Facilitating Developer-User Interactions with Mobile App Review Digests

Jeungmin Oh
Department of Knowledge Service Engineering,
KAIST, Daejeon, Korea
jminoh@kaist.ac.kr

Daehoon Kim
Department of Knowledge Service Engineering,
KAIST, Daejeon, Korea
daehoonkim@kaist.ac.kr

Uichin Lee
Department of Knowledge Service Engineering,
KAIST, Daejeon, Korea
uclee@kaist.ac.kr

Jae-Gil Lee
Department of Knowledge Service Engineering,
KAIST, Daejeon, Korea
jaegil@kaist.ac.kr

Junehwa Song
Department of Computer Science,
KAIST, Daejeon, Korea
junesong@kaist.ac.kr

Abstract

As users are interacting with a large of mobile apps under various usage contexts, user involvements in an app design process has become a critical issue. Despite this fact, existing apps or app store platforms only provide a limited form of user involvements such as posting app reviews and sending email reports. While building a unified platform for facilitating user involvements with various apps is our ultimate goal, we present our preliminary work on handling developers' information overload attributed to a large number of app comments. To address this issue, we first perform a simple content analysis on app reviews from the developer's standpoint. We then propose an algorithm that automatically identifies informative reviews reflecting user involvements. The preliminary evaluation results document the efficiency of our algorithm.

Author Keywords

Mobile app; app review; comment classification; user involvement; user-centered design

ACM Classification Keywords

I.5.2 [Design Methodology]: Classifier design and evaluation; H.1.2 [User/Machine Systems]: Human factors
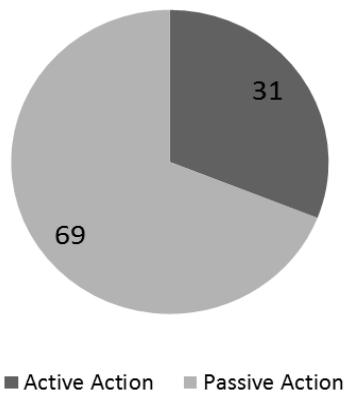
Figure 1. "How do you react when you want to communicate with app designer regarding a mobile app?"

Introduction

User involvements in software design have been one of the important issues in software engineering [1, 2]. Understanding user needs and usage contexts would be the key factors in increasing the potential utility/value to the end users. For this reason, practitioners often perform various forms of user involvements across the development lifecycle including customer interviews, usability testing, beta deployment, and end user support.

However, the widespread use of smartphone apps has significantly changed the traditional software development environment. App design is mostly utility-driven, and rapid prototyping with a limited form of user involvements is often performed. Further, usage patterns of mobile apps are quite unique as opposed to existing software (e.g., short session time of mobile apps); and usage context also varies widely (e.g., at home, on the move). In this environment, we think that facilitating user involvements will help the developers to better understand user needs and usage context.

Existing apps and app store platforms often rely on a passive form of user involvement (e.g., posting app reviews in app stores or sending emails to the developers). In the case of open source apps, sometimes a bug tracking system is used to promote participation of grassroots developers (e.g., Bugzilla for *Firefox for Andriod*). This means that user participation is mostly inactive (i.e., unilateral communication) and fragmented (across different app stores and locales). Likewise, developers are challenged with a large volume of app comment streams (from different app

stores using heterogeneous devices/OSs, and with possibly different languages).

While building a unified platform for facilitating user involvements with various apps is our ultimate goal, as a first step, we focus on mitigating developers' information overload attributed to a large number of app comments. In the field of software engineering, there have been several studies on summarizing bug reports in often source software (e.g., Debian) [3, 4], but our work differs as we consider end-user feedback posted in app stores. In this paper, we investigate how developers and users interact within an app store environment. We present a method of filtering mobile app reviews to reduce the information overload of developers.

Developer-User Interaction

To get a basic understanding of how developers and users interact, we conducted a survey (n=100) on user's motivation and behavior. The survey was administered to randomly chosen smartphone users via a survey research company. The majority of participants are in their 20s (55%), and 22% are under 20. 18%, 2%, 2% and 1% are in their 30s, 40s, 50s and 60s respectively. 40% are males.

