PAID: Prioritizing App Issues for Developers by Tracking User Reviews Over Versions

Cuiyun Gao*†, Baoxiang Wang†, Pinjia He†, Jieming Zhu†, Yangfan Zhou*‡, and Michael R. Lyu*†
*Shenzhen Research Institute, The Chinese University of Hong Kong, China
†Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, China
‡School of Computer Science, Fudan University, Shanghai, China
{cygao, bxwang, pjhe, jmzhu, yfzhou, lyu}@cse.cuhk.edu.hk

***Abstract*—User review analysis is critical to the bug-fixing and version-modification process for app developers. Many research efforts have been put to user review mining in discovering app issues, including laggy user interface, high memory overhead, privacy leakage, etc. Existing exploration of app reviews generally depends on static collections. As a result, they largely ignore the fact that user reviews are tightly related to app versions. Furthermore, the previous approaches require a developer to spend much time on filtering out trivial comments and digesting the informative textual data. This would be labor-intensive especially to popular apps with tremendous reviews.**

In the paper, we target at designing a framework in Prioritizing App Issues for Developers (PAID) with minimal manual power and good accuracy. The PAID design is based on the fact that the issues presented in the level of phrase, *i.e.*, a couple of consecutive words, can be more easily understood by developers than in long sentences. Hence, we aim at recommending phrase-level issues of an app to its developers by tracking reviews over the release versions of the app. To assist developers in better comprehending the app issues, PAID employs ThemeRiver to visualize the analytical results to developers. Finally, PAID also allows the developers to check the most related reviews, when they want to obtain a deep insight of a certain issue. In contrast to the traditional evaluation methods such as manual labeling or examining the discussion forum, our experimental study exploits the first-hand information from developers, *i.e.*, app changelogs, to measure the performance of PAID. We analyze millions of user reviews from 18 apps with 117 app versions and the results show that the prioritized issues generated by PAID match the official changelogs with high precision.

I. Introduction

Different from traditional software resources such as codes and documents, user reviews on mobile apps are resources of comments directly from customers and can be exploited by developers during the bug-fixing process. Some previous studies on app reviews have been conducted, such as retrieving app features [21], analyzing user sentiments [14], and comprehending user requirements [29], etc. Most of the work adopts traditional review mining methods, *e.g.*, topic modeling [12] or linguistic rules [32], to analyze user comments.

However, the app reviews are greatly distinct from these conventional ones. Different from the classic online reviews like hotel reviews [30] and commodity reviews [20], app reviews have three major characteristics: 1) They are shorter in length because most of them are posted from mobile terminals; 2) They contain massive non-informative reviews that cannot provide insights to improve the apps (such as "good app", "sucks", and "stupid you NEED to use this", etc.), and practically only 35.1% reviews are "informative" [13]; 3) They are time-sensitive, that is, the significant comments are varying with time. For example, according to the official announcement of WhatsApp, it commits a new version every 3.77 days on average from Nov. 2014 to Jan. 2015 (Fig. 1). The ratings fluctuate with the version updates, indicating that app versions indeed affect the user feedback. There exists some work concentrating on the first two features, but very few studies focus on analyzing the changes of main topics over versions. As far as we know, the only one investigation on this topic is [7]; however, the work focuses on all the apps of Blackberry app store to discover the migration of general app properties (*e.g.*, "Games" shows a tendency to "Sports & Recreation", "Finance" grows close to "Business"). Other than analyzing the trends of app features of the whole app store, in this paper, we aim at identifying crucial app issues for a specific app, so as to help app developers in bug fixing and feature enhancement.

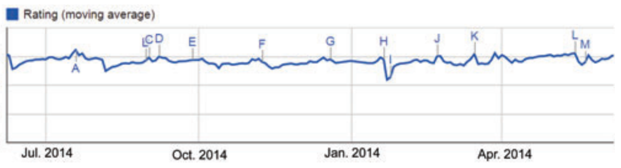
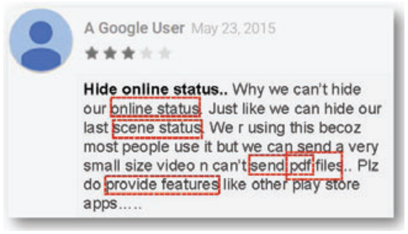


Fig. 1: The Official Changelog of WhatsApp Messenger on Android [2]. The letters indicate main version releases.

Furthermore, the existing exploration consumes a large amount of time on filtering meaningless reviews or interpreting the topics generated by topic modeling manually. This would be quite labor-intensive, especially when we need to handle excessive comments. Despite existing work [13], [15] on automatically extracting user concerns, we pay attention to different aspects. On one hand, current work still demands great human efforts in labeling non-informative reviews, while in our paper, we involve almost no labor power. On the other hand, previous studies do not consider the timeliness of reviews. In contrast, this is one major focus of this work.

In this paper, we propose a novel framework PAID for prioritizing app issues by tracking user comments over release versions. Our goal is to facilitate the process of analyzing reviews for developers while achieving good performance.

We assume that the issues represented in phrase can cost the developers less time than in sentence to recognize the urgent ones. Therefore, we propose to provide app issues in phrase instead of in sentence [13], [15] for viewers. Here, the "phrase" indicates 2-gram terms (*i.e.*, two consecutive words) especially. Figure 2 illustrates an example of the assumption. Intuitively, with key phrases (highlighted in dash rectangles) presented, the developers can learn the main aspects of complaints from users quickly.


A Google User May 23, 2015

★★★☆☆

Hide online status.. Why we can't hide our online status. Just like we can hide our last scene status. We r using this becoz most people use it but we can send a very small size video n can't send pdf files.. Plz do provide features like other play store apps.....

Fig. 2: An Instance of User Reviews with Useful Phrases in Dash Rectangles.

Similar to "*Bag-of-Phrases*", we define a *Phrase Bank*, a dictionary containing all the meaningful phrases in the user reviews. The *Phrase Bank* is established for the subsequent issue-prioritizing step. To exclude the non-informative reviews, we manually label 60 stop words, including common emotional words (such as, "like", "amazing", "cool", etc.) and meaningless description words (such as, "facebook", "star", "app", etc.). This labeling process just takes a couple of minutes, and the generated list of non-informative words can also be applied to the analysis of other apps. To make the prioritized issues more diverse, we adopt topic modeling to group review topics. Each topic is then interpreted by one closest phrase automatically. The produced phrases are the app issues we want to present to developers. For better understanding the textual outputs, we ultimately visualize the results with ThemeRiver [19].

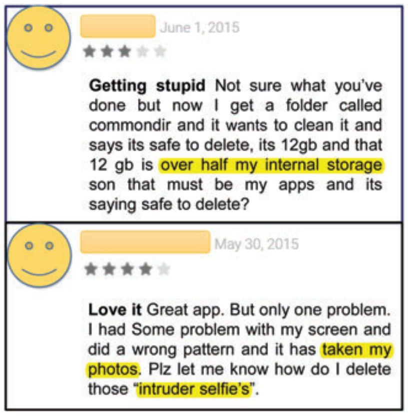
During the evaluation part, we crawl 2,089,737 reviews of 37 apps. Eighteen apps with detailed changelogs (117 versions in total) are employed to test the effectiveness of our framework. We measure the similarities between our results and the official changelogs and find that our method can achieve good precision.

