2016 IEEE 27th International Symposium on Software Reliability Engineering

Experience Report: Understanding Cross-Platform App Issues From User Reviews

Yichuan Man*†‡, Cuiyun Gao‡§, Michael R. Lyu‡§, and Jiuchun Jiang*†
*National Active Distribution Network Technology Research Center (NANTEC), Beijing Jiaotong University, Beijing, China
†Collaborative Innovation Center of Electric Vehicles in Beijing, Beijing Jiaotong University, Beijing, China
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
§Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
14121444@bjtu.edu.cn, {cygao, lyu}@cse.cuhk.edu.hk, jcjiang@bjtu.edu.cn

***Abstract*—App developers publish apps on different platforms, such as Google Play, App Store, and Windows Store, to maximize the user volumes and potential revenues. Due to the different characteristics of the platforms and the different user preference (*e.g.*, Android is more customized than iOS), app testing cases on these three platforms should also be designed differently. Comprehensive app testing can be time-consuming for developers. Therefore, understanding the differences of the app issues on these platforms can facilitate the testing process.**

In this paper, we propose a novel framework named CrossMiner **to analyze the essential app issues and explore whether the app issues exhibit differently on the three platforms. Based on five million user reviews, the framework automatically captures the distributions of seven app issues, *i.e.*, "battery", "crash", "memory", "network", "privacy", "spam", and "UI". We discover that the apps for different platforms indeed generate different issue distributions, which can be employed by app developers to schedule and design the testing cases. The verification based on the official user forums also demonstrates the effectiveness of our framework. Furthermore, we also identify that the issues related to "crash" and "network" are more concerned by users than the other issues on these three platforms. To assist developers in gaining a deep insight on the user issues, we also prioritize the user reviews corresponding to the issues. Overall, we aim at understanding the differences of issues on different platforms and facilitating the testing process for app developers.**

I. Introduction

Smartphones have penetrated into people's daily life. By 2015, the global user volume of smartphones has exceeded half the world's population [10]. Accounting for this popularity is the growing creation and usage of mobile applications (*i.e.*, apps). To distribute the apps to users, developers are required to publish the apps on the distribution platforms specific for mobile apps. Generally, app developers choose to deliver their apps on more than one platform to enlarge the potential user volume and revenue [15]. Currently, the three largest global platforms for app distribution are Google Play, App Store, and Windows Store, which occupy 82.8%, 13.9%, and 2.6% of the market, respectively [13]. These three platforms are also the focus of our study.

To ensure the user experience, developers should examine the software reliability before the app delivery. The unique characteristics of the operating systems indicate that the testing on these platforms are not exactly the same [11], shown in Fig. 1. For example, Android is more customizable and offers an open platform, while iOS prioritizes the user interface over just about anything [8]. Furthermore, the users of different platforms possess different preferences. For example, iOS users are considered to be more "addicted" to digital devices than Android users [18]. Therefore, different platforms may generate different app issues, and understanding the differences facilitates the app development process for the developers.

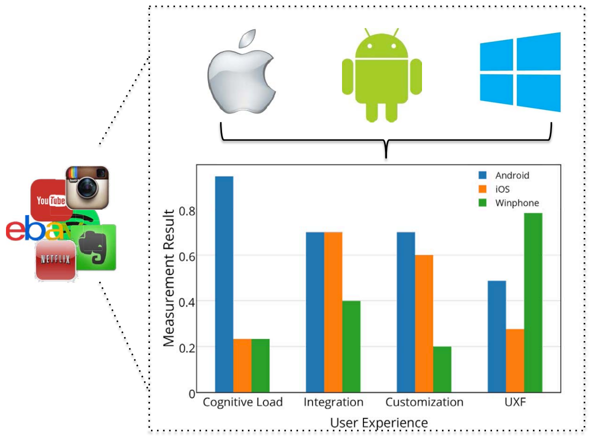


Fig. 1: User Experience on Different Platforms. Here, "UXF" denotes the user experience friction - the aspects of a device that can annoy users in a niggling way.

The existing studies concentrate on comparing the characteristics of the operating systems, such as the accessing Internet streaming services [32], security mechanisms [19], and the demographics [20]. There is no exploration of the differences of app issues on these platforms. Although one piece of work [37] analyzes bugs and bug-fixing for projects on different platforms, the work focuses on the updating rates and bug details. In this paper, we aim at comparing the app issues for the platforms and provide the developers with insights on testing apps.

Since user reviews provide a valuable data source for developers to identify potential issues of their mobile apps [23], we employ user reviews to discover the app issues. We have crawled about five million user reviews of 20 apps for the three platforms (*i.e.*, Google play, App Store, and Windows Store). By examining the app reviews, we have chosen seven issues for comparison. They are "crash", "battery drainage", "memory consumption", "network connection", "privacy", "spam", and "UI design". We have designed a novel framework named CrossMiner to comprehend these issues distributions on these platforms, and provide developers with crucial issues for different platforms.

To analyze the differences of app issues on different platforms, we first preprocess raw reviews to obtain input for word2vec [16] model and convert each word into a vector. Then, we cluster the words by k-means algorithm and summarize the corresponding keywords for each issue based on cosine similarity method. By employing the keyword-based method, we compute and visualize the distribution of each issue in different platforms. For better understanding specific issues, we also prioritize important reviews accordingly for the developers. We have conducted an ***empirical study*** on a large scale dataset (4,663,316 reviews of 20 popular apps), and demonstrate the differences and similarities existing along with the three platforms. We also verify that CrossMiner can reflect the importance of the user concerns accurately ultimately.

In this paper, we intend to answer the following three questions:
1) Are there any differences and similarities of different platforms regarding the issue distributions? (Section V-D)
2) Can CrossMiner embody the user concerns accurately? (Section V-E)
3) Can CrossMiner reflect platform-level issues? (Section V-F)

Generally, CrossMiner captures the representative keywords corresponding to one specific issue from massive user reviews, and aims at discovering the differences of user perceptions on different platforms. The contributions are summarized as following:

- We first propose a framework CrossMiner to extract issue-related keywords comprehensively from real user reviews, which can be facilitated for other research applications.
- We discover the differences and similarities on different platforms from users' perspective, especially from user reviews.
- We demonstrate that our framework reflects the importance of user concerns accurately. The developers can also analyze the detailed concerns based on the prioritized user reviews.

The remainder of this paper is organized as follows. Section II discusses the motivation of our exploration on user issues in different app stores. Section III illustrates the overall framework of CrossMiner. Section IV explains the issue-prioritizing model of our study. Section V presents the experimental results of 20 popular apps and demonstrates the effectiveness of our method based on official user forums. Section VI discusses some possible limitations and threats to validate. Section VII presents the related studies. Finally, Section VIII concludes the paper.

II. Motivation and Background

A report from [10] represents the quantity of apps available for downloading in leading app stores during July 2015. There are more than 1.6 million, 1.5 million, and 0.34 million in Google Play, App Store, and Windows Store, respectively. As a process for improving app's functionality, usability, and consistency, mobile app testing determines the delivery quality to end users, and becomes increasingly important for any companies that desire to keep competitive in the intensive app markets.

However, designing comprehensive app testing cases is time-consuming and sometimes difficult for app developers. One key challenge for the app testing is attributed to the diverse mobile platforms, such as Android, iOS, and Windows Phone. Each mobile operating system possesses unique limitations and properties. App testing across different platforms requires app developers to be familiar with the characteristics of each platform, and design test cases specifically. Moreover, users of different platforms embody different preferences and perceptions about an app [11]. Therefore, comprehending the user issues on different platforms can facilitate the whole process for app developers.

