User Review-Based Change File Localization for Mobile Applications

Yu Zhou, Yanqi Su, Taolue Chen, Zhiqiu Huang, Harald Gall, Sebastiano Panichella

Abstract—In the current mobile app development, novel and emerging DevOps practices (e.g., Continuous Delivery, Integration, and user feedback analysis) and tools are becoming more widespread. For instance, the integration of user feedback (provided in the form of user reviews) in the software release cycle represents a valuable asset for the maintenance and evolution of mobile apps. To fully make use of these assets, it is highly desirable for developers to establish semantic links between the user reviews and the software artefacts to be changed (e.g., source code and documentation), and thus to localize the potential files to change for addressing the user feedback. In this paper, we propose RISING (**R**eview **I**ntegration via cla**S**sification, cluster**I**ng, and linki**NG**), an automated approach to support the continuous integration of user feedback via classification, clustering, and linking of user reviews. RISING leverages domain-specific constraint information and semi-supervised learning to group user reviews into multiple fine-grained clusters concerning similar users' requests. Then, by combining the textual information from both commit messages and source code, it automatically localizes potential change files to accommodate the users' requests. Our empirical studies demonstrate that the proposed approach outperforms the state-of-the-art baseline work in terms of clustering and localization accuracy, and thus produces more reliable results.

Index Terms—User review; Mobile apps; Information retrieval; Change File Localization.

✦

1 INTRODUCTION

THE extensive proliferation of smart devices represents one of the most visible technology and society advances of the last years. Indeed, mobile phones, tablets and smart watches are widely used in many aspects of today's life [1], [2]. This phenomenon is particularly reflected in the growth of the app industry, with millions of developed and maintained mobile applications [3], [4].

This trend also impacts the current mobile app development, which is characterized by novel and emerging DevOps practices (e.g., Continuous Integration, Deployment, Deliver, and user feedback analysis) and tools [5], [6]. For instance, the integration of user feedback (provided in the form of user reviews) in the software release cycle represents a valuable asset for the maintenance and evolution of these apps [7], [8], or for ensuring a reliable testing automation for them [9]. Thus, a key and winning aspect of successful apps is related to the capability of developers to deliver high-quality apps and, at the same time, address user requests; this is crucial for the app to stay on the market and to keep gaining users [2], [10].

• *Yu Zhou, Yanqi Su and Zhiqiu Huang are with College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. Yu Zhou is also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China.*
E-mail: {zhouyu,suyanqi,zqhuang}@nuaa.edu.cn
• *T. Chen is with Department of Computer Science and Information Systems, Birkbeck, University of London, UK. He is also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China.*
E-mail: taolue@dcs.bbk.ac.uk
• *H. Gall is with Department of Informatics, University of Zurich, Switzerland.*
E-mail: gall@ifi.uzh.ch
• *S. Panichella is with Zurich University of Applied Science, Switzerland.*
E-mail: panc@zhaw.ch

Manuscript received xxxx, 201X; revised xxxx, 201X.

Mobile user reviews, mainly appear in major online app stores (e.g., Google Play and Apple AppStore), provide valuable feedback for further improvements of mobile apps. They might report software bugs, complain about usage inconvenience, request new features, etc [1], [8], [11]. Such information is valuable for developers, since it represents crowd-sourced knowledge from the customers' perspective, providing useful information for the evolution and release planning of mobile apps [7], [8], [11]. As a concrete example, among many reviews of a popular instant messenger app signal[1], one group concentrates on the theme issues. Particularly, one review states that "*Wish it had a dark or black theme.*" In the following release of the app, new themes, including the aforementioned dark and black ones, were integrated. As another example, for the app AcDisplay[2], one review states "*It would be great if you could just use the proximity sensors to wake the screen much like the Moto app uses IR sensors when you wave over the phone.*" Later on this feature was added in the next version of the app.

Due to the high number of app user reviews developers receive on a daily basis (popular apps could receive more than 500 reviews per day on average [12]), collecting and analyzing them manually becomes increasingly infeasible. As a result, developers are interested in adopting automated approaches which are able to classify/cluster such reviews, and to localize potential change files. This is key to enhance the development productivity, and in turn, to facilitate the continuous delivery of app products.

Recent work has proposed tools for user feedback classification [8], [13], clustering [14], [15] and summarization [16], [17]. Unfortunately, most of these tools suffer from various important limitations. First, the classification or

1. https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms
2. https://play.google.com/store/apps/details?id=com.achep.acdisplay

clustering accuracy is hindered by the general low-quality of user reviews [11], [8], [15]. Compared to other kinds of software artefacts such as software documents, bug reports, logging messages which are provided by developers, reviews are generated by (non-technical) users, who tend to produce reviews of lower-quality (e.g., the textual descriptions are usually short and unstructured, mixed with typos, acronyms and even emojis [18]). Second, existing classification and clustering is usually conducted at a coarse-grained sentence level containing potentially multiple topics without taking domain-specific knowledge into account. This further reduces the accuracy of classification/clustering, and impedes the effectiveness of further localization. Third, and the most important, available tools are not able to cope with the lexicon gap between user reviews and software artefacts (e.g., the source code) of the apps, which makes the standard textual similarity based localization approaches less effective [15], [9], [13]. Consequently existing tools have to settle with a low file localization accuracy [15], [13].