To sum up, we try to answer the following questions.

a) What is the trend of main topics reflected by users along with different app versions? (in V-A)

b) Will the developers modify the most urgent issues generated by our method firstly? (in V-B and V-C)

In general, PAID has several unique features: 1) It concentrates on discovering the critical issues from user reviews just for one specific app, aiming at its reliability aspect; 2) It recommends key phrases instead of long reviews as urgent quality issues to developers; 3) Almost no manpower is involved during the noise-filtering (for weeding out meaningless comments) and topic-extraction process; 4) It tracks user reviews over different app versions rather than based on the whole collection, thus achieving better personalization; 5) The


June 1, 2015

★★★☆☆

Getting stupid Not sure what you've done but now I get a folder called commondir and it wants to clean it and says its safe to delete, its 12gb and that 12 gb is over half my internal storage son that must be my apps and its saying safe to delete?

May 30, 2015

★★★★☆

Love it Great app. But only one problem. I had Some problem with my screen and did a wrong pattern and it has taken my photos. Plz let me know how do I delete those "intruder selfie's".

Fig. 3: Two Review Instances of Viber Which Indicate Two Different Issues of the App.

ultimate app quality issues are well visualized to facilitate the developers' understanding.

The main contributions of this work are summarized as follows:

- We design a novel framework PAID to prioritize phrase-level issues systematically and automatically.
- We propose a new evaluation metric to measure the prioritization of app issues by employing the official app changelogs.
- We implement experiments on a large real dataset to demonstrate the accuracy and effectiveness of our framework.

The remainder of the paper is organized as follows. Section II illustrates the importance of user reviews for app modification by an instance. Section III outlines the overall picture and Section IV explains the issue-prioritizing model of our work. Section V presents our technique dealing with the real app reviews and evaluates our method based on changelogs. Section VI discusses possible limitations. Section VII introduces related work. Finally, Section VIII concludes the paper.

II. A Motivating Example

To address the motivation of the work, we give an example in which the updated app version indeed fixes the issues reflected in user reviews.

We choose Viber (a tool app) for illustration. Viber just updated its version on June 3, 2015 in the Google Play Store. Glancing over its user reviews posted a few days before June 3, we find some comments demanding the same aspects. Figure 3 illustrates a couple of reviews of Viber. In the instances, the users complains the problem of too much internal storage occupied, and the second one describes the existence of "intruder selfie".

By checking the changelog (shown in Fig. 4) of the latest version of Viber, we discover that it covers these two problems with "save space" in the first log and "anti-intruder feature" in the second. It can be seen that user review analysis is a significant part during the app development, and a tool like

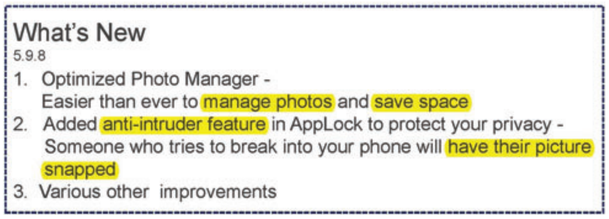

What's New
5.9.8
1. Optimized Photo Manager -
Easier than ever to manage photos and save space
2. Added anti-intruder feature in AppLock to protect your privacy -
Someone who tries to break into your phone will have their picture snapped
3. Various other improvements

Fig. 4: The Official Changelog of the Latest Version of Viber.

PAID can be greatly helpful to app developers, to improve software quality and reliability.

III. Framework of PAID

The overall framework of PAID is shown in Fig. 5, including three procedures (Data Extraction, App Issue Generation, as well as Visualization and Issue Analysis).

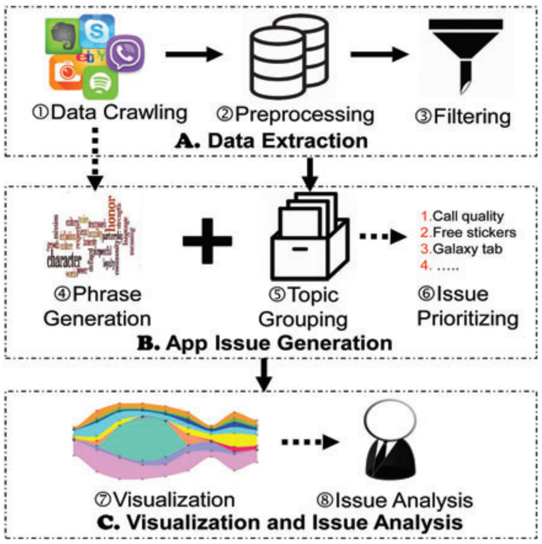


Fig. 5: The Framework of PAID

Data extraction serves to provide a formatted input to the topic grouping process. During the extraction (Section IV-A), we preprocess the raw data crawled from the website and then conduct a filtering process to weed out the noise data or non-informative reviews. Afterwards, we import the filtered data into the issue-generation segment (Section IV-B). App issues are represented in phrase level for developers' convenience. Based on the construction of *Phrase Bank* and topic groups, we rank phrases and display them to developers through visualization (Section IV-C). For the developers who want to gain an in-depth understanding of the issues, we also recommend important user reviews corresponding to these issues.

IV. Design and Implementation

In this section, we describe the design of PAID, including data extraction, app issue generation, as well as visualization and issue analysis.

TABLE I: A Snapshot of the Database

No.	Author	Review	Title	Date	Star	Version
1	Marvin	Not working in Lenovo A606 plus. Fix it.	Help.	2015-05-26T11:00:42	1	5.4.0.3239
2	Sibiya	I love it.	Best app ever.	2015-05-26T10:52:19	5	5.4.0.3239
3	Moham	Nice apps.	Khalil	2015-05-26T10:52:33	3	5.4.0.65524
4	Hassan	Superrrrrrr.	Alcshita	2015-05-26T10:41:01	5	5.4.0.45564
5	Andrew	Would's worst app.	Can't sing till	2015-05-26T10:34:56	1	5.4.0.3239

A. Data Extraction

Data extraction serves to format the raw data and remove the useless information for the subsequent analysis. It includes data crawling, data preprocessing, and filtering.

1) Data Crawling: PAID employs a specific crawling API (provided by AppFigures [3]). It provides access to app repositories, such as reviews, products, ratings, and ranks, etc., along with a variety of querying filters. It takes us a few weeks to collect the review resources. Table I presents a snapshot of the dataset. The dataset contains seven attributes for each item and we use five of them in the paper: No., Review, Date, Star, and Version. The words highlighted are considered *non-informative* in our framework. In our experiments, we have cralwed 2,089,737 user reviews of 37 apps during a period of 10 months (see details in Table VI).