Q1 (common) "How do you react when you want to communicate with app designer regarding a mobile app?"
Figure 1 shows that users are more likely to take a passive action such as deleting apps rather than to perform active actions such as writing an app store review or sending an email to the developer. To find the reason for this behavior, we asked an additional question both for active and passive groups separately.
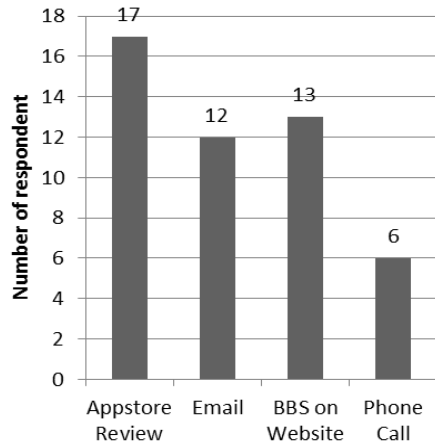
Figure 2. "Which methods do you use to communicate with developer?"

Q2 (active group) "Which methods do you use to communicate with developers?" (multiple answer)
In addition to four original selections, which we derived by interviewing graduate students in the authors' department, we allow participants to report any other channels in a free-text format, but we didn't find any other channels. As shown in Figure 2, the result reveals that the most popular channel is writing app store reviews as we expected, but the traditional methods like phone calls or BBS are used.

Q3 (passive group) "What are your reasons for reacting passively?" (multiple answer)
Like the previous question, we gave optional open-ended text field to find out unknown reason of passiveness and we identified 'tiresome' from comments of four users. As shown in Figure 3, most users expected the inquiry would take long time to be responded or receive no response. It also indicates that a non-negligible portion of recipients were not even aware of which channel to use.

To summarize, users would like to communicate with developers using app store reviews. Additionally, the reasons why the user passively reacts were mainly due to low responsiveness.

Filter for Classifying App Store Review

Since app review in app stores (e.g., Google Play, Apple App Store) is not mainly designed for reporting bugs as in bug tracking systems, it contains not only informative feedback but also simple expressions like a user's sentiment. Due to a large volume of app comments, developers wasteWH a significant amount of time on checking reviews without any meaningful issue, for example "What an awesome app, comes in handy
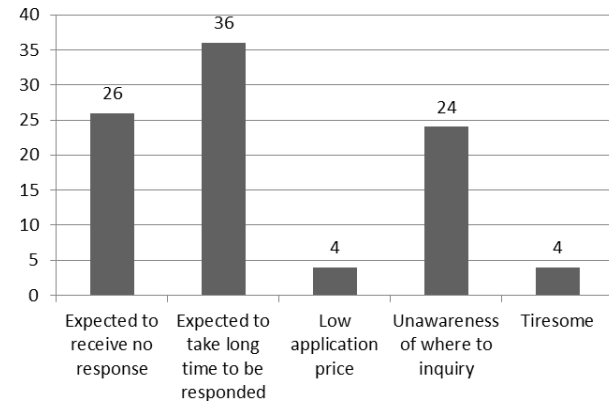

Figure 3. "What are your reasons for reacting passively?"

so many times a day. thank you", which is just mere expression of satisfaction. In this sense, a filter for classifying reviews which are less informative for developers could play an important role to save developer's time and energy, which otherwise could be used for quality/timely feedback.

To build a model that filters out uninformative comments, we used the following steps: (1) automated crawling publicly accessible reviews, (2) manual coding sampled reviews to decide whether each review contains issues from the developer's standpoint, (3) model construction/training/evaluation using the annotated dataset.

Data Collection
We collected public reviews of 24,000 applications in Google Play. Since Google Play doesn't provide the entire list of apps in search result, we put two digit combinations of alphanumeric characters as a search

Category	Description
Functional Bug	Functional bug of an app; e.g., "Black screen issue on the Galaxy S3"
Functional Demand	Functional demands about the features that a user wishes to have. e.g., "This seriously needs a postcode search! It's a must for any navigation app. Apart from that seems to work fine"
Non-functional Request	Non-functional request or complaint such as design and content related issues: e.g., "We want another car and more stages and more gas in high way map."

query for building a list of apps. We crawled the reviews from November 21 to 28, 2012, and the total number of reviews is 1,711,556. Each review contains an app's name, category, rating (in 5-levels), posting date, device, title and text.

Manual Coding by Developers
We performed preliminary content analysis and derived three issue categories as shown in Table 1. The categorization was done thoroughly in developers' standpoint because the readers of comments include people who are directly involved with software design and production. The issues consist of three categories as described in Table 1. We then classified 2,800 reviews into these issue categories and analyzed it in terms of rating and word counts to attain some insights into the patterns of app reviews. This data set was randomly chosen to avoid bias toward specific applications as follows: 20 reviews from 10 applications in 14 categories. The categories are determined by the raters who have mobile app programming experiences more than 1 year.

The coding results are presented in Figure 4. The reason why the sum of occurrences is not the same as the original number is that we excluded non-English and unreadable comments. The results reveal that 66% of reviews (1851 out of 2785) are uninformative for developers. The inter-rater agreement with Cohen's Kappa coefficient measured as 0.9025 indicating good agreement.

