Empir Software Eng (2019) 24:537–561
https://doi.org/10.1007/s10664-018-9618-5

EXPERIENCE REPORT

Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing

Ruizhi Gao[1] · **Yabin Wang**[2] · **Yang Feng**[2,3] · **Zhenyu Chen**[2] · **W. Eric Wong**[1]

Published online: 12 May 2018

Abstract The term *crowdsourcing* – a compound contraction of crowd and outsourcing – is a new paradigm for utilizing the power of *crowds* of people to facilitate large-scale tasks that are costly or time consuming with traditional methods. This paradigm offers mobile application companies the possibility to outsource their testing activities to crowdsourced testers (crowdtesters) who have various testing facilities and environments, as well as different levels of skills and expertise. With this so-called *Crowdsourced Mobile Application Testing* (CMAT), some of the well-recognized issues in testing mobile applications, such as multitude of mobile devices, fragmentation of device models, variety of OS versions, and omnifariousness of testing scenarios, could be mitigated. However, how effective is CMAT in practice? What are the challenges and issues presented by the process of applying CMAT? How can these issues and challenges be overcome and CMAT be improved? Although CMAT has attracted attention from both academia and industry, these questions have not been addressed or researched in depth based on a large-scale and real-life industrial study. Since June 2015, we have worked

Communicated by: Antonia Bertolino

✉ Zhenyu Chen
zychen@nju.edu.cn

✉ W. Eric Wong
ewong@utdallas.edu

Ruizhi Gao
gxr116020@utdallas.edu

Yabin Wang
wangyabin890512@gmail.com

Yang Feng
yang.feng@uci.edu

[1] Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

[3] Department of Informatics, University of California, Irvine, CA, USA

with Mooctest, Inc., a CMAT intermediary, on testing five real-life Android applications using their CMAT platform – Kikbug. Throughout the process, we have collected 1013 bug reports from 258 crowdtesters and found 247 bugs in total. This paper will present our industrial study thoroughly and give an insightful analysis to investigate the successes and challenges of applying CMAT.

Keywords Crowdsourcing · Crowdsourced mobile application testing · Android applications

1 Introduction

In an article for *Wired* magazine in 2006, Jeff Howe defined *crowdsourcing* as "an idea of outsourcing a *task* that is traditionally performed by an employee to a large group of people in the form of an open call" (2016). To be more specific, it is a process of completing tasks by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees or suppliers. Since 2006, by shaping the tasks of crowdsourcing into different forms, some of the most successful new companies on the market have been using this idea to make people's lives easier and better. Uber (2018) discovered that by connecting drivers directly to customers through a phone application, customers would pay less for rides, and drivers could find customers more quickly. The drivers have the ability to create their own schedule and use the car of their choice, not just a mandatory yellow cab. Passengers are able to use their mobile phones to quickly hail a private car instead of waiting in the street for a taxi to come around the corner. Most importantly, taxi services are crowdsourced to a wider range of drivers, not just licensed taxi drivers. In a similar manner, Amazon Mechanical Turk (2018) crowdsources generalized tasks in different domains; Adaptive Vehicle Make (2018) sponsored by DARPA crowdsources the design and manufacture of a new armored vehicle to crowds instead of particular companies; Waze (2018) crowdsources traffic jam reporting and navigation services; and CrowdMed (2018) crowdsources the services of medical care.

Software testing, which is well recognized as a time consuming and expensive process, can also be conducted using crowdsourcing. Currently, crowdsourced software testing is a new trend in the software engineering research community (Allahbakhsh et al. 2013; Chen and Luo 2014; Cheng et al. 2015; Dolstra et al. 2013; Goldman et al. 2011; Goldman 2011; Harman et al. 2014; Huang et al. 2013; Liu et al. 2012; Mantyla and Itkonen 2013; Mujumdar et al. 2011; Pastore et al. 2013; Yuen et al. 2011). In industry, intermediaries such as UTest (2018), UserTesting (2018), and MyCrowd (2018) crowdsource software testing and commercialize this service successfully. Of the various types of software, mobile application is believed to be one of the most appropriate software to be tested through crowdsourcing (Crowdsourcing.org 2013; Latoza and Van der Hoek 2016; Xue 2013).

At present, mobile devices are rapidly becoming the primary method of interaction for people worldwide. This phenomenon stimulates the generation of thousands of applications. However, various challenges are encountered when performing mobile application testing. The researchers who conducted the 2017–2018 Capgemini World Quality Report for Mobile Testing (Capgemini 2017) listed top 6 challenges to mobile testing (as shown in Fig. 1). It shows that 47% of the companies claim they do not have the right testing process/method, 46% of the them claim they do not have the right tool, and 40% claim they do not have the devices available for testing.

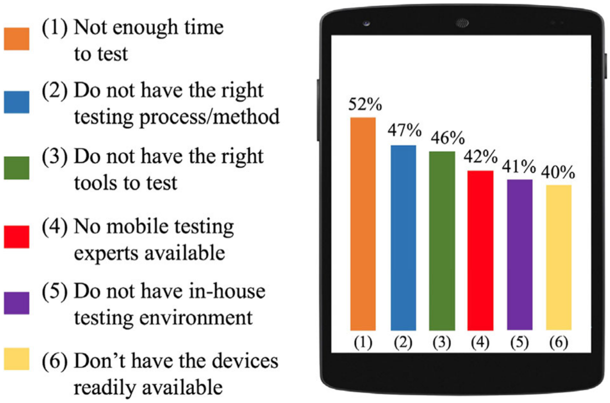


Fig. 1 Challenges to mobile testing

Purchasing one mobile phone is not difficult for a company, even for an independent application developer. However, it is far from adequate, especially for testing Android applications. First, there exists a multitude of Android devices. Over 500 million Android devices have been shipped since Android 1.0. This huge number of devices, ranging from handsets, to smart phones, to tabs, to wearable devices, provides a huge diversity of environments which your mobile application faces. Second, since Android is free and open-source, many mobile phone manufacturers install Android on their devices, which results in a serious device fragmentation problem. Based on the fragmentation report for Android by OpenSignal, Android Fragmentation Visualized (2015) in August 2015, there are 24,093 distinct models for Android devices of different brands, screen sizes, processors, memories, and so on. Third, Android itself evolves very rapidly. An application has to be compatible for different versions of Android. Many mobile phone manufacturers such as Samsung and Huawei have modified the interfaces of Android to make it their operating system of choice. Based on these reasons, to test an Android application well, developers have to purchase a large number of Android devices, which can be quite costly. In addition, the lack of good tools, processes, and testing environments (see Fig. 1) also makes mobile application testing challenging.

Crowdsourced mobile application testing (CMAT) could be a good candidate to tackle these problems. First, a CMAT platform is a useful tool to provide informative bug reports for companies uploading their applications. Also, the process of CMAT could be easily executed and cost-effective (a detailed explanation of the current CMAT workflow is given in Section 2.1). Most importantly, the problem of lacking devices could be greatly mitigated by CMAT. The particularity of CMAT is that testing is carried out by a large number of crowdtesters instead of hired consultants and professionals. A crowdtester can be any member of the general public with any mobile device and level of expertise. In this way, CMAT can better match users' demographics (e.g., location, age, gender, and operation habits), variety of devices, and diversity of testing environments (e.g., carrier and internet condition) compared to traditional in-house testing.

Despite the potential advantages, there is still comparatively little well-founded knowledge on CMAT, particularly with regard to how effective CMAT could be in practice, what the challenges of applying CMAT are, and how to tackle those challenges. These are the general topics we aim to investigate.

Since June 2015, we have participated in a project between Mooctest (a start-up CMAT intermediary) and five IT companies in China to apply CMAT on five real-life mobile applications using Kikbug (a powerful CMAT platform developed by Mooctest). The entire industrial study involved 258 crowdtesters with 1013 bug reports collected and 247 bugs[1] found. Based on our results, we observe that even though CMAT could be effective for detecting functional bugs for mobile applications, the application of CMAT is still in its infancy with a lot of room for improvement. In this paper, we present our study thoroughly and provide an in-depth analysis on important issues of CMAT inspired from this study. The first question regards the comparison between CMAT and in-house testing. We investigate the invalid bug reports issue in the second research question. In the third research question, we discuss whether the current compensation scheme of CMAT is reasonable. Lastly, duplicated bug reports issue is researched in question 4.