User reviews can be regarded as the "voices of users". They directly reflect the user experience [34]. Since analyzing user reviews assists developers in fixing bugs and adding new features [23], different user concerns on different platforms can be captured by utilizing the corresponding app reviews. Thus, developers can test apps more specifically and efficiently based on the extracted user issues.

In this paper, we select seven issues which are crucial for app testing [7], [17]. They are "battery", "crash", "memory", "network", "privacy", "spam", and "UI". To verify whether users concern these issues practically, we take Facebook as an example and examine the corresponding user reviews. Table I illustrates the user reviews regarding these issues.

In this paper, we aim at implementing a framework, namely CrossMiner, to help developers understand the differences of app issues on different platforms based on user reviews. Developers can then focus on the important issues on these platforms during the app testing. Given the app reviews of each platform, CrossMiner automatically prioritizes the issues on this platform. We focus on the seven issues shown in Table I.

III. Framework

Fig. 2 illustrates the overview of the proposed framework CrossMiner, which consists of three steps. The first step preprocesses and filters raw user reviews from the three app stores, including Google Play, App Store, and Windows Store (Section IV-A). In this process, raw user reviews are converted into clean user reviews to facilitate the following steps. The second step trains a model for our dataset. This model can extract the keywords automatically for the seven issues illustrated

TABLE I: User Review Related to Each Issue for Facebook

Issue	Platform	Review	Rating	Date
Battery	Google Play	This app is the main reason to drain down the battery!	1.0	Feb 07, 2016
	App Store	Nice but make wasteful battery, first fix dong.	4.0	Mar 05, 2016
	Windows Store	Battery draining app.	1.0	Mar 11, 2016
Crash	Google Play	The app crash as soon as i tap on the facebook icon.	1.0	Feb 08, 2016
	App Store	Crash and hang issue in ios.... Pls fix.	1.0	Mar 08, 2016
	Windows Store	Turrible, it crashes every 4 minutes and its just.	1.0	Mar 04, 2016
Memory	Google Play	The app is good but it takes too much memory space.	4.0	Jan 18, 2016
	App Store	Since the last update covers much memory space.	3.0	Mar 05, 2016
	Windows Store	This it takes whole space in my memory card.	1.0	Aug 08, 2015
Network	Google Play	It always gives me a network problem.	1.0	Feb 06, 2016
	App Store	Network connection error.	1.0	Nov 21, 2015
	Windows Store	Waiting for network for days, slowest app ever.	2.0	Sep 23, 2015
Privacy	Google Play	The Big Brother version. No privacy anymore.	1.0	Feb 07, 2016
	App Store	Poor. Privacy invading.	1.0	Jan 27, 2016
	Windows Store	It got privacy problems.	1.0	Mar 07, 2016
Spam	Google Play	Uses too many resources, and includes a lot of spam.	2.0	Aug 05, 2015
	App Store	All this spam and posts I didn't make are annoying.	2.0	Aug 31, 2015
	Windows Store	This app puts spam ads for weight loss on the news feed.	1.0	Feb 19, 2015
UI	Google Play	Change ui of app. its boring to use same ui app.	2.0	Sep 11, 2015
	App Store	This app can be so much better...yet the UI just drives me nuts.	1.0	Feb 05, 2016
	Windows Store	We need the call feature and little tweak in the UI.	1.0	Feb 22, 2016

in Table I (Section IV-B). Based on the extracted keywords, we prioritize these issues for each platform, and compare these issues distributions among the three platforms. To gain an in-depth understanding of specific issues, we also recommend essential user reviews corresponding to these issues. Finally, we visualize the experimental findings for app developers (Section V).

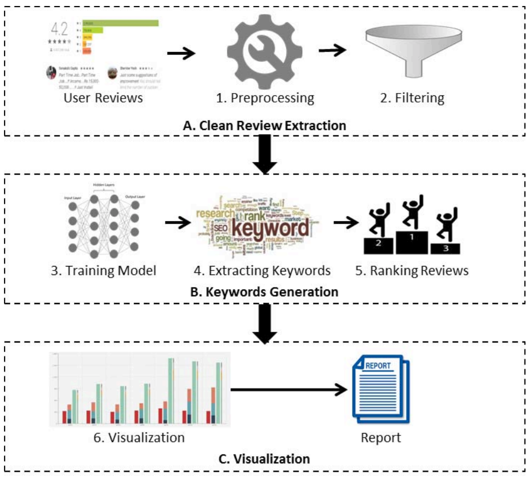


Fig. 2: Overview of the Framework CrossMiner

IV. Implementation

In this section, we present the details of our framework CrossMiner, including clean review extraction, and keywords generation. The visualization component will be discussed during the experimental study (Section V).

A. Clean Review Extraction

To our knowledge, app reviews are short in length, and contain massive misspelled words and made-up words [21]. In the first step, CrossMiner obtains clean reviews to facilitate the model training in the subsequent process. The step mainly contains two parts, *i.e.*, preprocessing and filtering.

1) Preprocessing: To facilitate the subsequent analysis, we first remove the non-English characters existing in the raw reviews and convert raw reviews into lowercases. Then we remove all the non-alpha-numeric symbols but keep the punctuations to ensure the semantic integrity [3]. Finally, we tokenize the reviews to word-level collections. To better reduce the inflectional forms to the common base forms, we propose a novel lemmatization method. We do not utilize stemming, since it usually refers to a crude heuristic process that chops off the ends of words, illustrated in Table II. Some words are difficult to identify after stemming (*e.g.*, "minutes" to "minut", and "adding" to "ad").

TABLE II: Results of Stemming and Lemmatization

Original Word	Stemming	Lemmatization(v)	Lemmatization(n)
are	be	be	are
adding	ad	add	adding
several	sever	several	several
settings	set	settings	setting
developers	develop	developers	developer
minutes	minut	minutes	minute
serves	serv	serve	serf
does	doe	do	doe
uses	use	use	us
pass	pass	pass	pas
less	less	less	le

Furthermore, considering the influence of the part of speech, we combine the lemmatization for verbs, denoted as Lemmatization(v), and the lemmatization for nouns, denoted as

Lemmatization(n). Lemmatization(n) can not convert verbs into the base forms. Moreover, some words are converted into other words that are totally irrelevant with original words, such as "serves" to "serf", "does" to "doe", "uses" to "us", which can be compensated by Lemmatization(v). However, Lemmatization(v) can not achieve the desired result either. For example, "settings" and "developers" keep unchanged after the lemmatization(v), while Lemmatization(n) can return the correct forms. Therefore, neither lemmatizations can achieve ideal results solely.

The combinations of Lemmatization(v) with Lemmatization(n) are implemented as following. We first lemmatize all words by Lemmatization(v). We then conduct the Lemmatization(n) for words without "ss" ends, since Lemmatization(n) converts the words ending with 'ss' into other words instead of their base forms (*e.g.*, "pass" to "pas", and "less" to "le", illustrated in Table II). Table III presents the results of our proposed lemmatization method, which demonstrates its effectiveness. The Lemmatizer employed is implemented based on the Natural Language Toolkit (NLTK) [9].