To overcome the aforementioned limitations, in this paper, we propose *RISING* (user-**R**eviews **I**ntegration via a cla**S**sification, cluster**I**ng, and linki**NG**), an automated approach to support the continuous integration of user feedback via classification, clustering, and linking of user reviews. Specifically, *RISING* leverages domain-specific constraint information and semi-supervised learning to group reviews into multiple fine-grained clusters concerning similar user requests. Then, by combining the textual information from both commit messages and source code, it automatically localizes the files to change to accommodate the users' requests. Our empirical studies demonstrate that the proposed approach outperforms state-of-the-art baselines [15] in terms of accuracy, providing more reliable results.

The main contributions of the paper are summarized as follows:

- We propose a semi-supervised clustering method by leveraging domain-specific knowledge to capture constraints between the reviews. The experiments demonstrate its efficacy in improving the accuracy of clustering, in particular, its superiority to other clustering methods reported in the literature.
- We propose a change file localization approach by exploiting commit messages as a media to fill the lexicon gap between user reviews and software artefacts, which, as the experiments demonstrate, enhances the localization accuracy substantially.
- We collect user reviews and commit messages from 10 apps available from Google Play and Github, and prepare a dataset with the processed reviews and commit logs.[3] This will not only facilitate the replication of our work, but also serve other related software engineering research for example mining mobile app store and intelligent software development.

Structure of the paper. Section 2 gives background information for a better understanding of the context of our work, while Section 3 details the proposed approach to address the limitations of state-of-the-art approaches on user reviews

3. https://csyuzhou.github.io/files/dataset.zip

analysis. Section 4 presents the main research questions driving our investigation and describes the case studies we conduct to answer the research questions. In Section 5 we provide some discussions, including the threats that may bias our results and how we mitigate them. Related work is discussed in Section 6, while Section 7 concludes the paper and describes our future research agenda.

2 BACKGROUND

This section provides a brief overview of (i) the contemporary development pipeline of mobile applications and (ii) the importance of user feedback analysis in the mobile context. Section 6 complements this section by providing related work on user feedback analysis and applying Information Retrieval (IR) in software engineering, with a specific emphasis on the mobile application domain.

Development Release Cycle of Mobile Apps. As shown in Figure 1 [19], the conventional mobile software release cycle has evolved in recent years into a more complex process, integrating DevOps software engineering practices [20], [21]. The DevOps movement aims at unifying the conflicting objectives of software development (Dev) and software operations (Ops), with tools for shortening release cycle activities. Continuous Delivery (CD) is one of the most emerging DevOps software development practices, in which developers' source/test code changes are sent to server machines to automate all software integration (e.g., building and integration testing) tasks required for the delivery [22]. When this automated process fails (known as *build failure*), developers are forced to go back to coding to discover and fix the root cause of the failure [23], [24], [25]; otherwise, the changes are released to production in short cycles. These software changes are then notified to users as new updates of their mobile apps. In this context, users usually provide feedback on the new version (or the A/B testing versions) of the apps installed on their devices, often in the form of comments in app reviews [11], [8], [26], [14], [17]. For completeness, it is important to mention that developers do not look only into user reviews to gather information about the overall user satisfaction. Indeed, they also rely on metrics concerning users' behavior inside the app [27], [28], and this is especially true when they apply A/B testing strategies [29].