The remainder of this paper is organized as follows. Section 2 demonstrates the workflow of CMAT using Kikbug and provides a running example. Section 3 reports an overview of our industrial study. Section 4 presents the data of our study from different perspectives. Investigations of different research questions are given in Section 5. Section 6 presents the threats to validity. Other studies that are related to our technique are presented in Section 7. Our conclusions and direction of future work can be found in Section 8.

2 The Workflow of CMAT

We now present the workflow of CMAT. This workflow is used by most CMAT intermediaries in the market such as UTest, UserTesting, and Mooctest. In general, it involves (1) customers from the companies who provide their mobile applications for testing, (2) crowdtesters, and (3) a CMAT platform. The CMAT platform is provided by the CMAT intermediary and is responsible for all interactions between crowdtesters and customers. In this section, we will explain the workflow using Kikbug as the CMAT platform[2] (more information about Mooctest and Kikbug can be found in Section 3.1).

2.1 Current Workflow of CMAT

In general, there are five major phases in the workflow of CMAT as shown in Fig. 2.

- Phase 1: *Application Upload*

First, customers upload their applications to the application plaza (where the crowdtesters can select the task that they would like to work on) of the CMAT platform and specify the corresponding testing tasks and compensation information (e.g., an estimated payment for detecting one bug approved by the customers). Each task can be associated with one or multiple functions to be tested. The number of tasks and how to design each task is decided by the customers.

- Phase 2: *Task Selection and Environment Setup*

[1] In this paper, we use “bug” and “fault” interchangeably.

[2] The functionalities of different CMAT platforms may vary. However, the impact on the general CMAT workflow is insignificant.

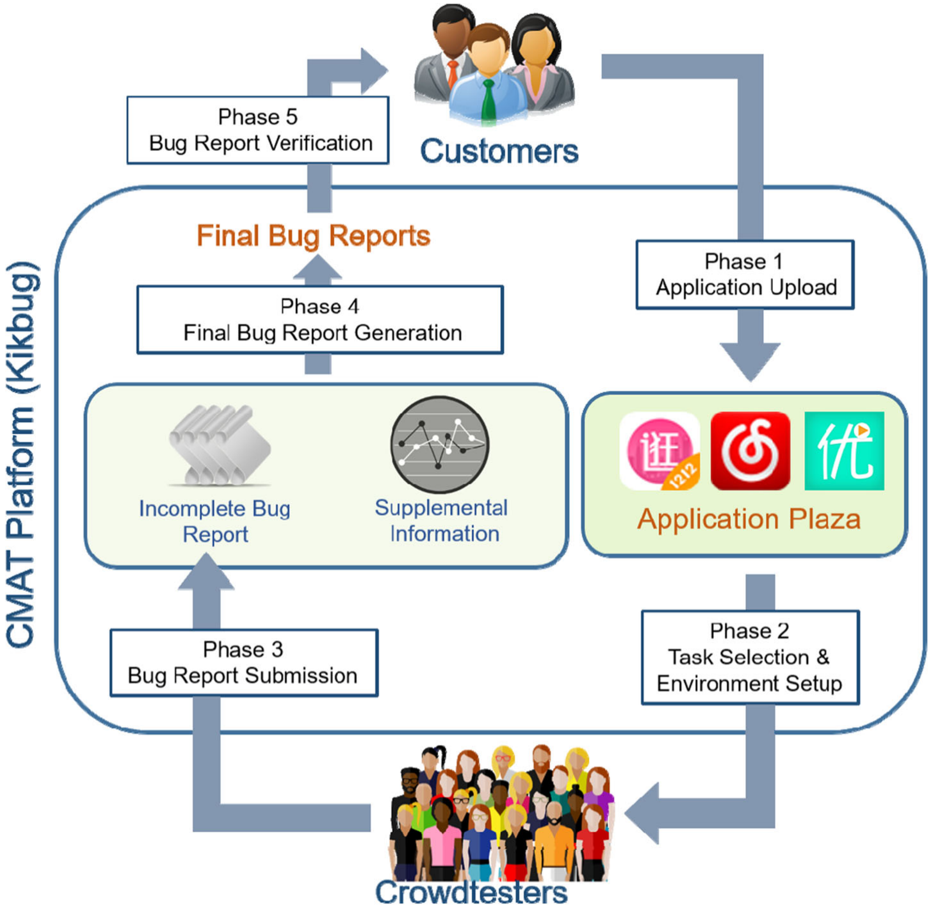


Fig. 2 The workflow of CMAT using Kikbug

Once applications with detailed testing tasks are activated, crowdtesters are allowed to choose any task(s) which they would like to work on. One observation we have made is that even though a crowdtester selects a specific task, it is not guaranteed that this crowdtester will follow the task's requirement strictly. It is highly possible that he or she will test additional functions belonging to other tasks and report the bugs accordingly. After task selection is completed, the crowdtesters will download the application from the platform. With Kikbug, in addition to the application itself, the crowdtesters are also required to download a driver for this application. Kikbug works with the driver to monitor the testing process (refer to Section 2.2 for more details).

- Phase 3: *Bug Report Submission*

Based on the selected task, crowdtesters start to test the application. During the testing, crowdtesters can view the bug reports submitted by others and submit their own bug reports to the platform (i.e., multiple reports can be submitted for one testing task). Different CMAT platforms may require different information for submitting a bug report. Generally, the report contains some natural-language descriptions about the testing process and the bugs detected (Dolstra et al. 2013; UTest 2018; Zogaj et al. 2014).

For Kikbug, the bug report submitted by a crowdtester needs the following input: (1) title of the report, (2) description of the bug(s) detected; (3) description of the testing process; and (4) screenshots taken while testing, if any. We call the test report submitted by a crowdtester an *incomplete bug report.*

- Phase 4: *Final Test Report Generation*

After each incomplete bug report is submitted, a CMAT platform collects supplemental information to generate the final bug report. The supplemental information collected by Kikbug includes (1) general information – submission time, task ID, report ID and crowdtester ID; (2) device information – brand, model, OS version, and screen size; and (3) operation path. The contents of the incomplete bug test report, together with (1) and (2), form the descriptive information of the final bug report. The operation path is a chronological, sorted list of all activities triggered during testing. An activity in Android is an application component that provides a screen with which users can interact. This operation path is mainly used by customers for debugging purposes, and it can also be used to determine whether a crowdtester indeed conducted the testing as he or she describes. An example of a final bug report and its operation path will be given in Section 2.2. After a final bug report is generated, it is saved into the CMAT platform.

- Phase 5: *Bug Report Verification*

Customers will verify all the final bug reports for their applications and decide how to compensate each crowdtester who submitted bug reports. The level of compensation which the crowdtester receives should be commensurate with their bug reports' quality. However, how to set up an appropriate incentive mechanism and compensation scheme is still an open question for all crowdsourced-based systems (Latoza and Van der Hoek 2016; Mantyla and Itkonen 2013; Zogaj et al. 2014). See Section 5.3 for more detailed discussion.

