CoCoTest: Collaborative Crowdsourced Testing for Android Applications

Haoyu Li
State Key Laboratory
for Novel Software Technology
Nanjing University, China
Mooctest Inc., Nanjing, China

Chunrong Fang*
State Key Laboratory
for Novel Software Technology
Nanjing University, China
Mooctest Inc., Nanjing, China

Zhibin Wei
Zhenyu Chen
State Key Laboratory
for Novel Software Technology
Nanjing University, China
Mooctest Inc., Nanjing, China

ABSTRACT

Testing Android applications is becoming more and more challenging due to the notorious fragmentation issues and the complexity of usage scenarios in different environments. Crowdsourced testing has grown as a trend, especially in mobile application testing. However, due to the lack of professionalism and communication, the crowd workers tend to submit low-quality and duplicate bug reports, leading to a waste of test resources on inspecting and aggregating such reports. To solve these problems, we developed a platform, CoCoTest, embracing the idea of collective intelligence. With the help of CoCoTest Android SDK, workers can efficiently capture a screenshot, write a short description and create a bug report. A series of bug reports are aggregated online and then recommended to the other workers in real time. The crowdsourced workers can (1) help review, verify and enrich each others' bug reports; (2) escape duplicate bug reports; (3) be guided to conduct more professional testing with the help of collective intelligence. CoCoTest can improve the quality of the final report and reduce test costs. The demo video can be found at https://youtu.be/PuVuPbNP4tY.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Crowdsourced testing, mobile testing, collective intelligence

ACM Reference Format:
Haoyu Li, Chunrong Fang, Zhibin Wei, and Zhenyu Chen. 2019. CoCoTest: Collaborative Crowdsourced Testing for Android Applications. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3293882.3339000

*Corresponding author: fangchunrong@nju.edu.cn

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3339000

1 INTRODUCTION

Android is one of the most popular smartphone systems, and there are a large number of Android application (app) developers. However, testing on Android apps is becoming more and more difficult, especially for individual developers or small teams. There are various reasons, such as the fragmentation issues of the Android platform [5] [9], the complexity of the user environment in different usage scenarios and fast iteration of the Android platform. These problems make compatibility testing, reliability testing, and security testing for Android apps more difficult. Therefore, many developers turn to crowdsourced testing platforms to seek quality assurance on Android apps.

Crowdsourcing represents an act of a company or institution taking a function once performed by employees and outsourcing it to an undefined and generally large network of people in the form of an open call [1] [2] [6]. The crowdsourced testing is that the requestors outsource the test tasks to different workers from different places, who fulfill the tasks and submit reports [3]. It aims to introduce real users and cover a diversity of different mobile models and test environments. For the workers, they can obtain some rewards regarding their efforts, which are usually referred to the number and quality of bug reports they submitted. For the requestors, they can get a final report by aggregating all the reports collected by the crowdsourced testing platform. Crowdsourced testing is a good solution to the problems caused by the fragmentation and fast iteration of the Android platform and the complexity of the user usage scenarios, and it is a win-win situation for the workers and requestors.

However, there are two inherent shortcomings of this solution. 1) It is hard to guarantee that all the workers are professional testers because the workers are from an open call and most crowdsourced testing tasks are bided online. 2) All the workers cannot communicate with each other conveniently because of geographical and temporal isolation. These shortcomings will cause two problems. 1) The bug reports are submitted directly and lack verification and review, so the quality of the reports cannot be guaranteed. 2) There are many duplicates of the bug reports. These two problems will significantly increase the difficulty of completing the final report, and sometimes manual review and sorting is required, which is a waste of testing resources.

Therefore, we developed the CoCoTest platform to solve the problems mentioned above. We provide a bug report recommendation function, which will recommend bug reports that have been filtered and sorted to the workers in real time to guide the workers

[13]. When a worker finds a new bug and submits a bug report with bug attributions, short text descriptions, screenshots and background logs on their own devices, this bug report will be added to the system bug repository according to some rules, and this bug report can be recommended to another worker in real time. The other workers can verify and enhance the recommended report. Based on the labeled/enhanced report from the workers, we will further optimize the recommended bug repository. In addition, the SDK will automatically collect the test operation and application status. This status will help us to cluster the reports.

In this paper, we mainly make the following contributions.

- We firstly propose a cooperation platform, CoCoTest, for Android application testing to exploit the concept of collective intelligence.
- Through a real-time bug recommendation and verification mechanism, CoCoTest can help solve serval serious issues in crowdsourced testing, such as bug report duplication, bug report inspection, and worker isolation.
- CoCoTest can help researchers to conduct more comprehensive studies on real-time bug report aggregation, bug report recommendation, and testing guidance.