We also plotted the number of reviews in terms of word counts in Figure 6. It shows that there is a significant gap between the frequency of the overall comments
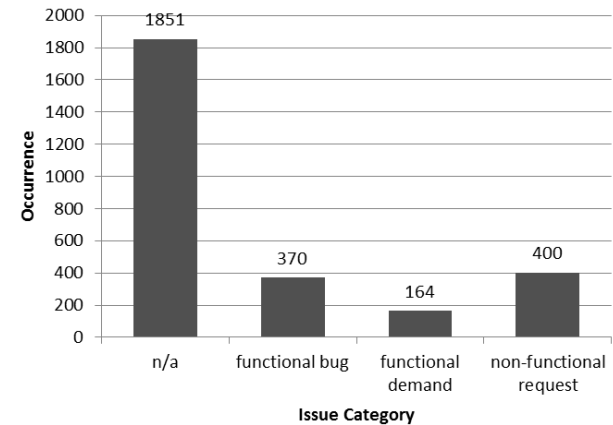

Figure 4. Distribution of categories including review without issue (marked as 'n/a')

and informative comments with respect to the word counts.

Issue Keyword Count
To find a set of keywords related to informative comments, we firstly ran *Latent Dirichlet Allocation (LDA)* on informative comments. However the result was unrecognizable even though it was examined by two developers. This means that existing automatic topic classifications like LDA cannot be directly used for finding informative comments. The reason why it fails is that the length of a comment is too short for producing topic keywords (mean=18.27 words).

Instead we devised a new method of extracting the keyword set implying informative comments. As a first step, we preprocessed the comment data using *Natural Language ToolKit (NLTK)* [5]: tokenize the word and apply stemming in order to clean the word set from
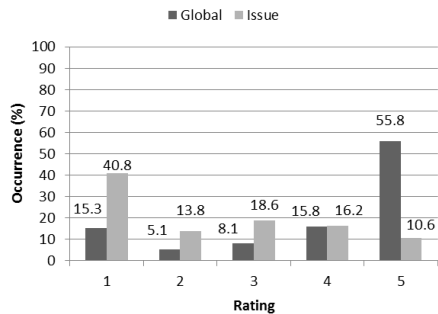
Figure 5. Distribution of rating

Figure 6. Distribution of word count

review. We define Set_{Issue} as a set of words that appear in informative comments, and $Set_{Non-Issue}$ as a set of words that appear in uninformative comments. We also define the sets' occurrence functions of a given word as $occur_{Issue}(word)$ and $occur_{Non-Issue}(word)$. Thus, we can define $Set_{Issue-freq}(k)$ an $Set_{Non-Issue-freq}(k)$ as follows.

- $Set_{Issue\_freq}(k)=\{word \mid \log(occur_{Issue}(word)) \geq k \times \log(occur_{average})$ and $word \subset Set_{Issue}\}$

- $Set_{Non-Issue\_freq}(k)=\{word \mid \log(occur_{Non-Issue}(word)) \geq k \times \log(occur_{average})$ and $word \subset Set_{Non-Issue}\}$

The reason why we apply a log function is that the occurrence distribution follows a power-law distribution. These two sets stand for the frequent word subsets of Set_{Issue} and $Set_{Non-Issue}$. Then we calculate $Set_{Issue-Only}$ which only frequents in Set_{Issue}, but do not frequent in $Set_{Non-Issue}$ as follows.

- $Set_{Issue-Only} = Set_{Issue\_freq}(k) - Set_{Non-Issue\_freq}(k)$

We empirically chose $k = 0.5$. When we chose $k = 0.6$, two coders agree that the result misses some of the very important words; e.g., 'complaint' or 'trouble'; in contrast, when k = 0.4, the result includes too many commonly-used words.

We generate a new measure called Issue Keyword Count which indicates how many words a review

includes within $Set_{Issue-Only}$. We assume that the more the number of informative keywords, the higher the value of information.

Building a model
Based on the coded data set, we trained a SVM classifier to determine whether a given comment is informative or not. Since the main goal of our model is to reduce developers' efforts to check all meaningless messages, we set our class variable to predict as a binary value indicating whether a comment is informative, instead of predicting an exact issue category.

For implementation, we used python *LIBSVM* 3.14 [6]. We tested three features, namely rating, word count and issue keyword count.