2.2 A Running Example

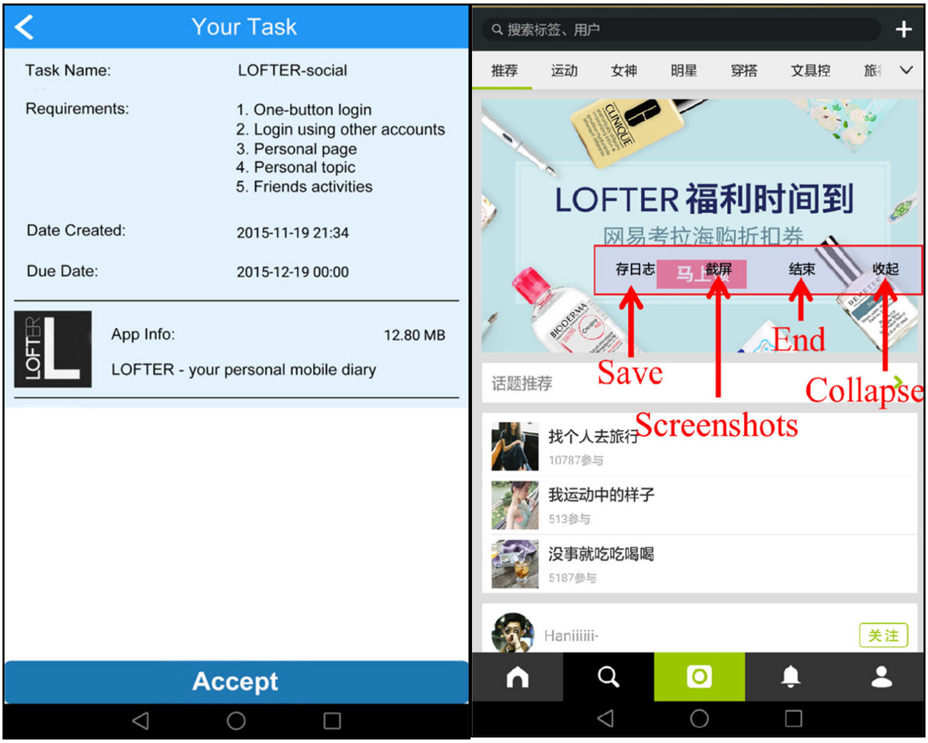
Let us use a running example to demonstrate the workflow of CMAT with Kikbug as the platform.[3]

Suppose that, in Phase 1, several applications have been uploaded to the application plaza with corresponding testing tasks and compensation information.

In Phase 2, a crowdtester is logged in and selected to test the task "LOFTER-social" of the application LOFTER. The screenshot in Fig. 3a specifies the general information about this application and detailed requirements of this task. The crowdtester then downloads the application and the corresponding driver from Kikbug. The driver will place a translucent pop-up window on top of the application as shown in Fig. 3b. There are four options in this window: (1) Save – to submit an incomplete test report whenever the crowdtester feels necessary; (2) Screenshots – to take a screenshot of the current screen which will be attached automatically to the next report submitted; (3) End – to end testing; and (4) Collapse – to minimize the pop-up window. The crowdtester will use these functions while testing the application.

In Phase 3, the crowdtester tests the application based on the task and submits one incomplete bug report.

[3] You may visit http://mooctest.net/wiki for a test trial with more detailed instructions.



Your Task

Task Name: LOFTER-social

Requirements:
1. One-button login
2. Login using other accounts
3. Personal page
4. Personal topic
5. Friends activities

Date Created: 2015-11-19 21:34

Due Date: 2015-12-19 00:00

App Info: 12.80 MB

LOFTER - your personal mobile diary

Accept

(a) task information of LOFTER-social (b) translucent pop-up window

Fig. 3 Screenshots of Kikbug

As described in Phase 4, once the incomplete bug report is submitted, Kikbug collects the supplemental information and generates the final bug report. The descriptive information of this final bug report is shown in Table 1. In this bug report, the crowdtester claims that one bug has been found when he tries to use his Weibo account to log into LOFTER.

In this example, the crowdtester has triggered seven activities. Kikbug sorts these activities in chronological order and generates an operation path. The operation path provides useful information when the customers try to locate the bug and validate that the crowdtester indeed conducts the testing as he describes and is not cheating in the report.

Table 1 Descriptive information of the final bug report

Report ID	1358	Report name	My Report
Brand	Huawei	Bug Description	I tried to use Weibo account to log in but failed. Then I logged in with my WeChat account.
Model	Y511-T00		
OS Version	4.2.2		
Submission Time	2015–11-19 21:36		
Task ID	133		
Tester ID	164	Test Description	Functions 1,3,4, and 5 are OK
Time Used	2 min		
Screen Size	4.1 in.	Screenshots	None

In Phase 5, this report is verified by the customer. By reading the description and conducting a series of checks, the customers decide to approve this bug. The crowdtester receives corresponding compensation from the customer for this bug.

3 Industrial Study Overview

We now give an overview of our industrial study. Section 3.1 gives the background information of Mooctest and Kikbug. Section 3.2 gives an introduction of the Android applications used in our study.

3.1 Mooctest and Kikbug

Mooctest, Inc. is a start-up company in Nanjing, China, founded in 2012. It provides crowdsourced software testing services with a focus on mobile applications. Mooctest has been developing Kikbug since 2014. Currently, Kikbug only supports CMAT for Android applications. Corresponding functions for IOS applications are under implementation. You can visit http://www.mooctest.net for more information about Mooctest, Inc.

After the first version of Kikbug was released in January 2015, Mooctest has provided the CMAT service to more than 20 IT companies in China thus maintaining a customer base consisting of multiple small and mid-sized, as well as a few large-sized, companies (including the most influential three − Tencent, Baidu, and Alibaba). Based on the feedback from these companies, they keep improving their services. By January 2016, Mooctest has recruited a crowd consisting of over 1000 crowdtesters. Most of these crowdtesters are 18 to 30 years old. Based on the information from Mooctest, about 90% of their crowdtesters hold a bachelor's degree in Computer Science or Software Engineering and about 30% of them hold a Master's degree. In addition, 75% of all crowdtesters have taken a software testing class.

3.2 Experiments Setup and Android Applications

Five Android applications from different companies are used in our study. The name, company, type, and description of each application is given in Table 2. Currently, all of these applications, with the exception of SE-1800 (used internally by Panneng, Inc.), have been released into the Android market.

The companies upload the beta versions of their applications to Kikbug and define the testing tasks. Each task may contain multiple, detailed requirements. The total number of tasks and requirements for each application is given in Table 3.

Table 2 Information of Android applications used in our study

Name	Company	Type	Description
iShopping (2018)	Alibaba	Shopping	Online shopping application
JustForFun (2018)	Ming Li, Inc.	Social	A picture-sharing social network
UBook (2018)	New Oriental Education	Education	An online education application
CloudMusic (2018)	NetEase, Inc.	Music	Music player and broadcasting station
SE-1800	Panneng, Inc.	Management	Dynamic information monitor for engineering projects

Most of the information regarding the applications (e.g., names, tasks, requirements, and comments) was written in Chinese. This information has been translated into English for the reader's convenience

Table 3 Testing tasks for all applications

Name	Number of tasks	Number of requirements
iShopping	7	37
JustForFun	4	29
UBook	5	25
CloudMusic	3	7
SE-1800	4	16

All tasks for these five applications only require functional testing. For example, the tasks of UBook are given in Table 4. Each task represents one function of UBook. The compensation for detecting a bug varies for different tasks. When a bug is detected by multiple crowdtesters, only the first one will be compensated. More discussion about the compensation scheme for CMAT can be found in Section 5.3.

In total, there are 258 crowdtesters involved and 1013 bug reports collected in this study. The detailed data for each application (one crowdtester can accept multiple testing tasks) is given in Table 5.

The devices of 258 crowdtesters cover 29 mobile brands (such as Samsung, Huawei, Sony, etc.), 181 Android models (Galaxy S4, Coolpad 5981, M1 Note, etc.), 22 Android versions (from 1.6 to 5.11), and 27 screen sizes (from 3.9 to 6.8 in.). The detailed distribution of mobile brands and OS versions are shown in Fig. 4. This demonstrates that our study surveys a good variety of Android devices.

Table 4 Sample testing tasks for UBook

ID	Task Name	Requirements
U_1	View & Search	1. View the content of a course. 2. Search for the course in which you are interested. 3. View the assignment for a course.
U_2	Sign up & Login	1. Sign up as a new student (receive the verification code via text message). 2. Use “Forgot Password” function to change the password. 3. Log in as a student. Try “Account Bound” function in the “Intro” page (optional). If the “Intro” page is skipped, there will be a pop-up toast. 4. Enter the “Learning” page and read instructions for subscribing. 5. Log in repeatedly without password.
U_3	Learning Materials	1. Subscribe to the materials and view the posted video. 2. List all materials. Download and view some of them. 3. Add new comments to a material. 4. Like and cancel. 5. Favorite and cancel. 6. Share one material on WeChat.
U_4	Course Video	1. List all videos and view one or two of them. 2. Add new comments to a video. 3. Like and cancel. 4. Favorite and cancel. 5. Share one video on WeChat.
U_5	Other	1. View the course timetable by week and month. Click “Today” button to view today’s course. 2. Change profile image. 3. Check the items subscribed to and add new item using subscribe ID. 4. Check items which are marked as “Favorite”. 5. Try options in “Settings” page. 6. Log out.