2) Preprocessing: The preprocessing part and filtering part prepare the dataset for the subsequent topic grouping process. As we can observe from Table I, the reviews contain casual (*e.g.*, "superrrrrrr") and non-informative words (*e.g.*, "nice", "love", and "worst"). Firstly, we take the lowercase of all the words in the dataset. To remove those casual words, we use the typical actual wordset - Synsets [9] as the first filter. Then we guarantee that the remaining words exclude the stop words in the NLTK [8] corpus.

Next we reduce the words to the root form by lemmatization [6]. The reason why we do not choose stemming [10] is that stemming crudely chops off the ends of words, which is not suitable for reviews with large numbers of casual words. Meanwhile, lemmatization can preserve the informative derivational ends with the inflectional ends removed. Table II illustrates this fact (*e.g.*, "occasions" is reduced to "occas" in Stemmer and to "occasion" in Lemmatizer). With respect to weeding out the non-informative reviews, we will discuss in detail in Section IV-A3.

TABLE II: Comparison between Stemmer and Lemmatizer

Original Word	Stemmer	Lemmatizer
another	anoth	another
attentions	attent	attention
available	avail	available
compatible	compat	compatible
concentrations	concentr	concentration
occasions	occas	occasion
notifications	notif	notification
solutions	solut	solution

Our goal is to track the user reviews over versions; therefore, we need to divide user reviews into different app versions. As some versions possess insufficient reviews for analysis, we combine the consecutive ones to form a larger review collection.

3) Filtering: The filtering part targets at removing non-informative reviews (*e.g.*, "Nice apps", "I love it", etc., as shown in Table I).

In our framework, we simply remove 60 meaningless words that frequently appear in the non-informative reviews, such as emotional words (*e.g.*, "annoying" and "awesome"), everyday words (*e.g.*, "good" and "much"), etc. We call these words *Filter Bank*. Our prior work [4] shows the effectiveness of this approach. The *Filter Bank* used in the framework is listed in Table III. The words in the *bank* are manually identified. Since the number is rather small, it just takes us a couple of minutes to label them. In contrast, it takes about half an hour for the approach proposed in [13]. The output is fed into the topic grouping process.

TABLE III: *Filter Bank* to Filtering Non-Informative Reviews

app, good, excellent, awesome, please, they, very, too, like, love, nice, yeah, amazing, lovely, perfect, much, bad, best, yup, suck, super, thank, great, really, omg, gud, yes, cool, fine, hello, god, alright, poor, plz, pls, google, facebook, three, ones, one, two, five, four, old, new, asap, version, times, update, star, first, rid, bit, annoying, beautiful, dear, master, evernote, per, line.

B. App Issue Generation

App issue generation aims at recommending the most important app issues to the developers. We employ a rule-based method to produce the *Phrase Bank*. The app issues are prioritized by combining the results of phrase generation and topic grouping.

1) Phrase Generation: The foundation of prioritizing phrase-level app issues is to build a *Phrase Bank* (*i.e.*, a phrase collection). Since the preprocessed review texts have been lemmatized and filtered, the meaning of the phrases generated from these texts may be confusing (*e.g.*, "applic unstabl", "get unlik", etc.). Hence we extract the phrases (specifically referring to 2-gram terms) from the raw user reviews directly instead of the preprocessed reviews by a rule-based method. Four rules are adopted during this process. In the first step, we use TMI (True Mutual Information) [28] to rank the 2-gram phrases. TMI is defined as the weighted average of the pointwise mutual information for all the observed and expected value pairs, indicating the co-occurrence rate of the words in each pair. Intuitively, a meaningful phrase should frequently occur in the collection, in which the words tend to be highly correlated to each other. The TMI between two words w_1 and w_2 is defined as

$$\mathrm{TMI}(w_1, w_2) = \frac{Pr(w_1, w_2)}{Pr(w_1)Pr(w_2)}, \tag{1}$$

where $Pr(w_1, w_2)$ and $Pr(w)$ denote the probability of phrase w_1w_2 and ungram w respectively, and are estimated by their frequencies in the review collections.

Rule 1 (Length Limit)**.** *The length of each word in the phrase must be no less than three.*

Rule 2 (Informative Assurance)**.** *Each word in the phrase should not appear on the stop-word list of NLTK or in the Filter Bank.*

Rule 3 (Part of Speech Limit)**.** *Each word in the phrase should not be an adverb or a determiner.*

Rule 4 (Quality Assurance)**.** *We set a threshold to the probability of $Pr(w_1, w_2)$. The co-occurrence frequency of w_1 and w_2 must exceed five times. Furthermore, we only consider the top 150 phrases based on TMI score. These are to ensure the quality of the phrases in the Phrase Bank.*

Based on the above rules, we obtain the *Phrase Bank*, from which we will recommend important ones to the developers.

2) Topic Grouping: We prioritize phrases in the *Phrase Bank* to the developers by a grouping-based ranking strategy. First, we adopt a topic modeling method to summarize the words into different topics. Then for each group, we pick a phrase to represent the topic, which can be regarded as an *interpretation* process (Section IV-B3).

To capture the timeliness of user reviews, we use dLDA (Dynamic Latent Dirichlet Allocation) [11] to discover the latent topic sets for each version. In dLDA, the reviews in the collection are categorized into discrete topics according to their corresponding versions, and the topics evolve as time goes by. A direct explanation of dLDA model is depicted in Fig. 6, in which the number of app versions is m. For each version, an LDA model [12] is established on the user reviews. The connections between topics of different versions stem from the assumptions of two Dirichlet distributions α and β. Thus we can view the changes of users' attention among versions.

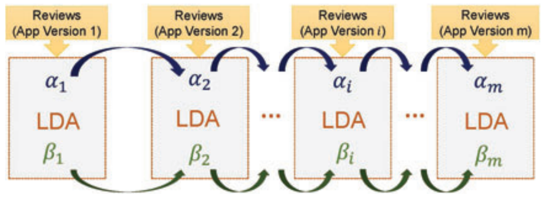


Fig. 6: A Direct Explanation of dLDA Model.

Like LDA, the number of topics in dLDA also needs to be defined in advance. Preprocessed user reviews in Section IV-A are used as the input of topic modeling. For example, if we set the magnitude of topics to be five, one sample output of reviews on Facebook is shown in Table IV. We can observe the transformation of latent topics across different app versions. For example, the top word "video" in Topic 1 disappears during the next few versions while the other words such as "fix" and "close" gradually surface.

3) Issue Prioritizing: From the generated *Phrase Bank*, we select the most suitable one for each topic of dLDA and recommend these phrase-level issues to the developers.