TABLE III: Results of Proposed Lemmatization

Original Word	Proposed Lemmatization
adding	add
several	several
settings	setting
developers	developer
minutes	minute
serves	serve
does	do
uses	use
pass	pass
less	less

2) Filtering: The previous step generates a preprocessed review collection, with examples presented in Table IV. We then classify each review into three types, *i.e.*, "useless", "non-informative", and "informative". The "useless" reviews are those reviews with too much made-up or misspelled words. Some users type letters just arbitrarily during the review writing, which cannot provide any suggestions to developers. The "non-informative" reviews contain no information beneficial for the app development (*e.g.*, "nice app.", and "pls fix it!"). We retain the "non-informative" reviews since they possess intact sentence structures, which can serve as the input of the model training. All the other reviews are determined as "informative" reviews, which offer developers suggestions on fixing bugs or adding features. In the end, only the "useless" reviews are filtered out for the subsequent process.

Subsequently, to filter noises in the reviews, we conduct a rule-based method in the **word-level** granularity and spell checking at the **review-level** granularity.

a) Word-Level: Three rules are adopted during the word-level filtering process, illustrated in the following.

Rule 1 (Consecutive Duplicate Letter Limit). *We remove consecutive duplicates, since the length of consecutive repeated letters is less than three generally [4]. Specifically, if the repetition times of a letter is more than two, the repeated ones will be eliminated (e.g., "suuuuper" to "super").*

Rule 2 (Word Length Limit). *We remove all the words whose length is more than 15, since 99.93% English words' lengths are less than 16 [6] (e.g., "jfieendkwjjfkkdn").*

Rule 3 (Consecutive Duplicate Word Limit). *We remove consecutive duplicate words in a sentence (e.g., "very very very beautiful" to "very beautiful").*

b) Review-Level: In review level, we employ Enchant [5], a generic spell checking library, to conduct the spell checking in each review. Any reviews with more than half words not correctly spelled will be removed.

After preprocessing and filtering, we convert all raw reviews into clean reviews. Table IV presents the results after preprocessing and filtering.

TABLE IV: Results of Preprocessing and Filtering

Type	Preprocessed Review	Clean Review
Useless	gk bgitu jlek anyway. tp gmna lg mw. hrus ttap there perubhan.	
Non-informative	nice app.	nice app.
Non-informative	pls fix it!	pls fix it!
Non-informative	it be suuuuper.	it be super.
Non-informative	jfieendkwjjfkkdn i dont know what to say its awsone.	i dont know what to say its awsone.
Non-informative	very very very beautiful.	very beautiful.
Informative	it be so slow and it glitch up.	it be so slow and it glitch up.

B. Keywords Generation

We have obtained clean reviews based on preprocessing and filtering in Section IV-A. In this section, we train the model for our dataset (Section IV-B1), from which the keywords are then generated with respect to the seven issues (Section IV-B2). Finally, reviews for each issue are prioritized according to its importance and usefulness to developers (Section IV-B3).

1) Training Model: To establish the model, we first convert all words to vectors by employing word2vec [33], a neural network implementation for learning vector representations of words. Single sentence serves as the input of word2vec, generally represented by a list of words. Since reviews may consist of several sentences, we demand to split the reviews into sentences. Here, NLTK's punkt tokenizer [12] is employed for the splitting. Based on the obtained parsed sentences, we then adopt skip-gram, one flavor of word2vec, as our training model.

Given a sequence of words to train $\{w_1, w_2, w_3, ..., w_T\}$, the training objective of skip-gram is to learn word vector representations that are good at predicting the nearby words. The objective function of skip-gram model is to maximize the log probability of any context word given the current center word, defined as

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log p(w_{t+j}|w_t), \quad (1)$$

where T is the number of training words, m is the size of training context, and θ represents all variables to be optimized. The log probability $logp(w_{t+j}|w_t)$ can be trained by hierarchical softmax or negative sampling. We leverage the hierarchical softmax as the training algorithm, since it achieves better performance for infrequent words.

2) Extracting Keywords: Based on the training model, we attain the vector representation of each word in the clean reviews. For the seven user-concerned issues (*i.e.*, "crash", "battery", "memory", "network", "privacy", "spam", and "UI"), we capture 21 most related words to each issue based on the cosine similarity. Given two vectors of the app issue I and the examining word W, the cosine similarity is determined as

$$similarity = \frac{\sum_{i=1}^{n} I_i W_i}{\sqrt{\sum_{i=1}^{n} I_i^2}\sqrt{\sum_{i=1}^{n} W_i^2}}, \tag{2}$$

where I_i and W_i are the ith components of the vectors I and W, respectively.

By employing the cosine similarity method, we obtain 21 most similar words corresponding to each issue. Instead of regarding these words as keywords directly, we remove the stop words which occur frequently but carry fewer meanings in the reviews, *e.g.*, "a", "is", and "the". We eliminate the stop words provided by the NLTK [9] corpus. Moreover, we remove these words that appear close to the app issue in distance but not semantically related actually. To achieve this, we employ k-means algorithm to cluster all words into groups. Thus, these words in the same group are consistent semantically in theory.

Table V shows similar words and keywords of "battery". Several similar words are removed from the keywords list because they are not in the same cluster with battery, including "cpu", "ram", "deplete", "memory", and "foreground". After getting keywords of each issue, we prioritize issues in different platforms by a keyword-based method.

TABLE V: Similar Words and Keywords of **Battery**

Issue	Similar Words	Keywords
Battery	battery, drain, usage, consumption, overheat, drainer, consume, cpu, power, ram, hog, electricity, drainage, charger, batter, standby, discharge, energy, deplete, memory, foreground	battery, drain, usage, consumption, overheat, drainer, consume, ~~cpu~~, power, ~~ram~~, hog, electricity, drainage, charger, batter, standby, discharge, energy, ~~deplete~~, ~~memory~~, ~~foreground~~

3) Ranking Reviews: To help developers understand one specific issue deeply, we also prioritize raw user reviews regarding the issue according to their importance and usefulness for app developers. We consider one review related to the issue, if the review comprises the corresponding keywords. For all the related reviews, we rank their importance based on the lengths and ratings. Generally, reviews with lower ratings and longer lengths are preferred by developers, since they tend to express the app bugs or the necessary features [23]. The ranking score $score(t)$ for the issue t is defined as follows.

$$score(t) = e^{-r(\frac{1}{ln(h)+1}+\frac{1}{ln(n_t)+1})}, \tag{3}$$

where n_t indicates the number of keywords for the issue t, r denotes the user rating, and h represents the review length. The definition ensures the ranking score to be ranged from 0 to 1. Finally, the reviews with lower ratings and longer lengths are prioritized.

V. Experimental Study

In this section, we present the experimental results of CrossMiner. To verify that CrossMiner can really help developers, we conduct several experiments and case studies.

A. Dataset

Our dataset has been collected from AppFigures [2], a website providing API to crawl user reviews in multiple app stores, including Google Play, App Store, Windows Store, etc. Our dataset contains 4,663,316 reviews posted by users between September, 2014 and March, 2016. 20 popular apps belonging to 8 categories are studied. Specifically, our dataset comprises 2,637,438 reviews from Google Play, 1,687,003 reviews from App Store, and 338,875 reviews from Windows Store, which are large enough for review analysis [21]. Table VI lists the details of our dataset.