2 THE COCOTEST TOOL

CoCoTest is a cooperation platform for crowdsourced testing of Android application. The architecture is shown in Figure 1 and consists of three parts.

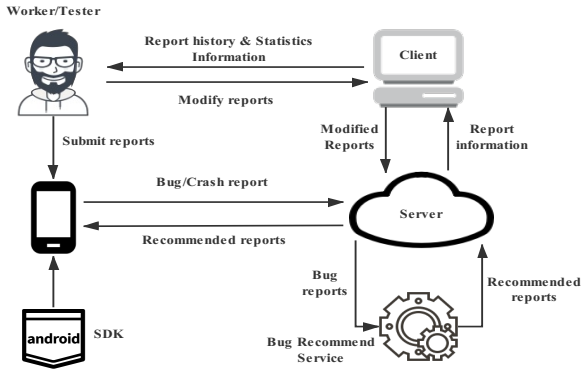


Figure 1: The architecture of CoCoTest.

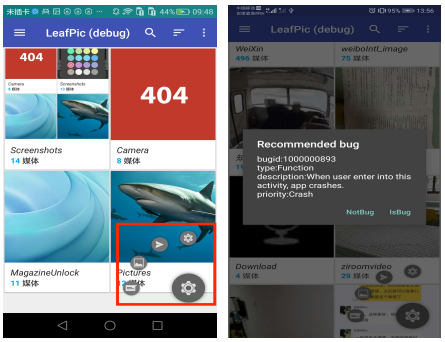
The first part is the CoCoTest SDK. Developers can import the SDK into their own application, and the configurations required is shown in Figure 2. During the test process, an auxiliary button will appear, through which the workers can submit the bug report and obtain our recommended bug reports. The bug report contains the description, type and severity of the bug and screenshots and background log. Our SDK will also automatically submit a crash report when the application crashes. The second part is a browser-based client application that uses Angular2 framework. The main function is to provide front-end interaction and operation for the requestors to manage workers, applications to be tested and the bug reports. The third part is a web application backend using Java, which provides services encapsulated by the RESTful API for front-end to manage workers, applications and bug reports. In addition, it provides the SDK with the services to submit bug reports, crash reports and to obtain the recommended bug reports.

Figure 2: The guide to use the SDK.

2.1 Using CoCoTest

When the requestors log into the CoCoTest client application, they need to add an app in the system. After filling in some necessary information of the app, they will get an AppKey, which is the unique identifier for the application in our system and they will see a brief guide about how to import the SDK into their apps, as shown in Figure 2. After importing the SDK into the application and adding the code into the activity to be tested, the requestors can publish test tasks to the crowds.

When the workers test the application, they can see an auxiliary button on the activities that need testing, as shown in Figure 3(a).



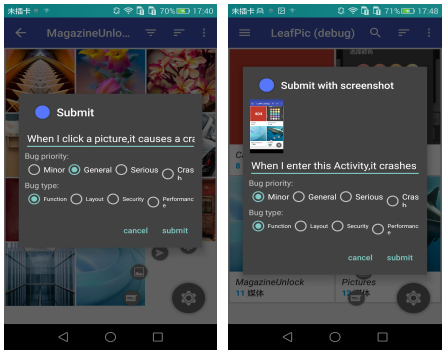
(a) Auxiliary Button (b) Recommended Bug

Figure 3: Auxiliary Button and Recommended Bug

The button has three main functions. 1) Log in. The workers can log in using the account registered in the browser client. 2) View the recommended bug report of the current activity as shown in Figure 3(b). The bug report listed is the result of collating and filtering all the bug reports of this activity in the backend. Since the aggregation is real-time, the recommendation report pushed to each worker may be different. At the same time, the workers can verify and modify the recommended bug report and then submit a new one. The new one will be linked to the old one and will be added to our bug repository according to the rules. 3) Submit a

bug report. The workers can submit a bug report with or without a screenshot based on the test results as shown in Figure 4. The screenshot can be automatically intercepted by the SDK or by the worker manually. In addition, our SDK will automatically monitor the status of the app under test. When the application has a bug or an exception that the developers have not handled, the SDK will automatically collect the crash information of the app and various states of the mobile and then submit them to the backend server.

At the same time, the workers can log in to the web client to review and modify the bug reports they have submitted. The requestors can also modify the resolution status of the bug, including new, handling, resolved if confirmed.



(a) Submit Bug (b) Submit Bug with pic

Figure 4: Submit bug report

2.2 Service Implementation

This section will briefly introduce the implementation details of the two core functions of our platform, report submission and report recommendation.

