref Commit SHA</td>
<td>Commit SHA referenced by issue (binary)</td>
<td>✓ Fixed</td>
</tr>
<tr>
<td>Issue Reply Number</td>
<td>The number of replies that an issue received (int)</td>
<td>✓ has more replies (comments)</td>
</tr>
<tr>
<td>Hit_all</td>
<td>Find all search keywords in the corpus (binary)</td>
<td>Search for shared UI components:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Match all keywords (better)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓ Has some shared components</td>
</tr>
<tr>
<td>Hit_overlap</td>
<td>Overlap coefficient between search keywords and corpus (float)</td>
<td></td>
</tr>
<tr>
<td>Hit_Hot_Words</td>
<td>Word count of descriptive hot words like reproduce, defect (int)</td>
<td>Issues that meet the criteria for good bug reports is better</td>
</tr>
</tbody>
</table>
## Evaluation of Bugine

<table>
<thead>
<tr>
<th>App Name</th>
<th>Category</th>
<th>KLOC</th>
<th>#Downloads</th>
<th>Rating</th>
<th>Version No.</th>
<th>#GitHub Stars</th>
<th>#GitHub Issue (closed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera-Roll</td>
<td>Gallery</td>
<td>26.00</td>
<td>100,000+</td>
<td>4.2</td>
<td>1.0.6</td>
<td>420</td>
<td>227(133)</td>
</tr>
<tr>
<td>PocketHub</td>
<td>GitHub client</td>
<td>31.35</td>
<td>10,000+</td>
<td>3.3</td>
<td>0.5.1</td>
<td>9429</td>
<td>644(526)</td>
</tr>
<tr>
<td>Simple File Manager</td>
<td>Explorer</td>
<td>5.84</td>
<td>50,000+</td>
<td>4.5</td>
<td>6.3.4</td>
<td>378</td>
<td>189(130)</td>
</tr>
<tr>
<td>Zapp</td>
<td>Broadcast</td>
<td>8.41</td>
<td>N.A.</td>
<td>N.A.</td>
<td>3.2.0</td>
<td>60</td>
<td>151(137)</td>
</tr>
<tr>
<td>Simpletask</td>
<td>Reminder</td>
<td>24.80</td>
<td>10,000+</td>
<td>4.7</td>
<td>10.3.0</td>
<td>349</td>
<td>821(583)</td>
</tr>
</tbody>
</table>
Evaluate the ranking performance of Bugine

Use two metrics commonly used in prior recommendation systems:

1. \( \text{Prec@k} = \frac{\# \text{relevant documents in top } k}{k} \)

   - Retrieval precision over the top \( k \) documents in the ranked list

2. \( \text{MRR} = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{1}{\text{first}_q} \)

   - For each query \( q \), the MRR measures the position \( \text{first}_q \) of the first relevant document in the ranked list
Ranking performance of Bugine

- Prec@10: 0.1–0.7
  - Among the top 10 issues recommended by Bugine, there is ≥ 1 relevant issue
- MRR values: 0.34–0.75
  - Ranking for the first relevant document is btw 3rd (0.34) and 1st (0.75)
  - Bugine could recommend relevant issues for all evaluated apps
# Bugs found by Bugine

- Found 34 new bugs and 13 old bugs
  - In first two settings, 29 students find 17 new bugs in 20 apps
  - Bugine could discover more bugs despite being evaluated only on five apps.

<table>
<thead>
<tr>
<th>App Name</th>
<th>#Bugs Found (new, old)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera-Roll</td>
<td>(11, 0)</td>
</tr>
<tr>
<td>PocketHub</td>
<td>(12, 2)</td>
</tr>
<tr>
<td>Simple File Manager</td>
<td>(6, 2)</td>
</tr>
<tr>
<td>Zapp</td>
<td>(2, 7)</td>
</tr>
<tr>
<td>Simpletask</td>
<td>(3, 2)</td>
</tr>
</tbody>
</table>
Feedbacks from the ICSE 2020 Reviewers

“This is one of those simple but great ideas that make a lot of sense…”

“I thought the idea of collaborative testing was intriguing and thought provoking... I really liked the effort by the authors to think creatively about this and present an out of the box idea for test generation”

“The idea of collaborative bug finding is refreshing and interesting”

Interesting Research Questions:

- Can the authors provide any insights on how to automate collaborative bug finding?
  - Can we extract fix patterns from these common issues?
Could we make sure of the fixes of similar bugs for Automated Program Repair?

- Could provide high-level fix suggestion
  - Use Observable vs. Background Service for handling asynchronous events for fixing the outdated view problem
Future Work: Collaborative Testing and Repair

- How about recommendation systems for bug fix commits?

![Diagram]

Program

Similar bugs &
Similar code

My Program

Bug fix
commits

Has bug fix commits, recommend these commits to another Program
Summary

- 3 settings:
  - Coders-vs-Coders
  - Coder-vs-Manual-Issues
  - Coder-vs-Auto-Issues (Bugine)

- Reusing bug reports from different apps
  - Exploit the redundancies of bug reports in open-source repositories
  - Use similarities between apps

- New way of testing Android Apps
  - Instead of test input generation, we re-formulate the test generation problem as bug report recommendation problem
    - Bugine recommends relevant bug reports automatically
  - Developer could read reports written in Natural Language
    - Do not need to learn a new testing framework nor API

- Found 51 new bugs, 5 confirmed, 7 fixed