Project Report

Study of Functional Bugs in Android Apps

Manikanta Vattem

M15893841

1. Problem: Android apps are ubiquitous and serve many aspects of our daily lives. Ensuring their functional correctness is crucial for their success. To date, still lack a general and in-depth understanding of functional bugs, which hinders the development of practices and techniques to tackle functional bugs. To fill this gap, conducted a systematic study on functional bugs from popular open-source and representative Android apps to investigate the root causes, bug symptoms, test oracles, and the capabilities and limitations of existing testing techniques.

2. Introduction:

Android applications operate through graphical user interfaces (GUIs), providing interactive functionalities that are pervasive in various aspects of daily life. Recent reports highlight the significance of user experience, with only 16% of users willing to retry a malfunctioning app more than twice. Malfunctioning apps can significantly impact users in real-life scenarios, underscoring the crucial importance of ensuring an app's functional correctness for success. Despite this, effectively identifying non-crashing functional failures, commonly referred to as functional bugs, in Android apps remains a formidable challenge. A key obstacle lies in the lack of a comprehensive understanding of functional bugs, including their root causes, the impact on apps, and the mechanisms through which these bugs are induced.

3. Collecting the most popular apps:

App Name	App Feature	First Release	#Installations	#Stars	#Bugs Selected
Simplenote	Notebook	Nov. 2013	1M~5M	1.5K	35
AnkiDroid	Flashcard Learning	Jun. 2009	10M~50M	5.4K	82
Amaze	File Manager	Nov. 2014	1M~5M	4.0K	30
K-9 Mail	Email Client	Jan. 2014	5M~10M	7.1K	36
NewPipe	Video Player	Sep. 2015	5M~10M	20.7K	65
AntennaPod	Podcast Manager	Feb. 2014	500k~1M	4.4K	41
WordPress	Blog Manager	Dec. 2015	10M~50M	2.7K	67
Firefox	Web Browser	Jun. 2017	10M~50M	2.1K	43

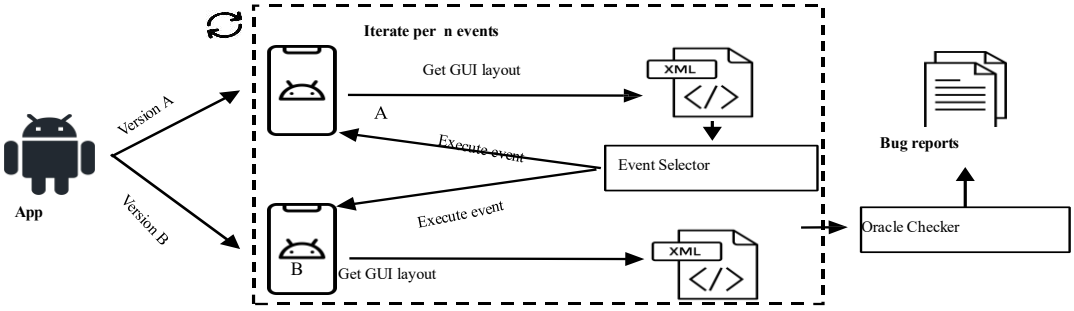
4. Design Approach:

Implemented the preceding idea as an automated differential testing tool named RegDroid to find functional bugs. Figure depicts the workflow of RegDroid. At the high-level, RegDroid generates random GUI tests (in the form of UI event sequences), runs these tests separately on the two app versions (e.g., version A and B), and checks whether the GUI pages of version A and B (along the GUI tests) contain the similar UI widgets. Specifically, RegDroid runs two identical Android devices in parallel to improve testing efficiency. It includes two key modules: (a) event selector and (b) oracle checker, which explain as follows.

Event selector. This module is responsible for generating random GUI tests. It randomly selects one executable GUI widget w from the current GUI page ℓA of app version A and generates the corresponding event e (e.g., click, edit, scroll) according to w 's widget property (e.g., clickable, editable, scrollable). Later, this event e will be executed in version A and B, respectively. Note that the oracle checker module will be called before the event execution. This module will be iteratively called until generating n events, or some inconsistency is found.