TABLE VI: Review Dataset of 20 Subject Apps

Category	App Name	Google Play	App Store	Windows Store
Communication	LINE	102,155	104,960	9,511
	Messenger	244,516	234,400	15,801
	Skype	186,868	8,834	35,355
	Viber	161,833	109,710	21,569
	WeChat	89,205	204,922	9,508
	WhatsApp	241,792	85,117	25,130
Education	Duolingo	65,632	59,659	12,365
	TED	778	905	380
Entertainment	Netflix	97,503	45,383	28,846
	Spotify Music	178,477	249,212	33,143
	VLC	3,725	771	4,674
	YouTube	69,300	210,371	13,404
Photography	Camera360	122,350	51,777	2,319
Productivity	Evernote	65,540	30,795	2,308
Shopping	eBay	142,129	20,000	4,485
Social	Facebook	244,897	232,347	51,040
	Instagram	249,132	13,741	55,596
	Tango	122,638	200	53
	Twitter	246,546	23,200	13,218
Transportation	HERE	2,422	699	170
Total Reviews		2,637,438	1,687,003	338,875

B. Performance Metrics

To measure the performance of the issue prioritizing results based on CrossMiner, we adopt the well-known Normalized Discounted Cumulative Gain (NDCG) in the following [21]:

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \tag{4}$$

where $NDCG@k \in [0, 1]$, with 1 representing the ideal rank order. The higher value indicates the predicted rank order is closer to the ideal rank order.

C. Keywords Generation Results

We train the word2vec model by all the clean reviews, 3,113,111 reviews totally. As for the parameter settings, we set the word vector dimensionality as 300, the context size as 10, and the minimum word count as 80 empirically. The model training process takes several minutes to tens of minutes depending on the vocabulary size. Ultimately, we obtain the word2vec model of our dataset. Each word in the dataset is represented by a 300-dimension vector.

Next, we extract the keywords for each issue based on the keywords generation method introduced in Section IV-B. Table VII depicts the relevant keywords corresponding to the seven issues.

TABLE VII: Keywords of Seven Issues

Issue	Keywords
Battery	battery, drain, usage, consumption, overheat, drainer, consume, power, hog, electricity, drainage, charger, batter, standby, discharge, energy
Crash	crash, freeze, foreclose, lag, crush, stall, close, shut, laggy, glitch, hang, load, stuck, startup, buffer, open, laggs, freez, glitchy, buggy
Memory	memory, storage, space, gb, internal, gigabyte, ram, 6gb, occupy, 4gb, mb, 300mb, 8gb, 500mb, 16gb, byte, 5gb, gig, 2gb, 1gb, 1g
Network	network, connectivity, internet, consumption, wifi, connection, reception, conection, connect, signal, 4g, wi, 3g, broadband, fibre, lte, reconnecting, fi, wireless, reconnect, disconnect
Privacy	privacy, security, invade, safety, personal, policy, invasion, breach, protection, protect, private, disclosure, secure, unsafe, insecure, permission, fingerprint, encryption, violation, encrypt
Spam	spam, spammer, scammer, unsolicited, harassment, unwanted, bot, bombard, junk, scam, advertisement, popups, scraper, hacker
UI	ui, interface, design, layout, gui, ux, clunky, redesign, aesthetic, navigation, usability, desing, sleek, appearance, aesthetically, intuitive, minimalistic, ugly, slick, graphic, unintuitive

Compared to the traditional method [30], which is selecting keywords manually for each issue, our keywords generation method has these following advantages. First, CrossMiner can automatically generate the keywords for each issue, which is more time-saving and more efficient. In contrast, manually selecting the keywords could be laborious. Second, CrossMiner can extract misspelled and made-up words that are related to the issue, which are generally ignored during the manual process. As illustrated in Table VII, CrossMiner specifies "conection" as a keyword for the "network" issue, although it is a misspelled word of "connection". Moreover, among the keywords of the "memory" issue, made-up words (*e.g.*, "6gb", "300mb", etc.) are utilized to discuss the issue. In summary, we present an automatic and effective keywords generation method.

D. Can CrossMiner Prioritize App Issues Between Different Platforms?

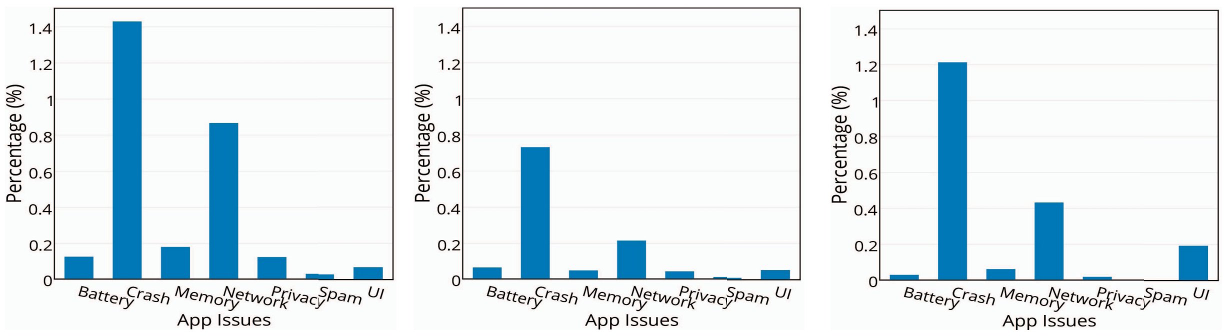
To answer this question, we conduct experiments on the 20 apps (listed in Table VI). The issue distributions for the apps are described in Table XI. In this section, we employ two apps - Spotify Music and eBay for illustration. In our dataset, Spotify Music has 178,477 reviews from Google Play, 249,212 reviews from App Store, and 33,143 reviews from Windows Store, while eBay has 142,129 reviews, 20,000 reviews, and 4,485 reviews from these three app stores, respectively. To reduce the influence of "useless" reviews defined in Section IV-A2, we only analyze the "informative" and "non-informative" reviews of the two apps. We determine whether a user indeed complains about a certain issue in his/her review based on two requirements: 1) The review must contain at least one of the keywords for the corresponding issue; 2) The rating of the review must be less than three stars to ensure that the reviews are expressing complaints. The experimental results of Spotify Music and eBay are discussed in the following.

1) Case Study on Spotify Music: We focus on studying Spotify Music in this part. After preprocessing, Spotify Music has 154,550 clean reviews from Google Play, 217,535 clean reviews from App Store, and 25,480 clean reviews from Windows Store. The issue distributions are illustrated in Fig. 3 and Fig. 4, visualizing issue percentages and corresponding average ratings, respectively.

Results: Fig. 3 presents the percentage distribution on the seven issues for Spotify Music. We discover that the "crash", "network" and "memory" issues are the primary concerns of Android users, accounting for 1.429%, 0.867%, and 0.181%, respectively. For the iOS users, they are more concerned about issues related to "crash" (0.732%), "network" (0.215%), and "battery" (0.061%). Among Windows Phone users, they complain more about the "crash", "network", and "UI" issues, occupying 1.213%, 0.432%, and 0.192%, respectively.

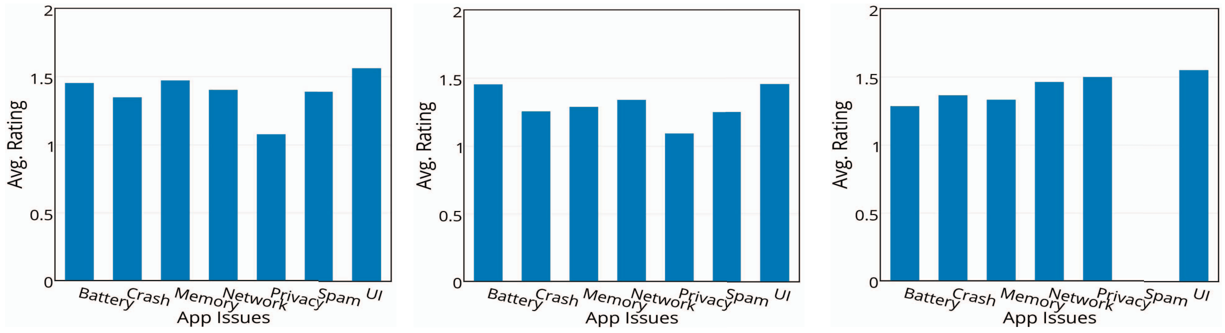
Fig. 4 depicts the rating distribution of the seven issues. We identify that the "privacy" (1.08), "crash" (1.35), and "spam" (1.39) issues represent lower ratings than other issues in Google Play. Similarly, in App Store, the three issues also correspond to the lowest ratings, which are 1.09, 1.25, and 1.26, respectively. However, in Windows Store, the issues with the lowest ratings become related to "battery", "memory", and "crash", with average ratings at 1.29, 1.33, and 1.37, respectively.

