An Analysis of Android Bugs for Mobile Applications[fix?]

XXXXXXXXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX, XX, XXX
XXXXXX@XXXXX.XXX

ABSTRACT

The emergence of mobile applications (apps) have not only changed the way we use software, but in how we live our lives. Unfortunately, mobile software is not immune to the bugs that affect all software. These defects range from those which may inhibit the user's experience and cause varying levels of frustration, all the way to serious security vulnerabilities which may place the user in danger from malicious attackers.

In order to better understand mobile bugs and how they are resolved, we examined the bug reports and version control systems of ten Android applications from seven genres. Our objective for this research is to understand the life cycle of Android bugs and the relationship between user ratings and the number of bugs. We found that higher quality bug reports help developers fix bugs faster, and that apps with more bugs generally have a lower user rating.

[Dan says: make really good]

1. INTRODUCTION

Mobile devices have become a ubiquitous part of our lives. Smartphones are really no longer primarily just phones, but are computers we carry around with us that just happen to be capable of placing calls. The fight for market share of Android apps is extremely competitive with over 1.6 million available apps in the Google Play store alone [2]. Customers demand that apps have few defects, and are constantly being updating for new devices and feature improvements. Buggy, or seldom updated apps will likely receive poor user ratings, which will lead to fewer downloads [10].

The mobile development process is similar to conventional software development. Developers still typically use version control systems during the lifecycle of an application to allow them to collaborate and share code and bug information. Like traditional software these apps unfortunately suffer from bugs which may be caused by developers misunderstanding requirements or documentation, simple coding mistakes, improper use of 3rd party libraries, or even problems with the 3rd party libraries used in many apps**[cite]**. Bug reports are invaluable for the tracking and triaging of defects. These bugs may be discovered and reported by developers, testers, or users.**[cite]** .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'16, April 4-8, 2016, Pisa, Italy.
Copyright 2016 ACM 978-1-4503-3739-7/16/04...$15.00.
http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx ...$15.00.

The timely discovery and mitigation of bugs is important of several reasons. The first is that bugs increase the maintenance costs of software which is potentially very problematic for an app since the maintenance phase of a software project has been found to typically comprise at least 50% of the cost of a software project [14]. Very often, bugs are security vulnerabilities that place users in danger in a variety of areas including theft of everything from a user's location all the way to banking passwords [15]. Finally, bugs often just make users angry. With innumerable Android apps in each genre, users have many options and will often decide to stay away from buggy apps or ones with generally poor reviews [17]. Ultimately, bugs often lead to lost revenue for the developers.

The goal of this work is to better understand the lifecycle of bugs in open-source Android apps and better ways of fixing them[Dan says: make perfect]. In order to help better understand Android bugs, and ways of fixing them faster, we analyzed the bug reports of 10 different open source apps.

RQ1: *How can the quality of bug reports help the contributors fix the bugs sooner?*
We found that the time required to fix defects was less for bugs that have longer description lengths and a high number of keywords. These keywords and descriptions likely made it easy for the developers and contributors to understand the bugs and get them fixed more quickly.

RQ2: *What is the relation between the rating of Application and number of Bugs?*
We determined if there was a correlation between the number of reported bugs for an application with the user rating of an app. We found that the apps which had a higher bug count typically had a lower user rating. This is important since it demonstrates the importance of developing high quality, low defect apps in order to gain essential user ratings points.

The remainder of this paper is organized as follows: Section **??** describes our experi**[finish]**

2. RELATED WORK

There is a significant amount of previous research related to bugs in Android applications. Bhattacharya et. al. [4] performed an empirical analysis to understand the bug fixing process in the Android platform and Android based applications. Some of the examined metrics included bug fix time, bug categories, bug priorities and also the interest of the users and developers to fix the bugs.

Previous research has demonstrated the importance of producing high quality apps and their direct impact on user ratings. Using FindBugs [1], Khalid et al. [9] examined the relationship of code quality and the app's user rating. They found that certain defects were more likely to lead to lower app ratings. Defects such as 'Bad Practice', 'Internationalization', and 'Performance' are typi-

cally found in lower rated apps. They found that they could limit user complaints about an app using static analysis tools such as FindBugs. Vasquez et al. [12] studied over 7,000 Android apps to determine how the fault proneness of APIs contributed to an application's lack of success. They found that APIs used by highly rated apps are much less fault prone than APIs used by low-rated apps. While these works were quite interesting, they all used static analysis tools to located defects, while we used the actual bug reports for the apps.