The result of dLDA for a specific app version can be displayed in Table V. Given the collection of user reviews $D = d_1, d_2, \ldots, r_n$ (n is the number of reviews), we denote the corresponding vocabulary $W = w_1, w_2, \ldots, w_g$ (g is the magnitude of the vocabulary). After topic grouping, we have the probability distribution $Pr(w|\beta)$ for each word w over the

TABLE IV: One Sample Output of dLDA on Reviews of Facebook

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
Version 1	**video**	use	download	post	updat
	fix	phone	messag	see	work
	time	need	messeng	feed	new
	play	access	take	recent	friend
	close	make	space	news	slow
Version 2	**video**	use	messeng	post	updat
	fix	phone	download	feed	work
	time	need	messag	news	new
	close	access	take	see	load
	keep	permiss	space	recent	slow
Version 3	**fix**	use	messeng	post	updat
	close	phone	download	feed	work
	time	need	messag	news	load
	keep	access	space	see	new
	open	make	take	recent	slow

topics β. We design a topic-interpretation method from both semantic aspect and sentiment aspect.

TABLE V: Review-Topic Matrix for a Specific App Version

	β_1	β_2	$\cdots$	β_k
w_1	$Pr(w_1\|\beta_1)$	$Pr(w_1\|\beta_2)$	$\cdots$	$Pr(w_1\|\beta_k)$
w_2	$Pr(w_2\|\beta_1)$	$Pr(w_2\|\beta_2)$	$\cdots$	$Pr(w_2\|\beta_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
w_g	$Pr(w_g\|\beta_1)$	$Pr(w_g\|\beta_2)$	$\cdots$	$Pr(w_g\|\beta_k)$

Semantic Aspect: A good semantic phrase for the topic should cover the whole topic as much as possible and discriminate across topics simultaneously. Similar to Mei *et al.*'s work [27], we use KL-Divergence to measure the similarity $Sim(\beta, l)$ between the topic β and the phrase candidate l.

$$\begin{aligned} Sim(\beta, l) &= -KL(Pr(w|\beta)||Pr(w|l)) \\ &\approx \sum_{w} Pr(w|\beta) log(\frac{Pr(w, l|c)}{Pr(w|C)Pr(l|C)}), \end{aligned} \tag{2}$$

where $Sim(\beta, l)$ is the similarity function, $Pr(w|\beta)$ and $Pr(w|l)$ are the probability distributions over word w respectively generated by β and l, and C indicates the whole review collection. We use the occurrence frequency to calculate the probability $Pr(x|C)$ (x denotes l or w).

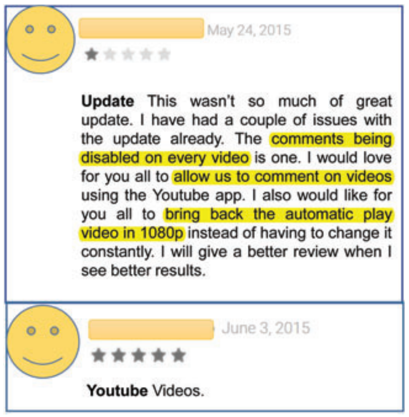
$$Pr(x|C) = \frac{\{d \in C | x\, occurs\, in\, d\}}{\{d \in C\}}, \tag{3}$$

where d represents one user review. The lower the KL-Divergence is, the more the phrase is semantically similar to the topic. Moreover, to make the phrases across topics to be diverse, we add a term to punish those which have a higher similarity score to multiple topics. Thus, the overall similarity function is modified as

$$Sem(\beta_i, l) = Sim(\beta_i, l) - \frac{\mu}{k-1} \sum_{j \neq i} Sim(\beta_j, l), \tag{4}$$

where β_i stands for the current topic to be scored, and μ is used to adjust the penalty due to the similarity to other topics, which is a parameter to be empirically set.

Sentiment Aspect: Intuitively, the developers prefer the review with lower rating and longer length. Fig. 7 reflects this fact. Obviously, the first review provides the developers more information about the app bugs and features, such as enabling the comment functionality and playing video in higher resolution automatically.


May 24, 2015

Update This wasn't so much of great update. I have had a couple of issues with the update already. The comments being disabled on every video is one. I would love for you all to allow us to comment on videos using the Youtube app. I also would like for you all to bring back the automatic play video in 1080p instead of having to change it constantly. I will give a better review when I see better results.

June 3, 2015

Youtube Videos.

Fig. 7: Two Review Instances Showing that Longer-Length and Lower-Rating Reviews are Preferred by the Developers.

For the phrases $l_1, l_2, \ldots, l_z$ (z denotes the number of the phrase candidates) in the *Phrase Bank*, we define their sentiment scores based on user ratings r and review lengths h.

$$Sen(l) = e^{\frac{-r}{ln(h)}}, \tag{5}$$

where r and h of one phrase are defined as the average rating and average length of all the reviews including the phrase, respectively. The phrases contained in the reviews with longer lengths and lower ratings will be scored higher.

Total Score: Combining the semantic aspect with the sentiment aspect by multiplication, the impacts of the both factors on the final score are interrelated. We select the phrase l with the highest score $S(l)$ to represent the topic β_i. The phrases are the app issues we will recommend to the developers for bug fixing or feature improving.

$$S(\beta_i, l) = Sem(\beta_i, l) * Sen(l). \tag{6}$$

C. *Visualization and Issue Analysis*

To help the developers better understand the changes of major app issues over versions, we also involve visualization in the end and provide the issue analysis.

1) Visualization: We adopt ThemeRiver [19] to represent the transformation of app issues. The technique has been used in handling large document collections [18], [22], but never in the user reviews. The reason we use ThemeRiver is that it provides users with a macro-view of issue changes in the corpus over a serial dimension.

Figure 8 shows a sample ThemeRiver visualization on Facebook. The “river” flows from left to right through versions, changing its width to depict changes in the thematic importance of temporally associated reviews. The *width* is denoted by the dash line in Fig. 8. Colored “current” flowing within the river narrow or widen to indicate decreases or increases in the importance of an individual issue or a group of issues in the related reviews. Here, the importance of the issue is defined as the occurrence frequency of the selected phrase for that issue in the corresponding review collection.

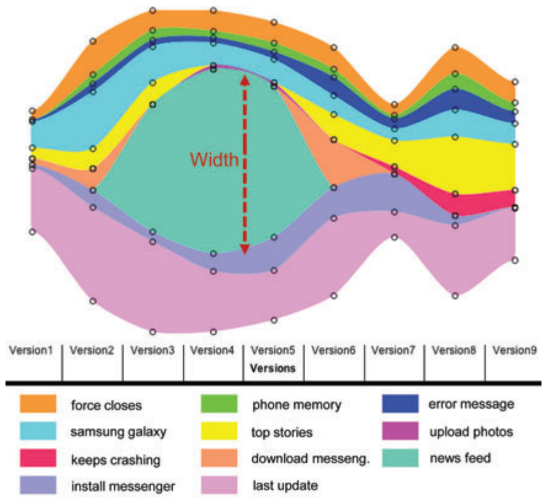


Fig. 8: A Sample ThemeRiver Visualization on Facebook. The colored “current” within the “river” indicates an individual issue. The width of the “current” changing with versions denotes the corresponding topic has different degrees of importance for different versions. The issue is represented in phrase.