Discussion: As Fig. 3 illustrates, Spotify Music users of the three platforms concern more about issues relevant to "crash" and "network". Frequently crash can definitely destroy users' perceptions and generate unfavorable reviews. Regarding the "network" issues, since Spotify Music is a music streaming app that provides digital music service, users may feel uncomfortable or annoyed if the music downloading is too slow or consumes too much traffic. Besides the "crash" and "network" issues, for the Android platform, users also complain about "memory" (0.181%), "battery" (0.127%), "privacy" (0.125%), "UI" (0.067%), and "spam" (0.027%). With respect to the iOS platform, 0.061% users convey dissatisfaction with "battery", with other issues "UI", "memory", "privacy", and "spam" accounting for 0.052%, 0.045%, 0.04%, and 0.009%, respectively. For the Windows Phone platform, "UI" (0.192%) are more concerned by users, followed by



(a) Google Play (b) App Store (c) Windows Store

Fig. 3: Percentage Distribution on Issues of Spotify Music



(a) Google Play (b) App Store (c) Windows Store

Fig. 4: Rating Distribution on Issues of Spotify Music

"memory" (0.059%), "battery" (0.125%), "privacy" (0.067%), and "spam" (0.027%).

Overall, the iOS version seems to outperform the Android version and Windows Phone version. For example, the percentages of the "crash" and "network" issues for the iOS version are lower than those of the other versions. To verify the fact, we also calculate the average ratings across the three platforms. We discover that the iOS version receives the highest ratings (4.48), with the Windows Phone version (4.1) followed after. The Android version only receives 3.90 stars.

Therefore, we suggest that the Spotify Music developers should focus on testing issues related to "crash" and "network" on the three platforms, especially weighting more on the Android version. Moreover, the developers should also design comprehensive testing cases for the "memory" for the Android version, "battery" for the iOS version, and "UI" for the Windows Phone version.

To help developers gain an in-depth understanding about one specific issue, we prioritize reviews associated with the issue based on the method introduced in Section IV-B3. Table VIII illustrates the top three reviews related to the "UI" issue in Google Play. As the Table shown, all the reviews complain about some aspects of "UI" (e.g., "no way to go back" in review 1, "missing basic and obvious music player features" in review 2, and "playing view the artwork is smaller to fit in the artwork on either side" in review 3). Thus, developers can schedule the app modification based on the prioritized reviews.

TABLE VIII: Top Three Reviews Related to "UI" of Spotify Music in Google Play

Rank	User Review	Score
1	Seriously bad user experience and interface. Once you've liked or unliked a song, there's no way to go back even if you've made a mistake. I don't know why Spotify is so popular with suck poor graphic design.	0.943
2	Clunky unintuitive interface missing basic and obvious music player features. You must get the basics right first before trying to push rubbish the user doesn't want.	0.914
3	Don't like the new design, in the now playing view the artwork is smaller to fit in the artwork on either side. I don't care what's on either end of my current playing track, or at least show it in a way that doesn't take up artwork space. The album art is always an awesome part of the music's personality so it shouldn't be minimised like this. Also the now playing bar at the bottom of the screen isn't flat looking, looks like design from windows XP. Not happy. An awesome service needs an awesome interface.	0.890
...	...	...

2) Case Study on eBay: We focus on analyzing the experimental results of eBay in this part. After preprocessing, the shopping app eBay possesses 122,977 clean reviews from Google Play, 19,192 clean reviews from App Store, and 4,207 clean reviews from Windows Store. The percentage and rating distributions on the seven issues are illustrated in Fig. 5 and Fig. 6, respectively.

Results: Fig. 5 describes the issue percentage distribution of eBay. As the figure illustrates, Android users are most concerned about issues related to "network", "crash", and "UI", accounting for 2.737%, 1.586%, and 0.644%, respectively. While iOS users complain more about "crash", "UI", and "network", with percentages 2.626%, 1.443%, and 0.245%, respectively. With regard to the Window Phone platform, the users also care about the "crash" (1.925%), "UI" (0.594%), and "network" (0.285%), similar to the iOS users.

Fig. 6 depicts the rating distribution on the seven issues. In Google Play, the "spam", "privacy", and "network" issues correspond to the lowest ratings than the others, scored at 1.11, 1.13, and 1.177, respectively. For App Store, the three issues with poorest ratings are "spam" (1.043), "battery" (1.1), and "privacy" (1.143). Regarding the Windows Store, the poorly rated issues are "battery" (1.0), "privacy" (1.0), and "spam" (1.0).

Discussion: As Fig. 5 illustrates, users of different platforms all complain more about "crash", "network" and "UI". Differences also exist across the platforms. For example, in Google Play, 1.586% users express about the "network" issue, much more than other two platforms. Therefore, the eBay developers are suggested to focus on testing the "network" issue for the Android version. As Fig. 6 depicts, users tend to give poor ratings to the "privacy" and "spam" issues. In comparison, the Window Phone users are more critical, since they all rate with the lowest ratings (1.0) for these two issues.

Summary: By these two app-level case studies, we discover that users of different platforms indeed concern about different issues of an app. CrossMiner automatically prioritizes the user-concerned issues. Developers can arrange and design the testing cases for the important issues on each platform. Moreover, based on the case studies, we also identify some similarities across the platforms. For example, users generally concern more about "crash" and "network" issues.

E. Evaluation of Prioritizing Performance

In this section, we aim at evaluating the issue prioritization of CrossMiner. If the prioritized issues are consistent with the practical user concerns, the performance can be verified.

We employ Spotify Music for the performance verification, and the official user forums as the groundtruth [14]. An issue with more user views indicates that the issue is more concerned by users. Thus, we obtain the ranks of the seven issues, with an example of Android forums illustrated in Table IX.

Similarly, we capture the issue rankings from the official iOS community and Window Phone community. For the iOS version, the ranked issues are crash(53683), memory(10797), UI(8439), battery(6174), connection(4767), spam(1903), and privacy(1558). Regarding the Windows Phone version, the issue order is crash(4097), connection(1362), battery(300), UI(282), memory(229), spam(94), and privacy(76). We then compare the prioritization results attained by CrossMiner with the groundtruth for these three platforms. The NDCG@7 scores introduced in Section V-B are utilized for the measurement, with results described in Table X.

TABLE IX: Ranked Issues from Android Community of Spotify Music

Rank	Views	User Feedback	Issue
1	56416	No internet connection available	Network
2	32495	No SD Card storage !!	Memory
3	24797	Spotify for Android causing massive battery drain and heating of phone	Battery
4	11315	Spotify crashes on Android	Crash
5	1796	Issues with Android UI context menu touch area	UI
6	197	Intrusive or what!!!!!!	Privacy
7	80	Tired of the push notification spam!	Spam

TABLE X: Prioritizing Results

	Android	iOS	Windows Phone
NDGC@7	0.943	0.911	0.982

By examining the results, we discover that CrossMiner achieves 0.943, 0.911, and 0.982, in terms of NDGC@7 for the Android, iOS, and Windows Phone versions, respectively. The average accuracy arrives at 0.945, which indicates that CrossMiner prioritizes issues effectively and reflects the user concerns accurately.