While this work was innovative...

There are several other studies in bug fix times in open source apps, but none that we are aware of that specifically analyze android apps (check this) [3, 13]

[Add more relevant related work]

3. ANALYSIS

In the following sections, we will describe the selection criteria for choosing our ten chosen mobile apps, and then answer our primary research questions.

3.1 App & Data Selection

For our analysis, we chose apps which were hosted on Google Play and the F-Droid[1] open source repository. Google Play contains the user ratings required for analysis, while F-Droid provides links to the app's version control systems and bug repositories. These bug tracking systems provide a myriad of useful information including all noted bugs, open & closed defects, the date they were found, and the date they were fixed. In our collection of an app's source code and other analytics, we used the work of Krutz et al. [11] as a guide. We chose these 10 apps based on their popularity, diversity and our ability to collect necessary information from their version control systems and bug repositories.

Table 1 provides the details of the 10 mobile applications selected for the project. We display the app's category, the download count, number of people rated the app, total number of releases and the bug count. The defined the bug count to be the sum of the open bugs and closed bugs since the app's inception.

3.2 Research Questions

RQ1: *How can the quality of bug reports help the contributors fix the bugs sooner?*

Our first research question was how well the quality of bug reports help the developers to fix the bugs easily and quickly. In order to answer this question, we have taken into account different characteristics of bug reports. These characteristics include the length of the bug description in the bug reports and number of keywords found in the description [4]. They keywords that we have considered are version, component, security, vulnerability, attack, failure, error, crash, buffer overflow, buffer overrun, question, problem, invalid, and incorrect.*[Dan says: why?]* We used a custom built script to collect this necessary information.

Also we wrote a script which took the input as the above mentioned keywords and found them in the bug descriptions. The description having highest number of keywords along with sufficient description length were chosen. The bug descriptions which were too lengthy and did not have high count of the keywords were ignored. Also the bug descriptions which were very short in length but had large number of keywords were discarded. Further, we calculated an average of description length and the average of number of keywords for each application. Table 2 shows an example of how each bug from every application was analyzed to find the description and number of keywords and the corresponding time spend to fix the bug. **[why these words?][reword all of this]**

[average keywords does not make sense]

We obtained the results as described in the Table 3 which shows the app, total number of bugs, average length of each bug description, average number of keywords in the description, and average time required to fix the bugs. After careful analysis, we found that the time span required to fix the bugs was less for the bugs which had good description length along with large number of keywords. The keywords and description made it easy for the developers and contributors to understand the bugs and get them fixed as early as possible. As we can see for the Zxing application, the average length is 60 and average number of keywords for the bug report are 112 so the average fix time is less 10 days. Similarly for Simon puzzle application the average length is 46 and in proportion to the length, average keywords is 40 so time span is 6 days.

Table 3 provides each application with total number of bugs, average length, average keyword, average time span of each bug to fix. This data shows that the quality of bug reports affect the speed of fixing the bug. The keywords can allow developers to easily understand and locate the bug. The length of bug report and the time for fixing is a negative correlation. The more information the developers get, the faster the bug can be fixed. Through the analysis of apps and the number of bugs they have, we found out the applications having higher ratings have smaller bug counts. The bug can reduce a user's experiences, thereby decrease the ratings.

RQ2: *What is the relation between the rating of Application and number of Bugs?*

In Google Play the people who downloaded it rate the application from 1 to 5 stars that is, from average to very good. Along with ratings of the stars, the people write their review on the the application use. The ratings is sum of the people rated application and people wrote reviews for it. The bugs count is from the GIT repository. The analysis is made for each selected 10 applications.

Figure 1 shows the relation between the ratings of the application and bugs count for each individual application, which shows the rating and bugs count to be inversely proportional. Higher rated apps typically have lower bug counts. Of the examined apps, the only one which goes against this trend is XBMC, a multimedia player.