Table 5 Number of crowdtesters and bug reports (for the remainder of the paper, "bug report" refers to the final bug report generated by Kikbug) collected

Application	Number of crowdtesters	Number of bug reports collected
iShopping	151	268
JustForFun	59	222
UBook	142	199
CloudMusic	117	87
SE-1800	132	237

4 Presentation of Data Collected in Our Industrial Study

This section provides data collected in our industrial study from different perspectives. Section 4.1 gives the number of bugs detected for each application. Types of these bugs are reported in Section 4.2. Details about bug detection capability of crowdtesters is shown in Section 4.3. Section 4.4 presents the correlations between the number of crowdtesters and the number of detected bugs.

4.1 Number of Detected Bugs

Table 6 gives detailed information about the total number of bugs[4] detected for each application and each task.

In Table 6, the second column specifies the number of distinct bugs detected for each application, and the third column provides the distinct bugs detected for each test task. S_n, J_n, U_n, C_n, and SE_n represent the identity of the *n*th task for iShopping, JustForFun, UBook, CloudMusic, and SE-1800, respectively. For example, there are a total of 95 bugs detected for the application iShopping and 58 bugs detected for its first task. We make the following observations from Table 6:

- CMAT can be very effective in detecting bugs for Android applications. For all five applications, there are more than 20 bugs detected by the crowdtesters. Notably, there are 95 bugs detected for iShopping and 54 bugs for UBook.
- The sum of bugs detected in testing each task is not equal to the total number of bugs detected for the application because we observe that it is very common for one bug to be detected in the testing of multiple tasks. Inappropriate task design is the main reason for this drawback. We use U_2 as our example. To test other tasks of UBook, crowdtesters must first log in, which means some of the requirements in U_2 are covered when testing other tasks. There is a significant coupling between U_2 and other tasks. As a result, four, one, and five bugs detected in U_2 are also detected in testing U_3, U_4, and U_5, respectively.
- For iShopping and UBook, the number of bugs detected for different tasks within the same application varies significantly. For example, there are 58 bugs detected for the first task (S_1) of iShopping. However, only four bugs are detected by the second task (S_2). For S_1, there are three requirements for testing the main page of iShopping. However, each requirement only contains a single word – (1) Browse, (2) Search, and (3) Activities. As

[4] For the rest of the paper, a "detected bug" is one that is reported and also approved by the customers. A "reported bug" is not necessarily a "detected bug" unless the customers approve it.

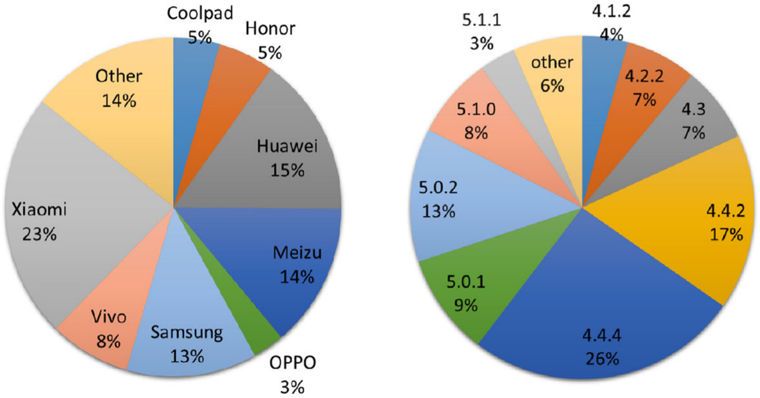


Fig. 4 Distribution of brands (left) and Android OS versions (right)

the vaguest and simplest task, S_1 attracts 133 crowdtesters to detect 58 bugs (both numbers are the largest for a single task). For S_2, the requirement becomes more complicated. We can observe that if the requirements of a task indicate to crowdtesters that this task can be very easily tested, then the task is more likely to involve more crowdtesters and consequently, more bugs are likely to be detected.

4.2 Types of Detected Bugs

There are multifarious bugs detected for these applications. Even though all tasks defined by the customers require only functional testing, the bugs detected are not limited to functional bugs. Based on the verification results from the customers, we have classified all detected bugs into three categories. For bugs in each category, we provide the description for one detected bug as an example.

- *Functional bugs*

Application does not behave the way it should according to the description in the testing task, however it is still running and does not exit. An example of a functional bug from CloudMusic is "The music can be downloaded but cannot be played."

Table 6 Number of bugs detected

Application	Total	Number of distinct bugs for each task						
iShopping	95	S_1 58	S_2 4	S_3 7	S_4 30	S_5 20	S_6 17	S_7 21
JustForFun	24	J_1 12		J_2 13		J_3 16		J_4 13
UBook	54	U_1 2		U_2 18		U_3 28	U_4 15	U_5 15
CloudMusic	28	C_1 11		C_2 12		C_3 15		
SE-1800	46	SE_1 20		SE_2 22		SE_3 12		SE_4 18

- *Crashes*

Application stops functioning properly and exits while testing a specific task. An example from JustForFun is "Application crashed when I try to share one image to my friends in WeChat."

- *Performance bugs*

Even though the application performs satisfactorily on a specific function, the crowdtesters have experienced significant performance issues. An example from iShopping is "It takes a significant amount of time to load all Gif images."

Furthermore, while testing, the crowdtesters also give some suggestions for improving the quality of an application. A suggestion from UBook is "The search function should provide associative-words for the keyword I type."

The number of bugs detected in each category, as well as the number of suggestions, is given in Table 7. We can observe that most of these bugs are functional bugs. However, the crowdtesters have a strong incentive to report whatever they experience, including crashes, performance bugs, and suggestions which they think need to be addressed. iShopping receives the most suggestions approved by the customers.

As we discuss in Section 1, since a crowdtester can be anyone with various testing facilities and environments, CMAT should perform well at detecting bugs related to compatibility issues. During the verification of bugs in our study, if a bug can only be reproduced on certain models of Android devices, or it is caused by a special setting of certain mobile devices, the customer will mark this bug as one related to compatibility issues. JustForFun provides an image browser for users. A bug related to compatibility issue in JustForFun is that when a user reaches the last image of an album, there will be a notification pop-up saying this is the last image. However, this notification does not show up for crowdtesters with Samsung and Oppo phones because these phones have special settings for notification balloons.

Table 8 gives the number of bugs related to compatibility issues approved by the customers. For example, there are 12 bugs (22.22% of the total number of bugs) related to compatibility issues in UBook. The data evidences the capability of CMAT in detecting bugs related to compatibility issues.

4.3 Bug Detection Capabilities of Crowdtesters

In our study, for each application, we observe that many crowdtesters have good bug detection capabilities in terms of detecting more than one bug in each application. Table 9 gives the

Table 7 Number of bugs in each category and number of suggestions

Application	Functional	Crash	Performance	Suggestions
iShopping	81	12	2	30
JustForFun	18	3	3	13
UBook	46	5	3	16
CloudMusic	20	4	4	5
SE-1800	34	9	3	6

Table 8 Number of bugs related to compatibility issue

Application	Number of bugs related to compatibility issue
iShopping	26 (27.36%)
JustForFun	4 (16.67%)
UBook	12 (22.22%)
CloudMusic	7 (25.00%)
SE-1800	9 (19.56%)

number of crowdtesters who detect a certain number of bugs for each application. For example, there are 51 crowdtesters who detect two bugs in iShopping.