2) Issue Analysis: We also prioritize user reviews for the developers in the case that they want to gain a deep insight of the phrase-level issue. Similar to the sentiment score of the phrase, the importance score $I(d)$ of the review d is computed based on its length h and user rating r as well. The number of top reviews to be displayed is determined by the developers.

$$I(d) = \mathrm{e}^{\frac{-r}{ln(h)}}. \tag{7}$$

Finally, the reviews with longer lengths and lower ratings will be ranked higher. The developers can comprehend more about the app issues by checking the reviews in the toplist.

V. Empirical Evaluation

For experiments, we have crawled more than two million user reviews beginning from August 2014, and 37 apps belonging to 10 categories. Each app receives roughly 56,479 reviews on average. With multiple categories and massive reviews, we can verify the effectiveness of our framework while mitigating the data bias from only one type of reviews.

Specifically, we aim to answer the following questions: 1) What is the trend of main topics reflected by app customers along with different versions? (in V-A) 2) Have the developers modified the issues prioritized by our framework? (in V-B and V-C). We also study the influences of parameter settings on the performance of our method in V-D.

TABLE VI: The Review Dataset of 35 Apps

Category	App Name	Review Quantity
Social	Facebook	176,362
	Twitter	132,981
	Instagram	130,855
Books & Reference	Amazon Kindle	575
	Wikipedia	541
Shopping	eBay	91,368
	Amazon Shopping	792
Photography	Photo Grid - Collage Maker	91,425
	Camera360 Ultimate	79,640
	PicsArt Photo Studio	569
	Autodesk Pixlr - photo editor	500
Tools	Clean Master (Boost & AppLock)	234,342
	Battery Doctor (Battery Saver)	116,534
	CM Security Antivirus AppLock	87,785
Travel & Local	Booking.com Hotel Reservations	29,632
	Google Earth	18,919
	Expedia Hotels, Flights & Cars	1,367
	Foursquare - Best City Guide	494
Communication	WhatsApp Messenger	130,761
	Skype - free IM & video calls	103,479
	Messenger	95,070
	Viber	87,647
	LINE: Free Calls & Messages	70,408
	Chrome Browser - Google	54,707
	Wechat	46,330
	Hangouts	27,515
	Gmail	21,138
	CM Browser - Fast & Secure	500
	Firefox Browser for Android	500
	Contacts+	499
Education	Coursera	608
	Duolingo: Learn Languages Free	487
Productivity	Evernote	48,525
	SwiftKey Keyboard + Emoji	48,028
	ES File Explorer File Manager	507
Music & Audio	YouTube	86,395
	Spotify Music	71,952

A. *Case Demonstration*

Here, we show the effectiveness of PAID on apps from different categories: Viber (Communication), Evernote (Productivity), Camera360 Ultimate (Photography), and Spotify Music (Music & Audio). The numbers of user comments belonging to various versions of these four apps are shown in Fig. 9.

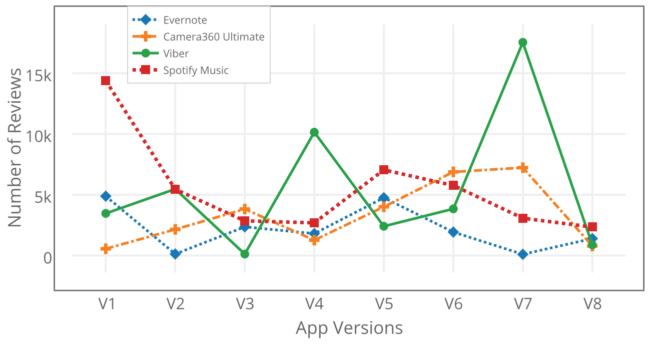


Fig. 9: The Distribution of User Reviews for Different Versions of the Mobile Apps. We overlook the versions with less than 100 user reviews, and display the latest eight versions in the collection.

The *Phrase Bank* is established based on the proposed rules. Table VII illustrates the top 25 phrases from the *Phrase Bank* of Viber.

TABLE VII: *Phrase Bank* (Top 25) of Viber

video call, video calls, activation code, samsung galaxy, call quality, phone number, internet connection, free calls, video calling, profile picture, sound quality, public chat, sony xperia, voice quality, online status, globus mobile, asus zenfone, start earning, voice call

In the toplist, phrases "video call", "video calls", and "video calling" are almost exactly the same. This is because Viber is characterized by video calling, but developers may not desire such repetitive information. Each prioritized phrase must resemble one most relevant topic and distinguish from the others. Based on dDAL and our issue-prioritizing method, the phrases possessing the highest total scores are elected for the topics. We present them with the corresponding scores and along with versions in Table VIII. The phrases can cover more kinds of app issues (*e.g.*, the phrases for topics of Viber 5.2.2.478 are "free calls", "sony xperia", "animated stickers", "chat background", "activation code", and so on.), and the semantics of the phrases are consistent within one topic (*e.g.*, the phrases for topic 3 are "animated stickers" for Viber 5.2.2.478, "download stickers" for Viber 5.3.0.2274, and "download stickers" for Viber 5.3.0.2331). They are the issues to be reported to the developers finally.

TABLE VIII: Phrases Prioritized for Viber Developers ($k = 8, \mu = 1$)

Topics	5.2.2.478	5.3.0.2274	5.3.0.2331
Topic 1	free calls:0.67	free calls:0.66	free calls:0.65
Topic 2	sony xperia:0.97	sony xperia:0.93	sony xperia:0.91
Topic 3	animated stickers:0.53	download stickers:0.56	download stickers:0.63
Topic 4	chat background:0.69	incoming messages:0.70	chat background:0.73
Topic 5	activation code:3.66	activation code:3.69	activation code:3.72
Topic 6	galaxy tab:0.63	galaxy tab:0.62	samsung galaxy:0.61
Topic 7	voice call:1.95	call quality:1.94	call quality:1.94
Topic 8	start earning:1.42	start earning:1.44	start earning:1.45

However, it is extremely tedious to decide which bug to fix by analyzing so many phrases and figures directly. Therefore, we apply ThemeRiver to visualize the generated results. Fig. 10 displays the ThemeRiver of Viber. The width of the "current" represents the importance score of the issue by counting the occurrence frequency of the corresponding phrase. The issues are in random order within one version, but their positions are consistent over versions. For example, Fig. 10 demonstrates there is an increasing trend of the issue "activation code", colored in pure blue, for Viber 5.2.1.36, 5.2.2.478, and 5.3.0.2339.