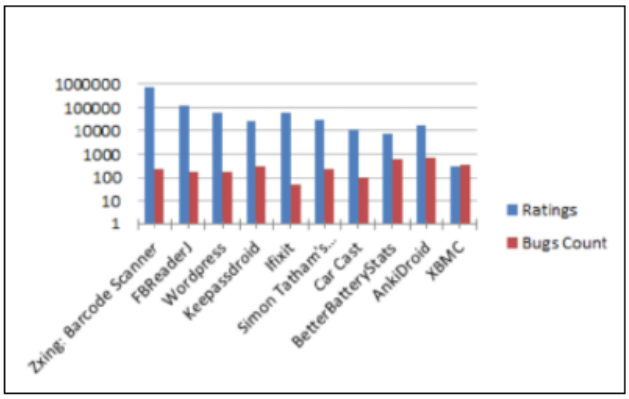

Figure 1: XXXX[make into latex]

We next decided to perform a case study on a single Android app to better understand how user reviews correlated with reported

[1]http://f-droid.org

Table 1: App Info

Name	Category	Downloads	Ratings	Releases	Bug Count
AnkiDroid	Education	1,000,000 - 5,000,000	18,166	389	745
BetterBatteryStats	System	100,000 - 500,000	7,986	139	601
Car Cast	Multimedia	100,000 - 500,000	1,240	77	121
FBReaderJ	Education	10,000,000 - 50,000,000	128,429	306	325
Keepassdroid	Security	1,000,000 - 5,000,000	28,904	110	317
Ifixit	Utility	500,000 - 1,000,000	5,760	28	251
Simon Tatham's Puzzles	Game	100,000 - 500,000	31,469	56	230
Wordpress	Editor	1,000,000 - 5,000,000	63,185	69	131
XBMC	Multimedia	100,000 - 500,000	281	81	343
Zxing: Barcode Scanner	Utility	100,000,000 - 500,000,000	704,060	16	372

Table 2: Example Bug Information

App	Bug ID	Bug Title	Time	Description Length	Keyword Count
AnkiDroid	105	Allow users to change AnkiDoird directory if current one is invalid	12D	190	3
FBReaderJ	219	Fatal exception in BookDownloaderService	Open	171	3
zxing	308	Possible ReedSolomon decoding problem	2D	253	4
CarCast	71	Review if debug mode is needed on release builds	978D	30	3
iFixit Android	106	SSL errors on Android 2.2	1D	313	4
KeeppassDroid	39	2nd try...	3D	68	1
sgtpuzzles	9	Build Failed	Open	79	3
WordPress	104	Bugfix - upload post thumbnails	3min	86	2

bugs. Specially, we examined if users are satisfied with application and not asking for many features and also whether the application has many bugs that the user are reporting.

We noticed that the *ZXING*[2] had a high level of user ratings and relatively low bugs count. Therefore, we decided to further analyze this app to understand any possible correlations between these values. We began by extracting user reviews from the ZXING application page in Google Play store and then classify them into feature requests and bug reports. This was accomplished by implementing an algorithm that splits a text into sentences, normalizes them, and compares them with a set of linguistic rules to find if it matches any of them. We used two set of rules for our classification. One is based on the linguistic rules defined by Lacob et al. [6] for feature requests, the other one is based on the linguistic rules they defined for bug reports [7]. We adapted the syntax of the rules to work with OpenNLP[3], the API we used for part-of-speech tagging. OpenNLP is a machine learning based toolkit for the processing of natural language text.

For issue classification, we created some rules due to assist with classifying the text.Table 4 shows some examples of linguistic rules for identifying feature requests and an example text that would be a match for each. Table 5 shows some examples of linguistic rules for identifying bug reports and an example text that would be a match for each. **[reword this section]**

We used Lingpipe[4] and OpenNLP to create an algorithm that classified the reviews based on those rules. Lingpipe is a toolkit for processing text using computational linguistics. We began by using Lingpipe to split the reviews into sentences. After the review was split into sentences, we normalized each sentence, replacing common misspelled words and abbreviations. When then used OpenNLP to tag the sentence where each word in the review sentence was tagged as a part of speech, e.g. for the text "it would be great" the tagger would generate "<personal_pronoun> <modal> <verb> <adjective>".

We ran our algorithm twice for each review we extracted. The first time it compared the review sentence with the linguistic rules defined for feature requests, and if it matched one of the rules, it classified the sentence as a feature request. If a sentence of a review is considered a feature request, then the whole review is marked as such. If any of the sentences is not a feature request, then the review is marked as not a feature request. The second time it did the same, but instead of comparing the sentences with the rules for feature requests, it compared them to the rules for bug reports. After the reviews in the database were classified, we counted the bug reports and feature requests for the application.

3.3 Discussion

[add to this]

4. PUBLIC DATASET

5. LIMITATIONS & FUTURE WORK

While we feel that our research was profound, there are several possible areas of improvement. In this study, we only analyzed 10 apps which represents only a very tiny portion of all Android apps. Future research may be done to broaden this study to more apps and more genres of apps.