For all five applications, over 40% of crowdtesters (71.74% for iShopping, 87.04% for JustForFun, 61.42% for UBook, 43.59% for CloudMusic, and 82.35% for SE-1800) can detect more than one bug. For iShopping, which has the highest number of detected bugs (95), there are seven crowdtesters who detect more than seven bugs. Among these seven, there are two crowdtesters who detect 13 and 14 bugs, respectively.

4.4 Correlation between the Number of Crowdtesters and Number of Detected Bugs

Intuitively, the more crowdtesters are involved in a task, the more testing efforts are put, and consequently the more bugs are likely to be detected. Figure 5 shows the number of crowdtesters involved and the number of bugs detected in each task of all applications. For example, 13 crowdtesters have detected seven bugs for task S_3 of iShopping.

From Fig. 5, we can observe that, in a general trend, the more crowdtesters get involved, the more bugs are detected. Also, for the number of testers as well as number of bugs detected, the difference across different tasks of JustForFun (from J_1 to J_4), CloudMusic (from C_1 to C_3), and SE-1800 (from SE_1 to SE_4) is not significant.

To further analyze the correlation between the number of crowdtesters and the number of detected bugs, we compute the Pearson and Spearman correlation coefficient (Hotelling 1953). For both Pearson and Spearman correlation coefficient, a value greater than 0 indicates a positive association; that is, as the number of crowdtesters increases, so does the number of detected bugs. The stronger the positive association of the number of crowdtesters and the number of detected bugs, the closer the Pearson and Spearman correlation coefficient will be to +1. In our study, the Pearson correlation coefficient is 0.646 and the Spearman correlation coefficient is 0.504. Both of them show that there is a relatively strong positive association between the number of crowdtesters and the number of detected bugs.

Table 9 Number of crowdtesters who detect a certain number of bugs

Applications	1 bug	2 bugs	3 bugs	4 bugs	5 bugs	6 bugs	7+ bugs
iShopping	39	51	20	10	6	4	7
JustForFun	7	6	12	16	7	4	2
UBook	49	29	23	15	6	2	3
CloudMusic	66	31	9	8	2	0	1
SE-1800	21	59	26	7	2	3	1

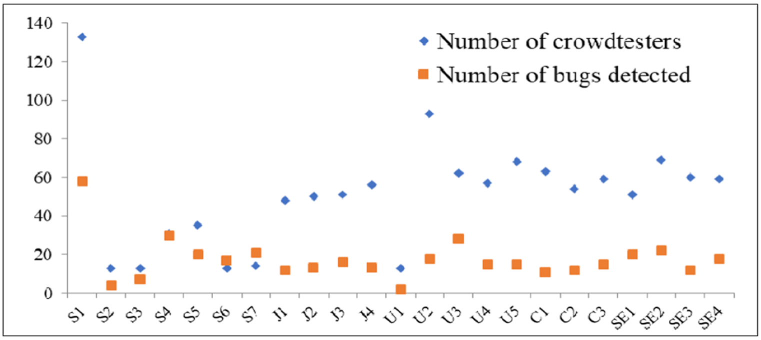


Fig. 5 Number of crowdtesters and number of bugs detected for each task of all applications

5 Discussion

In this section, we investigate four research questions based on the data collected from our industrial study. For each research question, we first describe its motivation followed by the measurement approach. Then, we present our collected data with detailed discussion.

5.1 RQ1: How does CMAT Compare to Traditional In-House Testing?

5.1.1 Motivation

The testing industry has been significantly shaken up by the rise of CMAT. How do you weigh up the pros and cons of having a dedicated in-house testing department, versus engaging crowdtesters? It is always an interesting subject to compare the performance of CMAT to that of in-house testing.

5.1.2 Measurement Approach

To conduct the comparison, an in-house testing for all five applications is conducted in the Advanced Research Center for Software Testing and Quality Assurance[5] (STQA). Nine Ph.D. students majored in software testing and quality assurance participated in this testing projected. All of them are experienced testers for Android applications. There are in total 20 different Android-based devices available in the testing center. One thing needs to be emphasized is that it is almost impossible to conduct a completely fair comparison for in-house testing and CMAT. However, we try to counter this threat by making the following two prerequisites: (1) for each application, in-house testing has to be completed in the same time as that used by CMAT. For example, it takes 3 days to collect all bug reports for iShopping. Then, for in-house testing, the testers have to deliver all their bug reports in the same time period. (2) for each application, the total compensation for in-house testing has to be same as that of CMAT. For in-house testing, all testers can work together and decide how much effort they should pay for each application based on the payment they receive. Table 10 summarizes the experimental setup of our CMAT industrial study and in-house testing. In this study, we recorded the total number of bugs detected by CMAT and in-house testing as well as the time spent by each approach.

[5] STQA is sponsored by NSF I/UCRC (Industry/University Cooperative Research Centers Program). You may visit http://paris.utdallas.edu/stqa for more detailed information about STQA.

Table 10 Experimental setup of our CMAT industrial study and in-house testing

Application	Number of testers		Number of bug reports		Time
	CMAT	In-house	CMAT	In-house	CMAT & In-house
iShopping	151	9	268	50	3 days
JustForFun	59	9	222	25	2 days
UBook	142	9	199	37	3 days
CloudMusic	117	9	87	21	3 days
SE-1800	132	9	237	32	5 days

5.1.3 Results and Discussion

The comparison results with respect to number of bugs detected are presented in Table 11. From this table, we can observe that, for all applications, the total number of bugs detected by the in-house testing is less than that of CMAT. Most of the bugs detected by the in-house testing can also be detected by CMAT. For example, only three bugs are identified by in-house testing but not by CMAT for JustForFun. For detecting bugs related to compatibility issue, CMAT obviously outperforms in-house testing. However, from Tables 10 and 11, we can also observe that CMAT can easily produce many bug reports. Some of these reports could be incorrect or meaningless.

5.2 RQ2: Will CMAT Result in Many Invalid Bug Reports?

5.2.1 Motivation

Invalid bug reporting is one of the biggest issues in applying crowdsourcing to software testing. Since the crowdtesters are at different skill levels, some customers are afraid that there may be many bug reports which cannot provide valuable information. It is important to investigate whether CMAT will produce many invalid bug reports.

5.2.2 Measurement Approach

In our study, there are 1013 bug reports collected for five applications. A crowdtester may report multiple bugs in one bug report. Based on the results of bug report verification, we classify all the bug reports into three categories.

- All Approved (AA)

Table 11 Comparison between in-house testing and CMAT

Application	Number of bugs detected		Number of bugs detected only by CMAT	Number of bugs detected only by In-house	Number of bugs related to compatibility issue	
	CMAT	In-house			CMAT	In-house
iShopping	95	46	49	0	26	9
JustForFun	24	18	9	3	4	0
UBook	54	35	24	5	12	2
CloudMusic	28	15	14	1	7	0
SE-1800	46	27	27	8	9	2

All bugs in the bug report are approved by the customers.

- Partially Approved (PA)

In the bug report, there is at least one bug not approved by the customers.

- Meaningless (M)

The description in the bug report is incomplete or does not provide any meaningful information.

For a given application, we may claim there are few invalid bug reports if both PA and M are very low.

5.2.3 Results and Discussion

Table 12 gives the number of bugs in each category (AA, PA, and M). We observe that, in our study, it is very uncommon for a crowdtester to submit an invalid bug report because both TN and M are small for each application.

However, we also observe that the number of bug reports in PA is small for iShopping (11 out of 268), JustForFun (13 out of 222), UBook (17 out of 199), and CloudMusic (2 out of 87), but relatively large for SE-1800 (33 reports out of 237). The reason behind this is that SE-1800 is the first internal-use application uploaded to Kikbug. It is specially designed for engineering monitoring. A crowdtester is more likely to use applications similar to iShopping, JustForFun, UBook, and CloudMusic in everyday life. However, the average crowdtester is much less likely to use specialty applications like SE-1800. Therefore, to conduct good testing, the crowdtesters need to have some domain knowledge to a certain extent. Accordingly, the possibility is higher that a bug reported by the crowdtesters is not considered a bug by the customers. Even though the number of FP for SE-1800 is not large, it signals to us that not every application is appropriate for CMAT. Indeed, CMAT can provide high effectiveness with low resources cost for testing mobile applications. However, before making the decision to use CMAT, customers have to ask themselves who their end users are. If they are the general users in the public at large, CMAT could be a great choice. However, if they are limited to a small number of people with certain expertise, the true assets of CMAT may not be represented thoroughly.