To obtain an in-depth knowledge about one issue, such as "activation code", we provide the top reviews associated with that issue based on the method in IV-C. Table IX lists the top three reviews related to "activation code". All these reviews express some bugs of the app and illustrate different aspects (*e.g.*, "Messages are not present" in review 1, "a white popup written only" in review 2, and "keeps on saying activation code sent to your device" in review 3). Therefore, developers can examine the urgent concerns from users by viewing issues

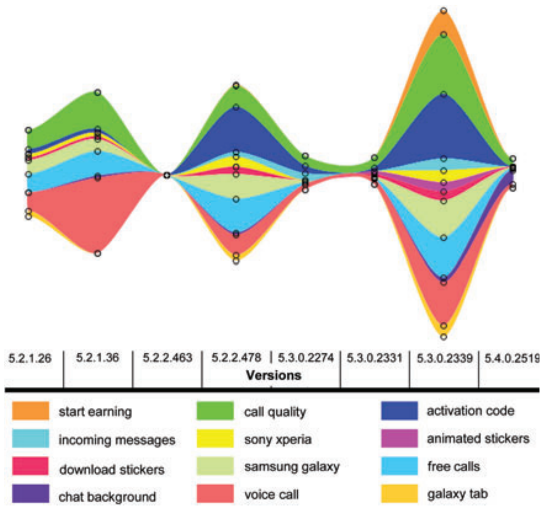


Fig. 10: The Themeriver of Viber. The horizontal axis denotes app versions while the vertical axis means the start point of the "river". Each "current" represents a phrase-level issue. Wider currents stand for more important issues.

first, then deciding and analyzing them deeply, and finally scheduling the modification.

TABLE IX: The Top Three Reviews Related to "Activation Code"

	User Review	Importance Score
1	Upload viber! I went. Enter a phone number. I enter. Asks for sure your phone? It will be sent an activation code. Ok. Messages are not present. He writes to activate viber here, install it to your phone first. But I have it pumped? What to do? Help!	0.836
2	I hard reset my tab 3. Installed viber for activation code when i write my phone number and press okay a white popup written only. ERROR no description given and an okay button on it please help me vibers my only way to contact my son abroad.	0.834
3	I don't know what's wrong with Viber. Just downloaded it nd it keeps on saying activation code sent to your device. For almost a month, no any activation code and it's really pissing me off. Pls fix.	0.828
…	…	…

B. Performance Evaluation

To establish the connection between the analysis of user reviews and decisions made by the developers, we employ the changelogs of mobile apps. Changelog [5] is a record of all the changes made to a software project, usually including such records as bug fixes, new features, etc. It is a first-hand and practical ground truth *labeled* by the developers directly.

We collect the official changlogs of six versions of Viber from APK4Fun [1] for evaluation. The version of the changelog we compare is the one immediately following the experimental version. For example, to assess the result of Viber 5.2.1.36, we need to inspect the changelog of the next version, *i.e.*, 5.2.2.478. Since our issues are in phrase level, we manually summarize the changelogs into phrases (not limited to 2-gram terms) as Table X. We remove the meaningless phrases and sentences such as "bug fixes", "General stability and bug fixes to improve overall performance.", etc. The highlighted phrases comprise the ground truth of Viber.

TABLE X: The Changelogs of Viber and Its Identified Phrases

Versions	Detailed Changelog & Identified Phrases
5.2.1.36	• Improved sticker menu[a] • redesigned forward screen gives you the option to send to groups, contacts; • Public chats; • ~~General stability and bug fixes to improve overall performance.~~[b]
5.2.2.478	• ~~Bug fixes~~ and updated emoticons.
5.3.0.2274	• Become an Admin and manage your group chats; • Send and receive messages from your watch; • Clearer contact info; • Public Chat ~~enhancements~~.
5.3.0.2331	Same as the previous version.
5.3.0.2339	Same as the previous version.
5.4.0.2519	• Enhancements for tablet users; • Easier to activate Viber account on your tablet; • Improved call and video screens; • Send multiple photos more easily; • Personalise your groups with your own icon; • Customise Viber notifications in Priority Mode on Android L only.

[a] The phrases highlighted constitute the ground truth.
[b] The phrases or sentences with line in the middle are discarded.

To measure the similarity between the prioritized issues L and the ground truth U, we adopt the sentence-based semantic similarity measure method in [24]. The method focuses directly on computing the similarity between very short texts of sentence length, which fits for our situation. We denote the similarity degree between two phrases as $s(u, l)$, where u indicates the phrase in the ground truth, and l means the phrase in the prioritized issues. For each phrase u in the ground truth, we compute its similarity degrees to all the phrases in our results, and the highest one is defined as the rate of the phrase $Rate(u)$.

$$\arg\max_{u} Rate(u) = \{s(u, l) | \forall l : l \in L\} \quad (8)$$

The precision of our method is indicated by the average rate of all the phrases in the ground truth. The pseudo-code is illustrated as below.

Similarly, the precision of other three apps (Evernote, Camera360 Ultimate, and Spotify Music) is also computed for demonstration. Figure 11 depicts the results of these four apps, with the average precision and standard deviations shown in Table XI. All the four apps have precision larger than 55% and two of them (Viber and Camera360 Ultimate) are larger than 70%. Three of the four apps produce the standard deviations less than 0.045, while only the output of Camera360 Ultimate is larger than 0.1. From the Fig. 11, we can observe that V2 of Camera360 reaches the lowest of its record. Checking the corresponding changelog, we find it is just one sentence "New HDR 'Storm' effect will blow your mind", which is manually identified as "HDR Storm effect" (a kind of technique used in imaging and photography). Contrasting with our prioritized issues ("selfie camera", "tilt shift", "stock camera", "save

```
Algorithm 1: Precision-Computation($U$, $L$)
Data: Prioritized issues $L$ and the ground truth $U$
Result: Return the precision of our method.
$Rate \leftarrow 0$;
while $u$ in $U$ do
    $s \leftarrow 0$;
    while $l$ in $L$ do
        $t \leftarrow s(u, l)$;
        if $t > s$ then
            $s \leftarrow t$;
        else
        end
    end
    $Rate \leftarrow (Rate + s)$;
end
$Rate \leftarrow Rate/Num(U)$;
```

edited", etc.), we consider the main reason for the result is that the phrase in the changelog is brand-new and has not been embodied in the semantic corpus. However, the acceptable performance of other apps displays the effectiveness of our method. Without the loss of generality, we provide the evaluation of the available apps in our collection in the following.

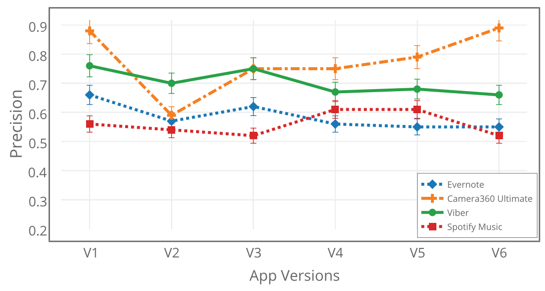


Fig. 11: The Precision of Our Method for Apps from Four Different Categories. The experimental parameters for similarity measure settings are $\alpha = 0.002, \beta = 0.3, \eta = 0.6, \phi = 0.8$, and $\delta = 0.9$.