Detecting and defining defects in software is a difficult task. While static analysis tools have demonstrated their benefits in numerous previous works [8, 9], they are far from perfect and of-

[2]https://play.google.com/store/apps/details?id=com.google.zxing.client.android
[3]https://opennlp.apache.org/
[4]http://alias-i.com/lingpipe/

Table 3: Bug Descriptions & Repair Time

Name	# of Bugs	Avg. Description Length	Avg. Keyword #	Avg. Fix Time (Days)
Zxing: Barcode Scanner	372	60	112	10
FBReaderJ	325	48	52	16
Wordpress	131	18	16	99
Keepassdroid	317	36	20	59
Ifixit	251	57	26	112
Simon Tatham's Puzzles	231	46	40	6
Car Cast	121	43	6	139
BetterBatteryStats	616	55	87	15
AnkiDroid	745	42	54	3
XBMC	352	52	61	4

Table 4: Examples of rules to identify feature requests

Rule	Text Match
Would be <adjective> if	It would be great if
Would <adverb> like to <verb>	Would really like to see
Needs option to	Needs options to share posts

Table 5: Examples of rules to identify bug reports

Rule	Text Match
<adverb> annoying	Incredibly annoying
Won't <verb>	Files won?t open
Keeps on crashing	Reader keeps on crashing

ten produce a large number of false positives or miss actual defects [5, 16]. Another way to determine the amount of bugs in software is through the examination of the application's bug reports. Unfortunately, the absence or presence of errors in bug reports is not always complete. Bugs may not exist in these reports merely because they have yet to be discovered by the users. Less users could mean that less bugs would be reported merely due to the fact that less people were using the app to report the issues. Additionally, duplicate bug entries are often made in bug reporting systems along with entries which are not actually bugs. To combat this, we only considered bug report items which had been considered to be legitimate, and 'resolved' in our analysis and ignored defects which had been marked as duplicate, or not actually a bug. Since we only analyzed apps with at least 100,000 downloads, we feel that this offers a reasonable enough sized user based to have found the majority of bugs.

6. CONCLUSION

In this paper, we analyzed the bug history of ten Android apps to better understand how the quality of bug reports can help developers fix bugs sooner, and the relationship between app ratings and bugs. We found a positive correlation between the quality of bug reports and a shorter time to repair the bugs. We also discovered that apps with more bugs typically had a lower user rating.

7. REFERENCES

[1] Findbugs - find bugs in java programs. http://findbugs.sourceforge.net, 2015.

[2] Number of apps available in leading app stores as of july 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2015.

[3] P. Bhattacharya and I. Neamtiu. Bug-fix time prediction models: Can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 207–210, New York, NY, USA, 2011. ACM.

[4] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru. An empirical analysis of bug reports and bug fixing in open source android apps. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 133–143, Washington, DC, USA, 2013. IEEE Computer Society.

[5] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, (6):76–79, 2004.

[6] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44, Piscataway, NJ, USA, 2013. IEEE Press.

[7] C. Iacob, R. Harrison, and S. Faily. Online reviews as first class artifacts in mobile app development. 6 2014.

[8] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 672–681. IEEE, 2013.

[9] H. Khalid, M. Nagappan, and A. Hassan. Examining the relationship between findbugs warnings and end user ratings: A case study on 10,000 android apps. *Software, IEEE*, PP(99):1–1, 2015.

[10] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan. What do mobile app users complain about? a study on free ios apps. *IEEE Software*, 99(PrePrints):1, 2014.

[11] D. E. Krutz, M. Mirakhorli, M. S. A., A. Ruiz, J. Peterson, A. Filipski, and J. Smith. A dataset of open-source android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. ACM, 2015.

[12] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: a threat to the success of android apps. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pages 477–487. ACM, 2013.

[13] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 11:1–11:8, New York, NY, USA, 2011. ACM.

[14] C. B. Seaman. Software maintenance: Concepts and practice. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(2):143–147, 2001.

[15] H. Shahriar and H. M. Haddad. Content provider leakage vulnerability detection in android applications. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, pages 359:359–359:366, New York, NY, USA, 2014. ACM.

[16] F. Thung, Lucia, D. Lo, L. Jiang, F. Rahman, and P. T. Devanbu. To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 50–59, New York, NY, USA, 2012. ACM.

[17] R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pages 241–244, New York, NY, USA, 2012. ACM.