Bugine: a bug report recommendation system for Android apps

Ziqiang Li
Southern University of Science and Technology
11510352@mail.sustech.edu.cn

ABSTRACT
Many automated test generation tools were proposed for finding bugs in Android apps. However, a recent study revealed that developers prefer reading automated test generation cases written in natural language. We present Bugine, a new bug recommendation system that automatically selects relevant bug reports in the first page. Our results show that Bugine could find 34 new bugs in five evaluated apps.

CCS CONCEPTS
• Software and its engineering → Software libraries and repositories, Software maintenance tools.

KEYWORDS
bug report, recommendation system, Android apps

ACM Reference Format:

1 INTRODUCTION
Bug finding is a creative and inspiring activity. Many automated test generation [8] and repair techniques have been proposed to ensure the reliability of Android apps [1, 3, 6]. However, reading and reproducing the automatically generated test cases could be time-consuming. A study showed that developers prefer reading automatically generated test cases written in natural language [4]. This study also revealed that developers prefer manual testing compared to automated testing due to the learning curve of automated tools or lack of specific knowledge. Moreover, automated testing techniques for Android apps mostly focus on finding crashes [5], but neglect other non-crash related bugs (e.g., UI bugs). Meanwhile, many manually crafted bug reports (in natural language) are available in open-source repositories like GitHub.

Inspired by developers’ requirements and the redundancy of bug reports, we propose Bugine [7]. Given an Android app A, Bugine will automatically select relevant bug reports for A. Each relevant bug report may include reproduction steps and bug fixes of similar bugs that could be useful for testing and debugging.

2 METHODOLOGY
Figure 1 shows the workflow of Bugine which includes: (S1) building a database of GitHub issues, (S2) finding common UI components between AppQuery and AppDatabase, (S3) constructing query based on common components from (S2) to search for relevant issues of AppDatabase, (S4) ranking the results based on the quality. Building a database of GitHub issues. Our crawler selects Android apps based on: (1) the users’ rating and downloads in the App store, (2) the number of discussion and comments by developers, (3) the number of the star and issue of GitHub repository, and (4) the category of GitHub repository. Then, we collected all the issues and the meta-data of the selected apps (i.e., title, author, number of user comments, labels, issue state, body, commit SHA, etc.). We also downloaded the source code from the master branch of each app for subsequent steps. Our database has 23980 issues from 34 different applications that are selected from 10 different functionalities (e.g., cloud client, Git hub client, file explorer, web browser, etc.).

Data Pre-Processing. Our data consists of GitHub issues and the source code of the corresponding apps. For the GitHub issues, Bugine pre-processes them with commonly used NLP techniques. We also use Humps¹ to unify the naming conventions of program variables, including snake case, camel case, into variable tuples and split compound (composite words). For the source code, specifically XML files, we also extract structural information of UI components. Extracting app description files. The UI of Android apps is typically declared in XML resource files that define the structural layout of the UI components (e.g., view classes and subclasses). Each defined resource will be mapped to a resource ID. To generate app

¹https://github.com/nficano/humps
For each query Mean Reciprocal Rank (MRR) been used before. Overall, the Prec@10 results range from 0.1 to 0.7, independently evaluate the top 100 ranked issues of each app that have not been used before. Overall, the Prec@10 results range from 0.1 to 0.7, which means that among the top 10 issues recommended by Bugine, there is at least one relevant issue. Meanwhile, the MRR values for Bugine range from 0.34 to 0.75, which means that the ranking for the first relevant document ranges between 3rd (0.34) and 1st (0.75). This indicates that Bugine could recommend relevant issues for the evaluated apps.

**RQ2: Number of bugs that Bugine finds.** With two raters, we evaluate RQ2 by manually replicating the top-ranked issues to check if they are reproducible in AppQuery and we consider that Bugine discovers a bug if such issue ranks in the top 100. The "# Bugs Found" column in Table 2 shows the number of issues found by Bugine. In total, we found 34 new bugs and 13 old bugs in all the five evaluated apps. Overall, our results show that Bugine could recommend relevant issues, which leads to the discovery of new bugs. All bugs found by Bugine are archived at https://bugine.github.io/.

**4 CONCLUSION**

We introduce a new approach that recommends relevant GitHub issues for an app under test. Given an app under test, Bugine searches for relevant GitHub issues based on the similarities of UI components shared with other apps in our database and further ranks them based on their quality. Our evaluation shows that it helps to discover 34 new bugs in the five evaluated apps.

**Acknowledgments.** This work is partially supported by the National Natural Science Foundation of China (Grant No. 61902170) and Natural Science Foundation of Guangdong Province (Grant No. 2020A1515011494).

**REFERENCES**