5.3 RQ3: Is the Current Compensation Scheme Reasonable?

5.3.1 Motivation

Compensation scheme is one of the key factors that may affect the effectiveness of CMAT. In most CMAT projects, crowdtesters work for financial compensation. Undercompensating may

Table 12 Number of bug reports in each category

Application	Total	AA	PA	M
iShopping	268	249	11	8
JustForFun	222	207	13	2
UBook	199	181	17	1
CloudMusic	87	83	2	2
SE-1800	237	201	33	3

lead to poor effectiveness in bug detection and minimal task testing by the crowdtester. However, overcompensating may result in unnecessary costs for customers. How to set up an appropriate compensation scheme is a hot topic for all projects applying crowdsourcing (Latoza and Van der Hoek 2016; Mantyla and Itkonen 2013; Zhang et al. 2014; Zogaj et al. 2014). This *pay-by-bug* compensation scheme or those similar are used not only by Mooctest but also by almost all companies providing CMAT services (UTest 2018; UserTesting 2018; Zogaj et al. 2014). In this scheme, after all reports have been verified, for each task of one application, the customers group the reports submitted by the same crowdtester and determine the amount of compensation he or she earned. The customers generally apply the following rules:

- If a crowdtester reports no bug or the customers do not approve any of the reported bugs, he receives no compensation.
- If multiple crowdtesters report the same bug which is approved by customers, only the first crowdtester who reports that bug receives compensation.
- If one bug is approved by the customers and a crowdtester is the first one who reports it, then that crowdtester is compensated per one or both of the following.

– The customers give a fixed basic compensation for detecting one bug in a task. This amount differs for different tasks. The customers publicize this information when they upload their application.

– If the crowdtester detects a bug or gives suggestions which the customers consider valuable, the crowdtester receives a floating bonus.

Regardless of variations among individual schemes, one aspect remains consistent: crowdtesters are paid for the number of bugs detected, not for the quality of their work. Is this reasonable?

5.3.2 Measurement Approach

When we compensate the crowdtesters involved in this study through our system, the crowdtesters need to answer the following question: “What do you think of the current compensation scheme? Is your compensation reasonable, underrated, or overrated? Please leave your comments” to receive their compensation. All crowdtesters replied. In addition to their feedback, we have also looked into some real scenarios to further evaluate the current compensation scheme.

5.3.3 Results and Discussion

The feedback from the crowdtesters showed that 56.2% of them think the compensation scheme is reasonable, 40.3% think the compensation is too low, and only 3.5% think they are overcompensated. A crowdtester who believes himself to be undercompensated commented “I found two bugs for this task. The second one took me a very long time. Why didn’t I receive a bonus for that one?” Since the basic compensation for detecting a bug for a task and whether the detection of a bug deserves a bonus is completely determined by the customers, it is highly possible that the compensation a crowdtester receives is not commensurate with the effort he or she exerts. Another crowdtester in UTest (Dargis 2013) makes a similar complaint: “From my perspective as a tester, the true down side to the UTest approach is that in most cases, you only get paid for the bugs you find and not the work you've put into finding them."

As we can observe, what crowdtesters complain about most is that their efforts are not appropriately considered by the customers. To detect different bugs, a crowdtester makes efforts and overcomes various difficulties. A fixed basic compensation with a possible bonus determined only by the customers might not account for the efforts and difficulty of detecting a bug. Consider the following real scenario, in iShopping, two bugs, B_1 and B_2, have been identified for one task. B_1 indicates that the QR code generated by the application cannot be recognized by other devices. B_2 indicates that when sending the voice message, if the duration of this message is extremely short (less than 1 s), there is no way to withdraw it. Both B_1 and B_2 are found by the same crowdtester. However, it takes five minutes to find B_1 but 40 min for B_2. This crowdtester receives the same compensation for B_1 and B_2.

In this scenario, we observe that the difficulties of identifying the input for triggering B_1 and B_2 are different. That task explicitly requires the crowdtester to generate a QR code and then scan it by using other devices. It will be easy for the crowdtester to detect B_1. However, to detect B_2, even though the crowdtester is required to test the function of sending voice messages, the test case for triggering B_2 is relatively difficult to locate. In this case, there is no bonus for B_2, since the customers think both B_1 and B_2 are in the same severity level (i.e., the customers consider B_1 and B_2 to have the same value).

There are other similar cases in our study, which may lead the crowdtester to believe that the number of bugs is the most important factor in computing their compensation. Therefore, in order to increase the possibility of winning more compensation, they should concentrate on detecting more easy-to-detect bugs as opposed to spending time and effort on a few hard-to-detect bugs. Because there is no guarantee that the customers will value those hard-to-detect bugs, or pay an adequate bonus for bugs they do value, crowdtesters might not consider it worth their while to spend time looking for hard-to-detect, but potentially valuable, bugs. In using CMAT, the customers hope to increase the chances of detecting valuable bugs by involving more crowdtesters who will spend the time and effort to find all bugs regardless of the level of difficulty involved. Unfortunately, if the efforts of crowdtesters are not appropriately valued, the motivation for them to work their hardest is certainly diminished.

Various bugs have multiple levels of difficulty to detect and have different values to the customers. However, in most cases, the value of the bug to the customer does not correspond to the level of difficulty in detecting it. Based on our results, the current compensation scheme is a working solution, but a more reasonable scheme should consider both the value of the bug and the difficulty of detecting it.

5.4 RQ 4: Are the Duplicated Bug Reports Useless in Testing Mobile Applications?

5.4.1 Motivation

For most companies using crowdsourced software testing, they do not prefer the submission of duplicated bug reports from crowdtesters. Thus, like Mooctest, they apply a first-come-first-pay policy and publicize all bug reports to avoid duplicated bug reports to the utmost extent. Is the first-come-first-pay policy really useful for preventing duplicated bug reports? If not, we want to investigate why such policy does not perform well. In addition, some studies suggest duplicated bug reports are meaningless in crowdsourced software testing (Leicht et al. 2016, 2017), is that true?

5.4.2 Measurement Approach

We collect the number of duplicated bug reports for each application and report the number of crowdtesters who detect the same bug for each application. We also selectively send emails to crowdtesters who have submitted more than 5 duplicated bug reports by asking why they refuse to follow the first-come-first-pay policy and submit duplicated bug reports. To investigate whether duplicated bug reports are useful or not, during the bug report verification, we have worked with the customers to check how they handle duplications.

5.4.3 Results and Discussion

In our study, even though the crowdtesters are notified that only the first to report an approved bug can receive compensation and they are able to review all the reports for a given application, we find that there are still many duplicated bug reports. Table 13 gives the detailed information about the number of duplicated bug reports for each application.

The second column shows the average number of duplicated bug reports. The third through fifth columns give the top three numbers in this regard. For example, for iShopping, there are on average 2.62 crowdtesters who detect the same bug. The three most popular bugs are detected by 12, 11, and 11 crowdtesters.

First, why there are so many duplicated results? A crowdtester commented that: "It takes me just ten minutes to finish testing a task and write my report. Why should I spend 30 minutes to look at others?" Because of its mobility, CMAT is very different from testing other kinds of software using crowdsourcing (e.g., desktop or web applications). The crowdtester does not have to sit in front of the computer, do testing, and write a long story about the testing process. The entire testing process of CMAT is fast and agile. The testing for CMAT can even be accomplished while the crowdtester is on a subway train. Hence, it is not reasonable for the CMAT crowdtester to review all the bug reports and avoid duplications if such review is not compensated.