Evaluation
We performed a 5-fold cross validation on 2,785 samples to evaluate the model. Table 2 shows the result of our model. It turns out that the model performs very well with high precision and recall. According to the prediction result described in Table 2, rating which partly represents sentiment and issue keyword count is the key feature. The result also implies that our model operates regardless of applications because we sampled reviews equally from various applications from every category.
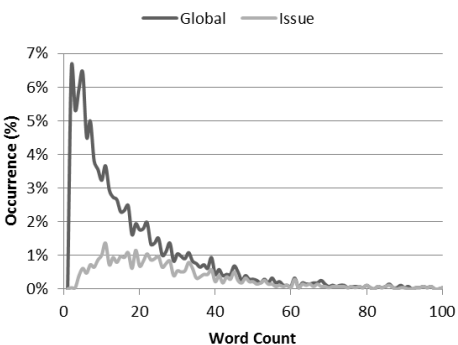
Set_{Issue-Only} of our data set

['**update**', 'text', 'rest', 'through', '**clos**', 'nam', '**slow**', 'const', 'tablet', '**delet**', 'brows', 'send', 'wer', 'going', '**black**', '4', 'photo', 'hap', 'els', 'hat', 'ic', '**crash**', 'big', 'watch', '**disappoint**', 'bit', 'thes', '**restart**', 'mod', 'wont', 'bar', 'bas', 'level', 'button', 'each', 'whil', 'droid', 'tel', 'back', '**everytim**', '**horr**', '**fail**', 'wo', 'click', 'er', '**freez**', 'review', 'landscap', 'xper', '7', 'rec', 'pls', 'mem', '**screen**', '**plz**', '**reason**', '**otherw**', 'phone', '**wtf**', 'fiv', 'post', 'wait', 'last', 'turn', 'menu', 'iphon', '**confus**', 'col', 'pop', 'sometim', 'doesnt', '**miss**', 'select', 'smal', '**crap**', 'log', '**support**', '**upgrad**', 'long', 'next', 'avail', 'anym', 'interfac', 'until', 'buy', '**forc**', 'screen.', '**uninstal**', '**bug**', '**annoy**', 'account', 'hour', 'default', '**remov**', 'un', 'ui', 'ful', 'control', 'almost', 'lock', '**sad**', 'cant', 'rid', 'seem', 'again', 'end', 'min', 'receiv', 'siz', 'check', 'ent', 'useless', 'pag', 'instead', 'complet', '**poor**', 'map', 'though', 'mess', 'lag', 'paid', 'alert', '**wrong**', '**kil**', 'why', 'icon', 'mobl', 'alarm', 'caus', 'pay', 'latest']

* k=0.6, bold words are strongly issue-related words by two raters

Features	Precision	Recall	F-measure
Random	0.3512	0.4790	0.4053
F1	0.8624	0.7306	0.7910
F2	0.6506	0.4984	0.5644
F3	0.8698	0.7033	0.7778
F1+F2	0.8591	0.7795	0.8174
F1+F3	0.8672	0.8224	0.8442
F1+F2+F3	0.8981	0.8165	0.8553

Table 1. Result of classification (F1 : rating / F2 : word count / F3 : issue keyword count)

Conclusion and Future Work

We studied the role of user involvements in mobile app development by studying app comments in an app store. Our survey results of smartphone users show that the most popular interaction channel with smartphone users is app review sections in the app store, and thus, end users are fairly passive in terms of user involvements during the app development lifecycle. We also analyzed the content of app comments and showed that there are mainly three comment types from the developers' standpoint: functional bugs, and functional demands, and non-functional requests. Further, to lower the information overload due to a large volume of app review streams, we proposed a simple approach of automatically identify informative comments. Our preliminary analysis results show that the method achieves fairly high precision and recall.

In the future, we plan to propose and implement a unified software architecture that can facilitate developer-user interactions. We will also improve the performance of our model by using better classification algorithms, and adding more functions (e.g., topic classification). Further, a longitudinal field study of a user-centered design process is needed to better understand the importance of user involvements for mobile apps.

Acknowledgements

References

[1] Michael Gallivan. The user-developer communication process: a critical case study. *Blackwell Publishing Ltd* (2003).

[2] Mark Keil, Erran Carmel. Customer-Developer Links in Software Development. *Communication of the ACM* (1995).

[3] Sarah Rastkar, Gail C. Murphy. Summarizing software artifacts: a case study of bug reports. *ACM/IEEE International Conference on Software Engineering (ICSE 2010)*.

[4] Lotufo, R., Z. Malik, and K. Czarnecki. Modelling the `Hurried' Bug Report Reading Process to Summarize Bug Reports. *Proceedings of the International Conference on Software Maintenance (ICSM 2012)*.

[5] Natural Language ToolKit (NLTK). http://nltk.org.

[6] C.-C. Chang and C.-J. Lin. LIBSVM : a library for support vector machines, *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011.