Unlike other kinds of tests, Android application test reports are naturally classified according to activity. So the reports mentioned later are the reports in the current activity. In a bug report, the most important parts of the bug report are the bug description and the application status. The application status will tell us the bug occurred on which activity and the bug description will help us to identify duplicated reports in the same activity. In addition, in order to make better calculations, each report has two parameters, reliability and edit count. Reliability indicates whether this bug is a real bug. The more reliability, the more likely this report is a real bug. Edit count indicates that the verified times of this bug. Edit count will increase by one each time this report is verified or marked as duplicated.

Report submission: The flow of report submission mode is shown in Figure 5. When a worker submits a new report, we use an open source Chinese word segmentation toolkit Pkuseg[1] and Bag of Words (BOW) to vectorize the bug description and then use the classic formula of cosine correlation to calculate the similarity of two bug reports. The two reports will be marked as duplicated when the similarity exceeds the threshold. 1) When the new report is not

[1]https://github.com/lancopku/PKUSeg-python

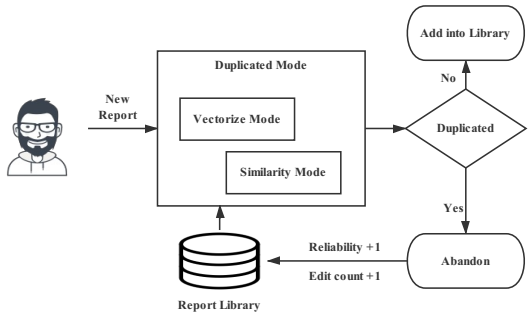


Figure 5: The flow of bug report submission.

duplicated with any reports in the library, the new report will be added to the report library. 2) When the new report is been marked as duplicated, the report will be abandoned and the old report's reliability and edit count will increase by one Because submitting duplicated report can be considered as a verification operation.

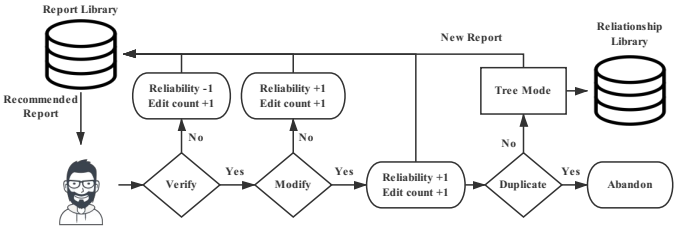


Figure 6: The flow of report recommendation.

Report recommendation: The flow of report recommendation mode is shown in Figure 6. When a worker tries to get recommended report, the recommendation service will choose a report. As mentioned above, each report has two params, reliability and edit count. We think the less the report is verified, the smaller the edit count is, the greater the recommended value. First, all the reports will be sorted by their edit count. And then the recommendation service will select the report which has not verified by the same worker and has the least edit count. Select the report that the worker has not verified is to avoid malicious verification.

Then the worker can verify this report. 1) When the verification result is yes, the report's reliability and edit count will increase by one. In this case, the worker can modify the bug report. The logic for submitting a modified report is the same with the logic for submitting a new report. The only differences are the modified report and the old report will form a tree structure and the threshold is lower. 2) When the verification result is no, the report's reliability will decrease by one and edit count will increase by one.

This is an iterative process where all the workers are involved, optimizing test reports and expanding the bug repository. Through our platform, the workers can overcome the geographical and temporal isolation and communicate with each other conveniently and in real time, which can help exploit collective intelligence [11]. Under the process mentioned above, the reports in the library will minimize the duplication. At the same time, we will have a tree-like

enhancement relationship between the reports. Using this relationship, it will not only significantly reduce the duplication of test reports in the crowdsourced testing process but also significantly reduce the workload of aggregating all the test reports with high quality.

2.3 Applications

There are many interesting and challenging studies based on our platform. These studies can be available because our platform 1) provides real-time bug report recommendation function, and 2) establishes a mechanism for crowdsourced workers to communicate with each other.

Report Recommendation: As mentioned above, in the crowdsourced testing, it's hard to guarantee that the workers are professional. Consequently, many poor reports could emerge and consume many resources to get rid off. So it is necessary to recommend bug reports to the workers and guide them to test so that the workers can mutually verify and enhance the bug reports.

Besides the strategies mentioned above, there are other strategy possibilities. The bug reports are usually comprehensive but with multi-resources, such as short text, screenshots, and logs. Different situations should have different strategies. CoCoTest supports activity-based report recommendation function and provides interfaces for newly proposed recommendation strategies according to different situations.

Guidance: In existing crowdsourced testing platforms, the workers are independent and cannot communicate with each other in real time, which will produce a lot of duplicated reports and waste many test resources. Besides, without effective guides, most unprofessional workers may only present shallow testing and miss the opportunity for finding deep bugs.