Second, are these duplicated reports really useless for CMAT? In our study, the customers always put the bug reported by the most crowdtesters at the top of the bug fixing list. In other words, the duplicated reports automatically help customers prioritize the bugs to be fixed. Furthermore, referring to the example of the bug related to compatibility issues in Section 4.2, because all crowdtesters who report that bug have either Samsung or Oppo devices, there is a strong inference that the bug could be due to a compatibility issue. However, if there is only one bug report for that bug, this inference will be much harder to make. Hence, in CMAT, the information provided by duplicated bug reports is not useless. Nevertheless, the existence of duplicated bug reports decreases the efficiency of reports verification. Therefore, the benefits of duplicated bug reports should be taken advantage of in different ways**.**

Table 13 The number of duplicated bug reports for each application

Application	Average	Top 1	Top 2	Top 3
iShopping	2.62	12	11	11
JustForFun	6.21	35	33	29
UBook	3.33	38	36	10
CloudMusic	3.11	18	17	10
SE-1800	3.50	32	28	7

6 Threats to Validity

The observations, guidance, and conjectures in this paper are based on our industrial study. The results may not be generalized to all scenarios using CMAT. However, we took steps to counter this threat by participating in real-life projects, involving over 200 crowdtesters, and employing five Android applications with different functionalities to make our study more comprehensive. This allows us to have higher confidence with respect to the applicability of CMAT to provide promising bug detection effectiveness in different applications, and in the universality of our guidance and conjectures in this paper.

7 Related Studies

We now provide an overview of related studies in addition to the ones discussed in previous sections. Readers interested in further details are directed to the accompanying references.

Crowdsourced software testing is a new trend in the software engineering research community (Allahbakhsh et al. 2013; Cheng et al. 2015; Dolstra et al. 2013; Goldman et al. 2011; Goldman 2011; Harman et al. 2014; Huang et al. 2013; Liu et al. 2012; Mantyla and Itkonen 2013; Mujumdar et al. 2011; Pastore et al. 2013; Yuen et al. 2011). *A Survey of the Use of Crowdsourcing in Software Engineering* (Mao et al. 2015) presents a comprehensive review of the state of the art application of crowdsourcing in software engineering. Leicht et al. (2016) present two case studies from a Swiss bank and an industrial enterprise to compare the performance of crowdsourced software testing to that of in-house testing. They find that crowdsourced software testing is comparable in terms of testing quality and costs, but it provides large advantages in terms of speed, heterogeneity of testers and user feedback as added value. Leicht et al. (2017) present a digest overviewing different crowdsourcing approach from a practical point of view. Bruun and Stage (2015) present and evaluate two approaches to overcome usability evaluation obstacles in software development: a basic approach where software development practitioners are trained to drive usability evaluations; and a crowdsourcing approach where end users are given minimalist training to enable them to drive usability evaluations. MoTiF, a crowdsourced approach to support mobile app developers in automatically reproducing context-sensitive crashes is proposed by Gomez et al. (2016). In order to explore how crowdsourcing software testing and in-house testing can co-exist, Guaiani and Muccini run a survey with crowdtesters to understand their perception on this matter (2016). Liu et al. (2012) applies crowdsourced testing in usability testing on webpages. They simulate the testing environment to study the differences between crowdsourced software testing and traditional usability testing. They found crowdsourcing exhibits some notable limitations in comparison to the traditional lab environment, its applicability and value for usability testing is evidenced. Studies in (Pastore et al. 2013) explore the possibility of using crowdsourcing to solve the oracle problem. The authors produce tasks asking crowdtesters to evaluate CrowdOracles - assertions that reflect the current behavior of the program. If the crowdtester determines that an assertion does not match the behavior described in the code documentation, then a bug has been found. The experiment results demonstrate that crowdsourcing is a viable solution to automate the oracle problem. Dolstra et al. (2013) use crowdsourced testing to accomplish GUI testing. They run the GUI application on virtual machines to enable semi-automated GUI testing by crowdtesters. Nebeling et al. (2013) evaluate the usability of some web applications with crowdsourcing data. A test prioritization approach using the idea of Adaptive Random Testing and natural language processing is proposed in (Feng et al. 2015). Zogaj et al. (2014) work with a company in Germany to discover the management issues within crowdsourced software testing. Authors of (Haerem and Rau 2007; Mantyla and Itkonen 2013; Mok et al. 2017) discuss the possible impact on crowdsourcing based on the various skill levels of crowdworkers.

Some studies in crowdsourced software testing discuss the CMAT to a certain extent (Mao et al. 2015; Yuen et al. 2011; Zogaj et al. 2014). However, the number of studies focusing on CMAT is very limited. The author of (Xue 2013) combines static analysis and the idea of CMAT to propose CrowdBlaze, a mobile application testing system. CrowdBlaze initially explores the app with static analysis and automatic testing, and then recruits crowdtesters to provide input for testing certain applications. Authors of (Yan et al. 2014) propose a prototype framework, iTest, for CMAT with more advanced features in crowdtester recruitment. However, iTest is still under implementation and not publicly available, and they only conduct simulated experiments with 18 crowdtesters involved. As far as we know, this is the first paper discussing CMAT in a real-life, large-scale industrial study.

8 Conclusion and Future Work

CMAT has gained much attention in academia and industry over the last few years. Despite its popularity, there is still little well-founded research on how effective CMAT could be in practice, what major challenges are encountered when applying CMAT, and how these challenges can be tackled for further improvement of CMAT. To investigate these questions, we have conducted an industrial study by cooperating with Mooctest, Inc. and participating in real-life projects between Mooctest and five companies to test five Android mobile applications using CMAT. Throughout this study, we have collected 1013 bug reports from 258 crowdtesters and found 247 bugs in total. Based on the results, we observe that CMAT could be very effective at detecting bugs. However, when applying CMAT, challenges still exist on how to provide reasonable compensation and how to manage duplicated bug reports. We provide guidance for addressing these challenges. More industrial studies are currently in progress to further validate and improve our proposed new CMAT process.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61690201).

References

Adaptive Vehicle Make (2018) http://www.darpa.mil/news-events/2014-02-05

Allahbakhsh M, Benatallah B, Ignjatovic A, Motahari-Nezhad H, Bertino E, Dustdar S (2013) Quality control in crowdsourcing systems: issues and directions. IEEE Internet Comput 17(2):76–81

Amazon Mechanical Turk (2018) https://www.mturk.com/mturk/welcome

Bruun A, Stage J (2015) New approaches to usability evaluation in software development: barefoot and crowdsourcing. J Syst Softw 105:40–53

Capgemini (2017–2018) World Quality Report for Mobile Testing

Z. Chen and B. Luo (2014) "Quasi-crowdsourcing testing for educational projects," in Proceedings of international conference on software engineering, pp. 272.275, Hyderabad, India, Mary

J. Cheng, J. Teevan, M. S. Bernstein (2015) "Measuring crowdsourcing effort with error-time curves," *in* Proceedings of ACM conference on human factors in computing systems, pp. 1365–1374, Seoul, Korea

CloudMusic (2018) https://play.google.com/store/apps/details?id=com.netease.cloudmusic

CrowdMed (2018) https://www.crowdmed.com/

Crowdsourcing.org (2013) "Using crowdsourcing for software testing"

Lucas Dargis (2013) "Is UTest a Scam" http://lucasdargis.com/is-utest-a-scam/

E. Dolstra, R. Vliegendhart, and J. Pouwelse (2013) "Crowdsourcing GUI tests," *In Proceedings of the IEEE International Conference on Software Testing*, *Verification and Validation*, pages 332–341, Luxembourg

Y. Feng, Z. Chen, J. A. Jone, C. Fang, and B. Xu (2015) "Test report prioritization to assist Crowdsourced testing," in Proceedings of joint meeting on foundations of software engineering, pp. 225–236, Bergamo, Italy

M. Goldman (2011) "Role-based interfaces for collaborative software development," in Proceedings of the 24th annual ACM symposium adjunct on user Interface software and technology, pp. 23–26, Charlotte, USA

M. Goldman, G. Little, and R. C. Miller (2011) "Real-time collaborative coding in a web IDE," in Proceedings of the 24th annual ACM symposium on user interface software and technology, pp. 155–164, Santa Barbara, USA