TABLE XI: Average Precision of Four Apps and Their Standard Deviations

	Viber	Evernote	Camera360 Ultimate	Spotify Music
Average Precision	0.703	0.585	0.775	0.560
Standard Deviation	0.042	0.045	0.109	0.041

C. Generality Evaluation

We employ the apps in Table VI to demonstrate the generality of PAID. Searching the changelogs online, we discover that 19 of the 37 apps have no detailed information about version modification. So we adopt the remaining 18 apps with 117 version changes for the verification. The result is shown in Fig. 12. The average precision for these apps is 60.3%, which is quite acceptable for app review analysis [13].

To analyze the correlation between the app category and the framework performance, we compute the average precision

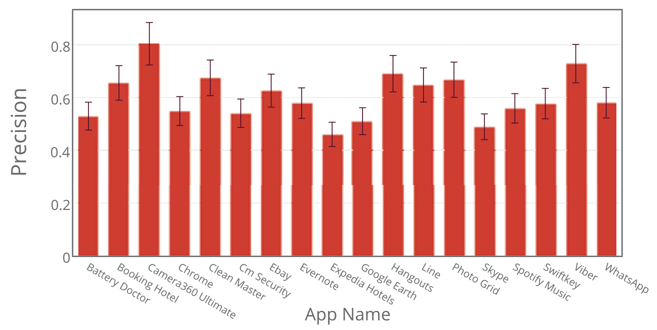


Fig. 12: The Precision of 18 Apps in the Collection. The overall times of comparing is 117.

of the apps belonging to one category and the corresponding average review number (Fig. 13). We discover that a larger review quantity tends to produce a better result.

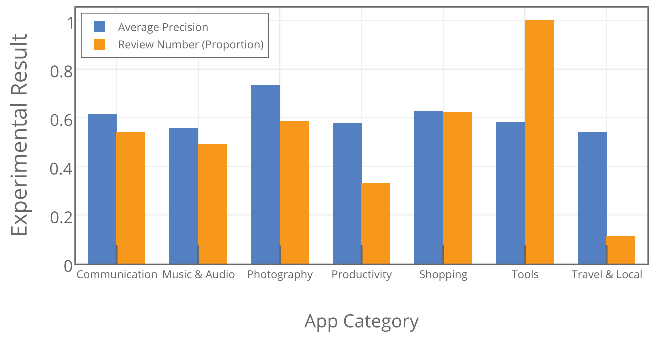


Fig. 13: The Relation between the Average Precision and the Review Number Regarding to App Category. The review numbers have been normalized by the maximum value in the experimental dataset.

D. Parameter Study

In this section, we study the influence of parameter settings (*i.e.*, the number of topics k for each version and the penalty factor μ in Section IV-B3) for our method. Figure 14 shows that a larger number of topics can produce better precision. This is because more topics introduce more issues, and hence possess a higher probability to cover the changelog. Figure 15 indicates more penalty on the dissimilarity to other topics can generate more promising precision and lower standard deviation. Larger penalty makes the prioritized issues more diverse. Therefore, more phrases in the changelogs can be covered.

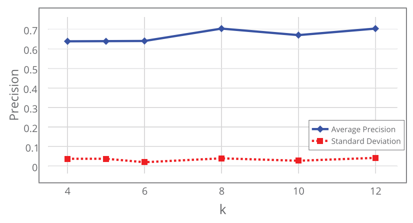


Fig. 14: The Influence of k on Precision.

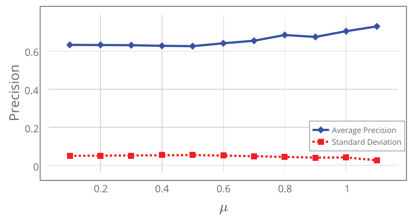


Fig. 15: The Influence of μ on Precision.

VI. Limitations and Discussion

This section discusses the validity and generalizability of our findings.

As for the validity, our framework requires a large number of user reviews. It restricts the size and quality of the *Phrase Bank* and further influences the performance of prioritizing issues. Moreover, since the width of “current” in ThemeRiver is calculated by the occurrence of the corresponding issue in the collection, insufficient reviews cause a narrow current, even if the issue is considerably crucial.

With respect to the generalizability, firstly, the category of apps may affect the phrase-prioritizing process. For game apps, since users tend to leave more significantly shorter and non-informative reviews than the apps of other categories [34], the generated *Phrase Bank* may contain few phrases. Hence, the selected phrases to topics may not be very meaningful. However, to reduce the interference, we can try to use the low-rating reviews and introduce linguistic rules to extract useful features. Secondly, for apps with only one version, dLDA is not applicable, but we can use LDA alternatively which possesses the same functionality as dLDA essentially. Finally, we implement and testify our framework on reviews of apps from Google Play. It is uncertain whether our framework can achieve similar performance for apps in other store (*e.g.*, App Store and Amazon Appstore). Future work will conduct a large-scale empirical study to address this threat.

VII. Related Work

Two lines of work inspire the design of PAID, namely exploration on mobile app resources, and NLP (Natural Language Processing) in app review analysis. The details are discussed in the following.

A. Exploration on Mobile App Resources

Same as traditional software repositories such as codes and documents [25], [35], user reviews on mobile apps are also beneficial for software development. They are valuable resources directly from customers and can be exploited by developers during the bug-fixing process.

There has been an amount of research work on this topic. In [16], the authors conduct a content analysis of online user reviews to spot the factors that users find important in mobile devices. They find four factors to be significantly related with overall user evaluation, namely functionality, portability, performance, and usability. Similarly, to understand user requirements, Elisabeth [32] explores automated classification of user reviews concerning the usage motives mentioned in the reviews. There is "MARA" (Mobile App Review Analyzer) [21], a prototype for automatic retrieval of mobile app feature requests from online reviews. Bin *et al.* [14] also probe into identifying users' major concerns and preferences from user reviews. One of the differences between their work and the previous work is that they further analyze the user issues of different types of apps. However, the issues in the work are manually defined. Continuously, with regard to automatically analyzing user issues, Stuart *et al.* [26] contribute to labeling the types of issues from the whole app store. Different from these investigations, we focus on displaying user issues of a specific app and aim at providing detailed user concerns specifically for the developers, instead of general issue types. Our work regards app review analysis as an indispensable link during the app modification process for quality and reliability purpose.

User sentiment analysis is also a popular research topic. In [14], the authors propose "WisCom" to discover reasons why users like or dislike an app, which is then utilized for identifying inconsistencies in reviews. More detailedly, Emitza and Wailid [17] present an automated approach to identify fine-grained app features in the reviews, from which more meaningful high-level features are extracted based on the user sentiments. In our work, we also define the user sentiment to detect the most helpful issues for the developers.

However, the existing studies mainly focus on static review analysis and ignore the fact that user reviews change with app versions. In [14], the authors try to analyze how the number of comments varies with time and recognize the main reasons for the spikes. Likewise, the study in [23] empirically examines the growth pattern of rating and price in the app store. Different from these tasks, we concentrate on analyzing the topic changes instead of the intuitive variations. Although there is one investigation [7] on the feature transformation in the Blackberry app store, it targets at all the apps belonging to one category and the features are more general. In the paper, we devote to providing detailed suggestions to the developers; therefore, our tool is app specific and the features are more concretely related to software quality and reliability issues.