Oracle checker. This module is responsible for oracle checking. It is called before the actual execution of the selected event e. Specifically, since the UI widget w exists on the current GUI page ℓA of app version A, this module checks whether w also exists on the corresponding GUI page ℓB of app version B. Specifically, we check whether the GUI page ℓB contains the same UI widget whose resource id and class name are identical to those of w. If w does not exist on ℓB, REgDRoID finds the inconsistency and reports a likely bug. Otherwise, e will be executed on version A and B, respectively. After the execution, the event selector is called.

REgDRoID is implemented in Python and uses uiautomator2 to parse UI widgets from the UI pages and send UI events.



Overview of RegDriod.

5. Evaluation Results:

Applied REgDRoID on five popular, long- maintained open-source apps (see below Table). Among these five apps, three apps, i.e., AnkiDroid, Amaze, AntennaPod, are convenient to setup on the machine for testing.

REgDRoID is configured to test any two continuous release versions of an app by generating 50 random GUI tests (each test contains 100 events), which took about 12 hours. Afterwards, inspected all the bugs reported by REgDRoID. If a bug is a true positive, tried to reproduce it on the latest app version to verify whether this is a new (unknown) bug or a fixed one.

6. Results. REgDRoID successfully found 14 unique functional bugs. Among these 14 bugs, 10 are new bugs. Table shows these 14 functional bugs, including the app name, the number of stars on GitHub, the bug id, the bug state, whether it is a new bug affecting the latest released version (with the issue id on GitHub), and the bug symptom. The found bugs cover five different symptoms (e.g., missing UI elements, incorrect in- teraction logic, functionality does not take effect).

App Name	#Stars	ID	Bug State	New Bug?	Bug Symptom
AnkiDroid	5.4K	1	Fixed	Yes (#12053)	Incorrect interaction logic
		2	Fixed	Yes (#11220)	UI element does not react
		3	Fixed	No	Incorrect interaction logic
		4	Fixed	Yes (#11363)	UI element does not react
Amaze	4.0K	5	Fixed	Yes (#3378)	Missing UI elements
		6	Fixed	Yes (#3394)	Redundant UI elements
		7	Fixed	No	Missing UI elements
		8	Fixed	No	Redundant UI elements
AntennaPod	4.4K	9	Fixed	Yes (#5977)	Missing UI elements
		10	Fixed	Yes (#5863)	UI Element does not react
Markor	2.4K	11	Fixed	Yes (#1800)	Functionality does not take effect
Omni-Notes	2.5K	12	Fixed	Yes (#865)	UI Element does not react
		13	Fixed	Yes (#867)	Functionality does not take effect
		14	Fixed	No	Missing UI elements

New Findings:

In the previous work they have tested with the common apps, but they did not include the gaming apps. So has a contribution for the project. I have tested with the gaming apps to find the functional bugs in the gaming apps. But this tool was not able to find the single bug in the gaming apps. So, tools have a problem with finding the functional bugs in the gaming apps.

REGDROID reported bugs. Among these bugs, 36 % are true positives and 64% are false positives. Many false positives are duplicated. Noted that these false positives are caused by two major reasons: (1) some app feature of the current version was updated in the newer version; and (2) some bug in the current version was fixed in the newer version. In practice, REGDROID highlights the inconsistencies on the UI pages to ease bug inspection. REGDROID already shows its promise in finding hard-to-detect functional bugs, which complements existing tools. In fact, its positive rate (64%) is comparable to existing sophisticated, state-of-the-art functional testing tools GENIE and ODIN, which respectively have 59% and 60% false positive rates. Moreover, it is feasible to avoid many false positives by choosing two "closer" app versions with fewer feature (UI) changes. It can be achieved by analyzing the app code to identify which code commits changed UIs.

The benefits of this project, compared to other tools RegDroid has 64% positive rates compared to Genie and Odin.

7. Limitation of this project:

- It is not common for all the other apps.
- Our findings may not be general to all the apps. To mitigate this threat, these apps are selected carefully to ensure that their representativeness.
- These apps are popular, actively maintained and have different features.
- Not supported to gaming apps.

8. Data Availability:

All the artifacts (including the bug dataset and the source code of REgDRoID) publicly available at https://github.com/manikantavattem/functional_bugs.

****Note**: As this is the study of the functional bugs there are few works to be considered and there are no screenshots available for the results. This result contains only the theoretical study but not much practical evaluation on this project.