F. Can Cross-Miner Give Platform-level Advice to Develpoers?

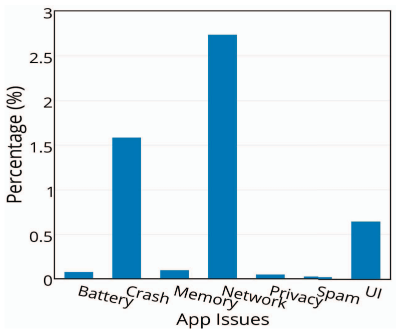
In this section, we aim at exploring the platform-level issues. Fig. 7 illustrates the issue distributions with respect to the 20 subject apps in Google Play, iOS, and Windows Phone.

Results: As Fig. 7 depicts, each bar in the graph represents the percentage of an issue. We discover that the top three issues Android users complain most about are "crash" (1.76%), "network" (0.85%), and "memory" (0.31%). Similarly, the iOS users also express more about "crash" (4.48%), "network" (1.05%), and "memory" (0.48%). For the Window Phone users, they are more concerned about "crash" (2.76%), "network" (0.66%), and "battery" (0.38%).

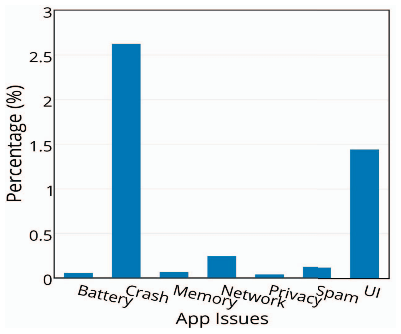
Discussion: We identify that users for all the platforms concern more about the "crash" and "network" issues. So developers should spend more time on these two issues during app development. Moreover, in Khalid *et al.*'s study [29], "app crashing" and "network problem" are ranked at the third and fourth position among all the most frequent complaints list (the top two complaints are "functional error" and "feature request", which are excluded from our study), which is compatible with our findings.

G. Parameter Study

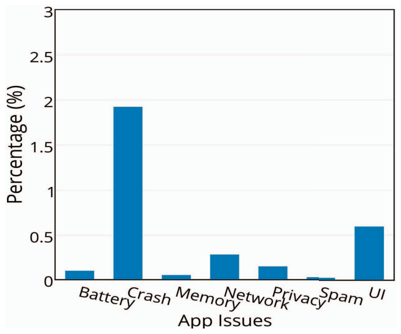
In our framework, one key problem is to set the number of the similar words n for each issue. To obtain an optimal solution, we conduct an experimental study on the parameter



(a) Google Play

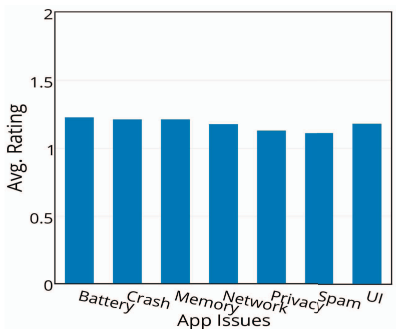


(b) App Store

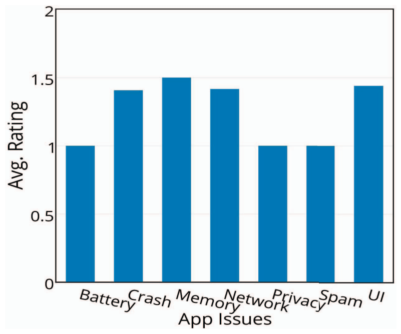


(c) Windows Store

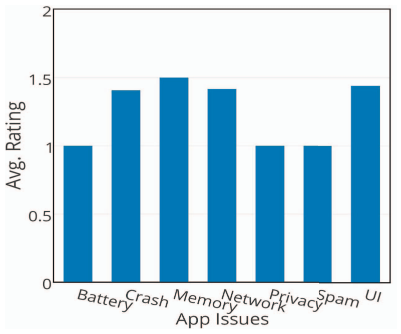
Fig. 5: Percentage Distribution on Issues of eBay



(a) Google Play



(b) App Store



(c) Windows Store

Fig. 6: Rating Distribution on Issues of eBay

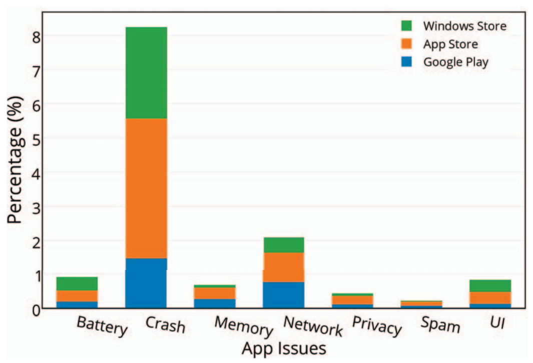


Fig. 7: Average Percentage Distributions on Issues for Different Platforms

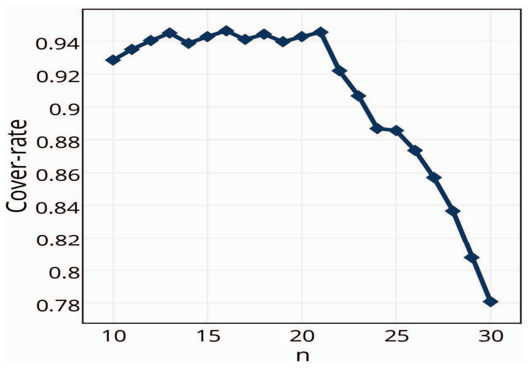


Fig. 8: Influence of n on Cover-rate

settings. We first determine n for the similar words extraction, and obtain the ultimate keywords corresponding to each issue based on Section IV-B. We then define "Cover-rate" to compute the ratio of the number of keywords to the number of similar words extracted in the first step. Smaller Cover-rate indicates that the similar words comprise more unrelated words to the issue. Fig. 8 depicts the Cover-rate along with the number of similar words n.

Fig. 8 illustrates that fewer similar words correspond to relatively high Cover-ratio. However, some keywords will be missed if n is set too small. On the other hand, larger numbers of similar words can cover most keywords, but also carrying with more unrelated words. Therefore, we set the number of the similar words n to be 21 due to the higher performance.

VI. Threats to Validity

In this section, we discuss the threats to validity of our work and talk about the steps we take to mitigate these threats.

First of all, the percentages in the results are so low and the small differences of the results might not be practically significant. Two reasons account for the low percentages: 1)

We remove the "useless" reviews but keep the other two type reviews (*i.e.*, "non-informative" reviews and "informative" reviews). To our knowledge, the "non-informative" reviews account for a large proportion of all reviews which lower the percentages. 2) The 20 apps in our dataset are popular among three platforms. The average rating is about 3.5 star which means the most of reviews are positive (more than 2 star). And we only regard negative reviews as user complaints. Moreover, each app has more than 200 thousand user reviews in our dataset. With massive user reviews, even small differences can reveal the prioritization of user concerns for app issues. Furthermore, we use analysis of variance (ANOVA) [1], a statistical hypothesis test for significance analysis, to check if the cross-platform differences are significantly different. Among all the 20 apps in Table XI, the average p-value of ANOVA is 0.012 which is close to 0.01. Through the ANOVA, we verify that the cross-platform differences are indeed significantly different.