B. NLP in App Review Analysis

Rule-based methods, topic modeling, and supervised learning are widely used in review analysis. In [21], [32], part-of-speech rules are utilized for limiting the contexts from which feature requests can be extracted.

In [13], [14], [17], the topic modeling method LDA (Latent Dirichlet Allocation) [12] is adopted. LDA is a probabilistic distribution method which uses Gibbs sampling to assign topics to user reviews. This indicates that each review can be associated to different topics and that topics are related to different words with a certain probability. Galvis and Winbladh [15] explore the usage of LDA to summarize user reviews, but their work focuses on semantic analysis while our framework aims to recommend significant software issues to developers.

In [13], a supervised learning method is used to filter out the non-informative reviews, where the non-informative reviews and informative reviews are labeled as the training set first. The subsequent analysis only considers the informative reviews. However, their work mainly considers from the perspective of users. Our work devotes to providing developers with suggestions on arranging modification items by involving app changelogs. That is, our focus is more on what developers concern, particularly with respect to software quality.

Furthermore, the previous research on app review analysis involves little visualization, which would greatly assist the developers in absorbing the knowledge mined from user comments. The topic-based visualization has a rich history due to the widespread application of topic modeling methods so far. Text clouds [33] is widely used in the field of text mining. The font size and color, the orientation of the text, and the proximity of tags to one another can be applied to convey some information to the observer. TextArc [31] is an alternative view of text, tailored to expose the frequency and distribution of the words of the entire text on a single page or screen. However, these techniques are not applicable for dynamic topic visualization. ThemeRiver [19] is such a visualization system aimed at displaying thematic variations among large collections of documents. In the paper, we employ ThemeRiver to represent the textual information to developers. Consequently, the developers can spot the most urgent issues quickly and do not have to struggle with the burdensome texts. This, to our best knowledge, is the first adaptation of the visualization technique for mobile app developers.

VIII. Conclusion

This paper proposes a framework PAID to prioritize app issues for the developers automatically and accurately. With PAID, the developers can utilize a systematic process to schedule the modification of the app, including which bugs to fix or what features to add. Almost no manual labors are consumed in the process. We adopt *Filter Bank* to exclude non-informative words. The developers are supposed spend less time on analyzing user reviews if the issues are in phrase level, and hence PAID focuses on recommending phrase-level issues to the developers. A *Phrase Bank* is establishes by the rule-based method, and then combined with topic modeling to generate the ultimate issues. Finally, ThemeRiver is employed to enable the textual results more seamlessly presented to the developers.

In future work, we will study what is the difference between the decisions made by the developers and the concerns made by users. Overall, our framework is rather generic and extensible to incorporating more requirements.

Acknowledgment

This work was supported by the National Basic Research Program of China (973 Project No. 2014CB347701), the Key Project of National Natural Science Foundation of China (Project No. 61332010), the Research Grants Council of Hong Kong (Project No. CUHK 14205214), and Microsoft Research Asia Grant in Big Data Research (Project No. FY13-RES-SPONSOR-036). Yangfan Zhou is the corresponding author.

References

[1] APK4Fun. http://www.apk4fun.com/.

[2] AppBrain. http://www.appbrain.com/.

[3] AppFigures. https://appfigures.com/getstarted.

[4] AR-Tracker: Track the Dynamics of Mobile Apps via User Review Mining. https://cdn.rawgit.com/TIIC/AR-Tracker/master/AR-Tracker.html.

[5] Changelog on Wikipedia. http://en.wikipedia.org/wiki/Changelog.

[6] Lemmatization. http://en.wikipedia.org/wiki/Lemmatisation.

[7] Life and Death in the App Store: Theory and Analysis of Feature Migration. http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/RN_14_12.pdf.

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

[9] NLTK Wordnet Synsets. http://www.nltk.org/howto/wordnet.html.

[10] Stemming. http://en.wikipedia.org/wiki/Stemming.

[11] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 113–120. ACM, 2006.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[13] 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 Conference on Software Engineering (ICSE)*, pages 767–778. IEEE, 2014.

[14] 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.

[15] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 582–591. IEEE, 2013.

[16] 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.

[17] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the 22nd International Conference on Requirements Engineering (RE)*, pages 153–162. IEEE, 2014.

[18] Y. Hashimoto and R. Matsushita. Heat map scope technique for stacked time-series data visualization. In *Proceedings of the 16th Conference on Information Visualisation (IV)*, pages 270–273. IEEE, 2012.

[19] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, 2002.

[20] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the 10th SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 168–177. ACM, 2004.

[21] 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.

[22] D. Ip, K. L. Ka Keung, W. Cui, H. Qu, and H. Shen. A visual approach to text corpora comparison. In *Proceedings of the 1st International Workshop on Intelligent Visual Interfaces for Text Analysis*, pages 21–24. ACM, 2010.

[23] J. Kim, C. Kim, Y. Park, and H. Lee. Trends and relationships of smartphone application services: Analysis of apple app store using text mining-based network analysis. In *Proceedings of the 4th ISPIM Innovation Symposium*, 2012.

[24] Y. Li, D. McLean, Z. A. Bandar, J. D. O'shea, and K. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1138–1150, 2006.

[25] H. Lu, E. Kocaguneli, and B. Cukic. Defect prediction between software versions with active learning and dimensionality reduction. In *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 312–322. IEEE, 2014.

[26] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, pages 1–40, 2015.

[27] Q. Mei, X. Shen, and C. Zhai. Automatic labeling of multinomial topic models. In *Proceedings of the 13th SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 490–499. ACM, 2007.

[28] R. C. Moore. On log-likelihood-ratios and the significance of rare events. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 333–340. Association for Computational Linguistics, 2004.

[29] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song. Facilitating developer-user interactions with mobile app review digests. In *Proceedings of the 2013 CHI Extended Abstracts on Human Factors in Computing Systems*, pages 1809–1814. ACM, 2013.

[30] M. P. O'Mahony and B. Smyth. Learning to recommend helpful hotel reviews. In *Proceedings of the 3rd Conference on Recommender Systems*, pages 305–308. ACM, 2009.

[31] W. B. Paley. Textarc: Showing word frequency and distribution in text. In *Proceedings of Symposium on Information Visualization*, volume 2002. IEEE, 2002.

[32] 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.

[33] A. A. Puretskiy, G. L. Shutt, and M. W. Berry. Survey of text visualization techniques. *Text Mining: Applications and Theory*, pages 105–127, 2010.

[34] 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 (CHI)*, pages 241–244. ACM, 2012.

[35] K. S. Yim. Norming to performing: Failure analysis and deployment automation of big data software developed by highly iterative models. In *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 144–155. IEEE, 2014.