M. Gomez, R. Rouvoy, B. Adams, and L. Seinturier (2016) "Reproducing context-sensitive crashes of mobile apps using Crowdsourced monitoring," in Proceedings of the international conference on mobile software engineering and systems, pp. 88–99, Austin, Texas

F. Guaiani and H. Muccini (2016) "Crowd and laboratory testing, can they co-exist? An exploratory study," in Proceedings of the second international workshop on CrowdSourcing in software engineering, pp. 32–37, Florence, Italy

Haerem T, Rau D (2007) The influence of degree of expertise and objective task complexity on perceived task complexity and performance. J Appl Psychol 92(5):1320–1331

M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu (2014) "Genetic improvement for adaptive software engineering," in Proceedings of the international symposium on software engineering for Adaptiveand self-managing systems, pp. 1–4, Austin, USA

Hotelling H (1953) New light on the correlation coefficient and its transforms. J R Stat Soc 15(2):193–232

J. Howe (2016) "The rise of crowdsourcing," Wired Magazine

Y.-C. Huang, C.-I. Wang, and J. Hsu (2013) "Leveraging the crowd for creating wireframe-based exploration of mobile design pattern gallery," in Proceedings of the companion publication of the 2013 international conference on intelligent user interfaces, pp. 17–20, Santa Monica, USA

JustForFun (2018) http://apk.hiapk.com/appinfo/com.xp.tugele

Latoza TD, Van der Hoek A (2016) Crowdsourcing in software engineering: models, motivations, and challenges. IEEE Softw 33(1):74–80

N. Leicht, N. Knop, I. Blohm, C. Müller-Bloch, and J. M. Leimeister (2016) "When is crowdsourcing advantageous? The case of Crowdsourced software testing," in Proceedings of European conference on information systems, pp. 1–17, Istanbul, Turkey

Leicht N, Blohm I, Leimeister JM (2017) Leveraging the power of the crowd for software testing. IEEE Softw 34(2):62–69

D. Liu, M. Lease, R. Kuipers, and R. Bia (2012) "Crowdsourcing for usability testing," in *Proceedings of the American Society for Information Science and Technology*, vol. 49, no. 1, pp. 1–10

Mantyla MV, Itkonen J (2013) More testers - the effect of crowd size and time restriction in software testing. Inf Softw Technol 55(6):986–1003

K. Mao, L. Capra, M. Harman, and Y. Jia (2015) "A survey of the use of crowdsourcing in software engineering, " Research Note, University College London

Mok R, Chang R, Li W (2017) Detecting low-quality workers in QoE Crowdtesting: a worker behavior-based approach. IEEE Transactions on Multimedia 19(3):530–543

D. Mujumdar, M. Kallenbach, B. Liu, and B. Hartmann (2011) "Crowdsourcing suggestions to programming problems for dynamic web development languages," in Proceedings of the 2011 annual conference extended abstracts on human factors in computing systems, pp. 1525–1530, Vancouver, Canada

MyCrowd (2018) https://mycrowd.com/

M. Nebeling, M. Speicher, and M. C. Norrie (2013) "CrowdStudy: General Toolkit for Crowdsourced Evaluation of Web Interfaces," in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pp 255–264, London, UK

OpenSignal, Android Fragmentation Visualized (2015)

F. Pastore, L. Mariani, and G. Fraser (2013) "Crowdoracles: can the crowd solve the Oracle problem?" *In Proceedings of the IEEE International Conference on Software Testing*, *Verification and Validation*, pages 342–351, Luxembourg iShopping (2018) https://play.google.com/store/apps/details?id=com.taobao.shopstreet

Uber (2018) https://www.uber.com/

UBook (2018) http://www.pgyer.com/qe34

UserTesting (2018) https://www.usertesting.com/

UTest (2018) https://www.utest.com/

Waze (2018) https://www.waze.com/

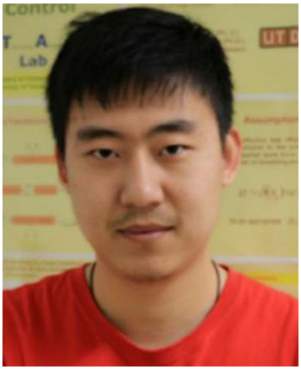
H. Xue (2013) "Using redundancy to improve security and testing," Ph.D. dissertation, University of Illinois at Urbana-Champaign

M. Yan, H. Sun, and X. Liu (2014) "iTest: testing software with mobile crowdsourcing," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, pp. 19–24, Hong Kong

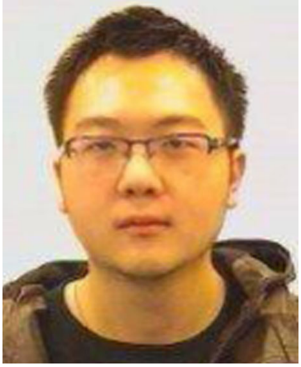
M. Yuen, I. King, and K. Leung (2011) "A survey of crowdsourcing systems," in *Proceedings of IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE Conference on Social Computing*, pp. 766–773, Boston, USA

Zhang X, Yang Z, Zhou Z, Cai H, Chen L, Li X (2014) Free market of crowdsourcing: incentive mechanism Design for Mobile Sensing. IEEE Trans Prallel Dist Syst 25(12):3190–3200

Zogaj S, Bretschneider U, Leimeister JM (2014) Managing Crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. J Bus Econ 84:375–405



Ruizhi Gao received the BS degree in software engineering from Nanjing University and the MS degree in computer science from the University of Texas at Dallas. He received the PhD degree in software engineering under the supervision of Professor W. Eric Wong with the University of Texas at Dallas. His research focuses are on software testing, fault localization, and program debugging. He is now working as a senior software development engineer at Sonos Inc.



Yabin Wang received his Bachelor degree in Software Institute from Nanjing University. He received his Ph.D. at the Software Institute, Nanjing University, under the supervision of Prof. Zhenyu Chen and Prof. Bin Luo. His research interest is software testing. He visited the University of Texas at Dallas as a visiting student under the supervision of Prof. Eric Wong in 2012.

Yang Feng is a PhD student in Software Engineering at the University of California, Irvine, advised by Prof. James. A. Jones. His research focuses on building intelligent tools to assist various software engineering tasks. His current research interests include software testing, debugging, program analysis, and program comprehension. Before joining UC Irvine, he received his B.E and M.E. degree in software engineering at Nanjing University, China, advised by Prof. Zhenyu Chen and Prof. Baowen Xu.

Zhenyu Chen is the founder of Mooctest (mooctest.net), and he is currently a Professor at the Software Institute, Nanjing University. He received his bachelor, and Ph.D. in Mathematics from Nanjing University. He worked as a Postdoctoral Researcher at the School of Computer Science and Engineering, Southeast University, China. His research interests focus on software analysis and testing. He has more than 70 publications in journals and proceedings including TOSEM, TSE, JSS, SQJ, IJSEKE, ISSTA, ICST, QSIC etc. He has served as the associated editor for IEEE Transactions on Reliability, PC co-chair of QRS 2016, QSIC 2013, AST2013, IWPD2012, and the program committee member of many international conferences. He also founded the NJSD (Nanjing Global Software Development Conference). He has won research funding from several competitive sources such as NSFC. He is a member of the IEEE.

W. Eric Wong received the MS and PhD degrees in computer science from Purdue. He is a full professor and the founding director of the Advanced Research Center for Software Testing and Quality Assurance in Computer Science, University of Texas at Dallas (UTD). He also has an appointment as a guest researcher with National Institute of Standards and Technology (NIST), an agency of the US Department of Commerce. Prior to joining UTD, he was with Telcordia Technologies (formerly Bellcore) as a senior research scientist and the project manager in charge of Dependable Telecom Software Development. In 2014, He was named the IEEE Reliability Society Engineer of the Year. His research focuses on helping practitioners improve the quality of software while reducing the cost of production. In particular, he is working on software testing, debugging, risk analysis/metrics, safety, and reliability. He is the Founding Steering Committee chair of the IEEE International Conference on Software Quality, Reliability, and Security (QRS) and the editor-in-chief of the IEEE Transactions on Reliability.