So we have proposed a mechanism to solve this problem. Through our platform, the crowdsourced workers can be guided to the activities or functions only tested by few workers, or be guided to perform tasks where they can perform better based on the history of their previous bug reports. It can use the strengths of different workers more effectively and improve the efficiency of the overall test and save costs.

However, how to make effective guidance remains changeling, such as identifying testing hotspots, graphical guidance, and global testing state maintenance.

3 RELATED WORK

Crowdsourced testing is becoming popular in both software engineering research community and industry. However, even though cooperation has been investigated in other crowdsourcing areas [12][7], cooperation is difficult to set up in crowdsourced testing. Consequently, there are many duplicated and ambiguity test reports. Some studies concentrate on addressing problems such as test report inspection. Yang Feng et al. proposed a text-based technique by combining a diversity strategy and a risk strategy to prioritize test reports[3]. Further, they proposed a hybrid analysis technique by leveraging both textual information and image information [4] to reduce unnecessary inspection on false positive test reports, some researchers adopt a cluster-based classification approach to discriminate the true positives from a large number of test reports[10]. All these studies deal with the test reports after the workers fulfill their tasks. CoCoTest proposes a mechanism to aggregate the reports in real-time during the testing process and introduce the workers to inspect test reports. Some researchers are concerned about app crashes. K. Moran et al. introduced a automated approach to trigger crashes and generate an augmented crash report [8]. This study focus on reproducing the crash. CoCoTest can verify crashes by recommending the report to the workers.

There are also many crowdsourced testing platforms to test Android apps, such as Test IO and Testlio. Comparing with CoCoTest, these two crowdsourced testing platforms focus on cross-platform and cross-terminal solutions. And our platform focus on Android application and how to realize collective intelligence by recommending bug reports and guiding unprofessional workers to test.

4 CONCLUSION

We developed the CoCoTest platform to solve the problems on the existing crowdsourced testing such as bug report duplication, bug report inspection, and real-time cooperation. CoCoTest exploits the concept of collective intelligence to recommend bug report of the current activity to the workers and guide them to verify and enhance the bug report through real-time bug report aggregation. Besides, CoCoTest allows workers to ignore the geographical and temporal isolation, and cooperate on bug report inspection and enhancement. Moreover, CoCoTest provides opportunities for researchers to further improve crowdsourced testing.

ACKNOWLEDGMENTS

The work is partly supported by the National Key Research and Development Program of China (2018YFB1403400) and the National Natural Science Foundation of China (61690201, 61772014).

REFERENCES

[1] Daren C Brabham. 2008. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence* 14, 1 (2008), 75–90.
[2] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara. 2012. Towards an integrated crowdsourcing definition. *Journal of Information science* 38, 2 (2012), 189–200.
[3] Yang Feng, Zhenyu Chen, James A Jones, Chunrong Fang, and Baowen Xu. 2015. Test report prioritization to assist crowdsourced testing.. In *FSE*. 225–236.
[4] Yang Feng, James A Jones, Zhenyu Chen, and Chunrong Fang. 2016. Multi-objective test report prioritization using image understanding. In *ASE*. IEEE, 202–213.
[5] Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia. 2012. Understanding android fragmentation with topic analysis of vendor-specific bugs. In *WCRE*. IEEE, 83–92.
[6] Jeff Howe. 2006. *Crowdsourcing: A Definition.* https://goo.gl/JDX5EH.
[7] Ziwei Liu, Xiaoguang Niu, Xu Lin, Ting Huang, Yunlong Wu, and Hui Li. 2016. A task-centric cooperative sensing scheme for mobile crowdsourcing systems. *Sensors* 16, 5 (2016), 746.
[8] Kevin Moran, Mario Linares-Vasquez, Carlos Bernal-Cardenas, Christopher Vendome, and Denys Poshyvanyk. 2016. Automatically Discovering, Reporting and Reproducing Android Application Crashes. In *ICST*.
[9] Sameer Singh. 2012. *An Analysis of Android Fragmentation.* http://goo.gl/gNDHC.
[10] Junjie Wang, Qiang Cui, Qing Wang, and Song Wang. 2016. Towards effectively test report classification to assist crowdsourced testing. In *ESEM*. ACM.
[11] Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. 2010. Evidence for a collective intelligence factor in the performance of human groups. *science* 330, 6004 (2010), 686–688.
[12] Rui Zhang, Jinxue Zhang, Yanchao Zhang, and Chi Zhang. 2013. Secure crowdsourcing-based cooperative pectrum sensing. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2526–2534.
[13] Xin Zhang, Zhenyu Chen, Chunrong Fang, and Zicong Liu. 2016. Guiding the crowds for Android testing. In *ICSE*. ACM, 752–753.