Second, we just analyze seven issues in the paper, which may not cover all the app issues. And these issues may be not mutually exclusive. However, since our framework can identify the keywords related to the issue effectively, other types of issues can also be analyzed similarly. This also illustrates the scalability and usability of our framework. Moreover, we aim to help developers by prioritizing issues from users perspective instead of source code.

Third, we are uncertain whether our discoveries can really facilitate the app testing process for app developers. Through the experiments, our framework verifies that issues for an app can be distributed differently for different platforms, and it can help developers prioritize which types of issues should be addressed for which platform. Developers can prioritize the testing cases accordingly, which are supposed to improve the efficiency of the testing procedure. Moreover, the prioritized issues are consistent with the issues reflected on the user forums. Therefore, we believe that our framework can facilitate the app development.

VII. Related Work

Our study on understanding app issues for different platforms is inspired by two lines of work, namely, cross-platform learning, and app review analysis.

A. Cross-Platform Learning

Various studies have focused on the similarities and differences among different mobile application platforms. Tor-Morten *et al.* [26] utilized a mobile game app to compare the four platforms (*i.e.*, Android, Windows Phone, iOS, and Firefox OS) in terms of technical functionality, APIs, development effort, development support and deployment to live devices. In [36], kim discovers the differences of development environment for iOS and Android OS. Zinaida *et al.* [20] conduct an online questionnaire to compare users on different platforms based on the demographic differences, security and privacy awareness. In Luo *et al.*'s work, they investigate the security impact of UI-based APIs in the WebView component for Android, iOS, and Windows Phone. Mohd *et al.* [19] examine the security requirements on Android and iOS platforms with respect to the application sandboxing, memory randomization, encryption, data storage format and built-in antivirus. Liu *et al.* [32] explore the Internet streaming access on Android and iOS by analyzing a server-side workload collected from a top mobile streaming service provider. In Zhou *et al.*'s work [37], they identify the different topics and attributes on different platforms (*i.e.*, desktop, Android, and iOS) from bug reports.

B. App Review Analysis

During the app development, the developers have limited contact with potential users [31]. The user reviews capture unique perspectives about the users' perception of the apps [29], and can facilitate the software development. In [25], the authors identify several user-concerned factors from online user reviews. In Platzer's work [35], an automated system is specially designed for user review classification. Iacob *et al.* [27] design "MARA" (Mobile App Review Analyzer) to automatically retrieve app features from user reviews. Similarly, in [22], the users' major concerns and preferences are extracted from user reviews. Gao *et al.* [23] [24] utilize user reviews to observe the app issues over versions. In [28], user reviews are more deeply analyzed to capture the relations between user complaints and low ratings. Kong *et al.* [30] explore the relations between users' reviews and security-related behaviors (*i.e.*, spamming, financial issue, over-privileged permission, and data leakage). The work discovers that user reviews can reflect how users think about the security issues.

VIII. Conclusion

This paper proposes a novel framework named CrossMiner to automatically analyze app issues from user reviews by employing a keyword-based method. We aim at discovering the differences and similarities of issue distributions on three popular app stores, *i.e.*, Google Play, App Store, and Windows Store. Based on the identified issue distributions, app developers can design and arrange the testing cases more efficiently for different platforms. To our best knowledge, CrossMiner is the first work that explores app issues on different platforms from users' perspective. The experimental study also verifies that our framework can reflect the user concerns accurately.

Acknowledgment

The work described in this paper was fully supported by the National Natural Science Foundation of China (Project No. 61332010), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14205214 of the General Research Fund), 2015 Microsoft Research Asia Collaborative Research Program (Project No. FY16-RES-THEME-005), and China National Energy Administration under Grant NY20150301.

References

[1] Analysis of variance. https://en.wikipedia.org/wiki/Analysis_of_variance.

[2] AppFigures. https://appfigures.com/.

[3] Bag of Words Meets Bags of Popcorn. https://www.kaggle.com/c/word2vec-nlp-tutorial.

[4] Consecutive Letters. http://www.fun-with-words.com/word_consecutive_letters.html.

[5] Enchant. http://www.abisource.com/projects/enchant/.

[6] English Letter Frequency Counts. http://norvig.com/mayzner.html.

[7] Introduction to Android. http://developer.android.com/guide/index.html.

[8] iOS vs. Android: Your Best Arguments. http://lifehacker.com/ios-vs-android-your-best-arguments-1334921103.

[9] NLTK. http://www.nltk.org/.

[10] 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/.

[11] Pfeiffer Report 2013. http://www.diffen.com/difference/Android_vs_iOS.

[12] Punkt. http://www.nltk.org/_modules/nltk/tokenize/punkt.html.

[13] Smartphone OS Market Share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[14] Spotify. https://community.spotify.com/. [accessed: 18-Mar-2016].

[15] Ten of the Best Cross-Platform Mobile Development Tools. http://appindex.com/blog/ten-best-cross-platform-development-mobile-enterprises/.

[16] Vector Representations of Words. https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html.

[17] What Is Mobile Testing. https://smartbear.com/learn/software-testing/what-is-mobile-testing/.

[18] What Kind Of Person Prefers An iPhone? http://www.forbes.com/sites/toddhixon/2014/04/10/what-kind-of-person-prefers-an-iphone/#2dc096983e5a.

[19] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan, and N. E. Othman. Comparison between android and ios operating system in terms of security. In *Proceedings of the 8th International Conference on Information Technology in Asia (CITA)*, pages 1–4. IEEE, 2013.

[20] Z. Benenson, F. Gassmann, and L. Reinfelder. Android and ios users' differences concerning security and privacy. In *Proceedings of the CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 817–822. ACM, 2013.

[21] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 767–778. ACM, 2014.

[22] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1276–1284. ACM, 2013.

[23] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu. Paid: Prioritizing app issues for developers by tracking user reviews over versions. In *Proceedings of the 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 35–45. IEEE, 2015.

[24] C. Gao, H. Xu, J. Hu, and Y. Zhou. Ar-tracker: Track the dynamics of mobile apps via user review mining. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 284–290. IEEE, 2015.

[25] J. Gebauer, Y. Tang, and C. Baimai. User requirements of mobile technology: results from a content analysis of user reviews. *Information Systems and e-Business Management*, 6(4):361–384, 2008.

[26] T.-M. Grønli, J. Hansen, G. Ghinea, and M. Younas. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In *Proceedings of the 28th International Conference on Advanced Information Networking and Applications (AINA)*, pages 635–641. IEEE, 2014.

[27] 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)*, pages 41–44. IEEE, 2013.

[28] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan. What do mobile app users complain about? a study on free ios apps. 2014.

[29] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *Software, IEEE*, 32(3):70–77, 2015.

[30] D. Kong, L. Cen, and H. Jin. Autoreb: Automatically understanding the review-to-behavior fidelity in android applications. In *Proceedings of the 22nd SIGSAC Conference on Computer and Communications Security (CCS)*, pages 530–541. ACM, 2015.

[31] S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering*, 41(1):40–64, 2015.

[32] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen. A comparative study of android and ios for accessing internet streaming services. In *Passive and Active Measurement*, pages 104–114. Springer, 2013.

[33] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[34] T.-S. Nguyen, H. W. Lauw, and P. Tsaparas. Review synthesis for micro-review summarization. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (ICDM)*, pages 169–178. ACM, 2015.

[35] E. Platzer. Opportunities of automated motive-based user review analysis in the context of mobile app acceptance. In *Proceedings of the CECIIS 2011*, pages 309–316, 2011.

[36] K. W. Tracy. Mobile application development experiences on apple?? s ios and android os. *Potentials, IEEE*, 31(4):30–34, 2012.

[37] B. Zhou, I. Neamtiu, and R. Gupta. A cross-platform analysis of bugs and bug-fixing in open source projects: desktop vs. android vs. ios. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 7. ACM, 2015.

TABLE XI: Issue Distributions for the 20 Apps

App Name	Platform	Battery	Crash	Memory	Network	Privacy	Spam	UI
Camera360	Google Play	**0.202%**	0.664%	**0.277%**	**0.048%**	**0.033%**	**0.008%**	0.058%
	App Store	0.032%	**1.419%**	0.089%	0.024%	0.008%	0.0%	0.041%
	Windows Store	0.12%	0.598%	0.0%	0.0%	0.0%	0.0%	**0.06%**
Duolingo	Google Play	**0.003%**	0.064%	**0.007%**	0.044%	0.0%	0.0%	0.0%
	App Store	0.0%	0.069%	0.006%	0.061%	0.0%	**0.003%**	**0.026%**
	Windows Store	0.0%	**0.467%**	0.0%	**0.308%**	0.0%	0.0%	0.0%
eBay	Google Play	0.072%	1.586%	**0.092%**	**2.737%**	0.044%	0.022%	0.644%
	App Store	0.052%	**2.626%**	0.063%	0.245%	0.036%	**0.12%**	**1.443%**
	Windows Store	**0.095%**	1.925%	0.048%	0.285%	**0.143%**	0.024%	0.594%
Evernote	Google Play	**0.071%**	0.206%	**0.177%**	0.102%	0.059%	**0.076%**	0.087%
	App Store	0.017%	**3.253%**	0.084%	**0.226%**	**0.077%**	0.02%	0.259%
	Windows Store	0.051%	2.092%	0.153%	0.153%	0.051%	0.051%	**0.459%**
Facebook	Google Play	0.594%	5.71%	**1.301%**	0.867%	0.231%	0.057%	0.066%
	App Store	**1.843%**	**11.923%**	0.661%	**1.297%**	**0.416%**	**0.097%**	0.122%
	Windows Store	0.225%	5.647%	0.144%	1.054%	0.262%	0.069%	**0.398%**
HERE	Google Play	0.288%	1.248%	**0.24%**	**0.672%**	0.096%	0.0%	0.24%
	App Store	**0.796%**	0.955%	0.0%	0.478%	**0.318%**	0.0%	**0.478%**
	Windows Store	0.694%	**2.778%**	0.0%	0.0%	0.0%	0.0%	0.0%
Instagram	Google Play	0.054%	1.456%	0.05%	**0.437%**	0.033%	0.09%	0.02%
	App Store	0.018%	**1.89%**	**0.073%**	0.336%	**0.073%**	**0.109%**	0.009%
	Windows Store	**0.068%**	0.996%	0.015%	0.367%	0.007%	0.011%	**0.063%**
LINE	Google Play	0.144%	1.554%	**0.275%**	1.048%	0.09%	0.21%	0.042%
	App Store	**0.246%**	**8.623%**	0.22%	**1.6%**	**0.246%**	**0.273%**	**0.396%**
	Windows Store	0.124%	1.751%	0.11%	0.386%	0.014%	0.083%	0.331%
Messenger	Google Play	0.868%	2.19%	**1.02%**	0.99%	0.531%	0.017%	0.042%
	App Store	0.565%	**4.3%**	0.932%	1.048%	**1.996%**	**0.109%**	0.069%
	Windows Store	**4.71%**	2.418%	0.237%	**1.323%**	0.628%	0.03%	**0.266%**
Netflix	Google Play	0.05%	1.198%	0.042%	0.949%	0.01%	**0.004%**	0.045%
	App Store	**0.087%**	**6.017%**	**0.083%**	**2.216%**	**0.018%**	0.004%	0.179%
	Windows Store	0.03%	3.8%	0.083%	0.838%	0.011%	0.0%	**0.315%**
Skype	Google Play	0.448%	2.396%	**0.247%**	0.982%	0.077%	0.048%	0.092%
	App Store	**0.824%**	**4.154%**	0.167%	**2.125%**	**0.298%**	**0.251%**	**0.919%**
	Windows Store	0.326%	3.769%	0.082%	0.96%	0.178%	0.049%	0.861%
Spotify	Google Play	**0.127%**	**1.429%**	**0.181%**	**0.867%**	**0.125%**	**0.027%**	0.067%
	App Store	0.061%	0.732%	0.045%	0.215%	0.04%	0.009%	0.052%
	Windows Store	0.027%	1.213%	0.059%	0.432%	0.016%	0.0%	**0.192%**
Tango	Google Play	**0.062%**	**0.18%**	**0.076%**	**0.298%**	0.175%	0.05%	**0.011%**
	App Store	0.0%	0.0%	0.0%	0.0%	**0.556%**	**0.556%**	0.0%
	Windows Store	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
TED	Google Play	0.0%	1.62%	**0.147%**	0.295%	**0.147%**	0.0%	0.295%
	App Store	0.0%	3.385%	0.0%	**0.923%**	0.0%	0.0%	0.154%
	Windows Store	0.0%	**4.237%**	0.0%	0.282%	0.0%	0.0%	**1.13%**
Twitter	Google Play	0.09%	2.19%	0.083%	0.262%	0.065%	0.069%	0.193%
	App Store	0.079%	3.201%	**0.168%**	0.299%	**0.126%**	**0.229%**	0.257%
	Windows Store	**0.12%**	**3.549%**	0.094%	**0.41%**	0.034%	0.077%	**0.65%**
Viber	Google Play	0.179%	0.748%	**0.171%**	**1.125%**	0.069%	0.549%	0.036%
	App Store	0.083%	1.252%	0.051%	0.935%	**0.125%**	**0.583%**	0.025%
	Windows Store	**0.327%**	**1.486%**	0.07%	1.115%	0.013%	0.077%	**0.154%**
VLC	Google Play	0.149%	1.295%	0.149%	0.0%	0.0%	0.0%	0.1%
	App Store	**0.837%**	12.552%	0.279%	**1.395%**	0.0%	0.0%	0.837%
	Windows Store	0.13%	**12.588%**	**0.285%**	0.026%	0.0%	0.0%	**0.856%**
WeChat	Google Play	0.189%	1.007%	**0.426%**	0.612%	0.124%	0.042%	0.108%
	App Store	0.058%	0.499%	0.292%	**0.704%**	**0.241%**	**0.059%**	0.074%
	Windows Store	**0.489%**	**1.363%**	0.103%	0.309%	0.051%	0.026%	**0.283%**
WhatsApp	Google Play	0.064%	0.582%	0.216%	0.288%	0.126%	0.043%	0.085%
	App Store	**0.337%**	**5.657%**	**3.191%**	**1.435%**	**0.366%**	**0.047%**	0.083%
	Windows Store	0.333%	1.386%	0.136%	0.278%	0.115%	0.0%	**0.202%**
YouTube	Google Play	0.133%	2.241%	**0.137%**	**2.447%**	**0.01%**	0.014%	0.238%
	App Store	**0.404%**	**9.155%**	0.114%	2.207%	0.01%	**0.025%**	**1.531%**
	Windows Store	0.06%	1.306%	0.103%	0.275%	0.0%	0.0%	0.232%