User Feedback Analysis in the Mobile Context. Mobile user feedback stored in different forms (e.g., user reviews, videos recorded, A/B testing strategies, etc.), can be used by developers to decide possible future directions of development or maintenance activities [8], [30]. Therefore, user feedback represents a valuable resource to evolve software applications [26]. As a consequence, the mobile development would strongly benefit from integrating User Feedback in the Loop (UFL) of the release cycle [9], [31], [32], [33], [34] (as highlighted by the **blue** elements/lines shown in Fig. 1 [19]), especially on the testing and maintenance activities. This has pushed the software engineering research community to study more effective automated solutions to *"enable the collection and integration of user feedback information in the development process"* [31]. The key idea of the techniques for user feedback analysis is to model [11], [8], [26], [13], classify
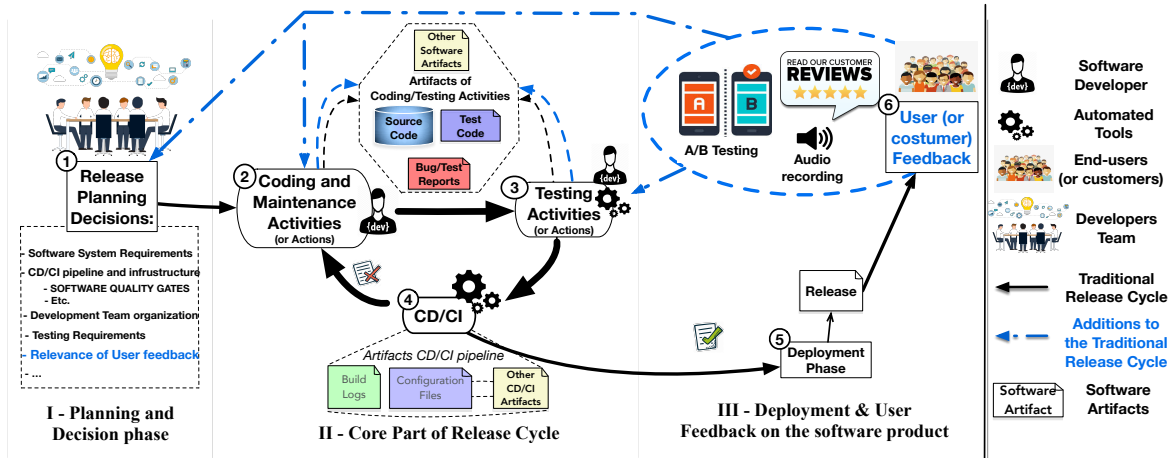
Fig. 1. Release Cycle

[11], [8], [13], summarize [34], [26], [17] or cluster [14] user feedback in order to integrate them into the release cycle. The research challenge is to effectively extract the useful feedback to actively support the developers to accomplish the release cycle tasks.

Mobile Testing and Source Code Localization based on User Feedback Analysis. User feedback analysis can potentially provide to developers information about the changes to perform to achieve a better user satisfaction and mobile app success. However, user reviews analysis alone is not sufficient to concretely help developers to continuously integrate user feedback information in the release cycle, and in particular (i) maintenance [13], [15], [8], [26] and (ii) testing [32], [9], [34], [33] activities. Recent research directions push the boundaries of user feedback analysis in the direction of *change-request file localization* [13], [15] and *user-oriented testing* (where user feedback is systematically integrated into the testing process) [9], [32], [33]. We will elaborate more on the literature in Section 6.

In this paper we focus on supporting developers with more advanced and reliable approaches to derive and cluster change-requests from user feedback, thus localizing the files to change [13], [15] to better support mobile maintenance tasks [13], [15], [8], [26].

3 APPROACH

As have been identified in the introduction, there are three major limitations in the existing approaches, i.e., low accuracy of classification and clustering because of low-quality of user reviews, difficulties in coping with the different vocabularies used to describe users' experience with the apps, and the existence of the lexicon gap between user reviews and software artefacts. RISING employs various techniques to mitigate these issues on which we will elaborate in this section. Outlined in Fig. 2, RISING consists of two major parts, i.e., *clustering* and *localization*. In the first branch, user reviews go through a series of textual processing, including fine-grained review segmentation , non-text removal, and lemmatization conversion; in the second branch, source code is first tagged with the commit messages, and comprehensive similarity scores are to be calculated, based on which the localization recommendations will be made. The

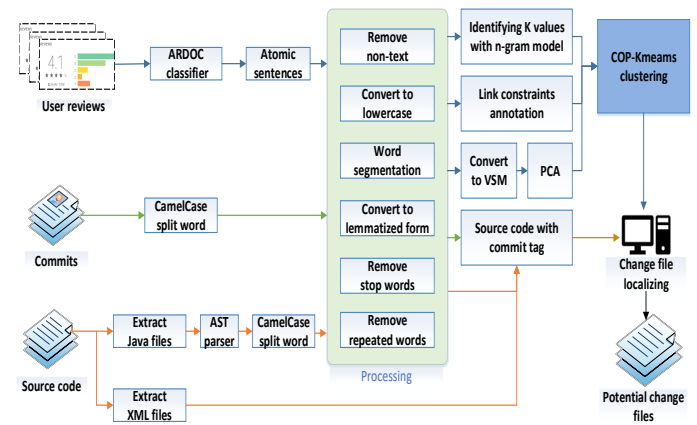details of these two parts will be given in Section 3.1 and Section 3.2 respectively.

Fig. 2. Approach Overview

3.1 User review clustering

Most user reviews are short textual snippets consisting of multiple sentences. These raw sentences may address different aspects of apps and need to be preprocessed before clustering. Based on their content, the reviews can mainly be classified into four categories, i.e., information giving, information seeking, feature request and problem discovery [8], [15]. In particular, "Information giving" denotes those sentences that inform or update users or developers about an aspect related to the app; "information seeking" denotes those which attempt to obtain information or help; "feature request" denotes those expressing ideas, suggestions or needs for improving the app's functionalities and performance; "problem discovery" denotes the sentences describing issues with the apps or their unexpected behaviors [8]. Since our aim is to identify those reviews which are directly relevant to apps' evolution, following [15] we only focus on the last two categories, i.e., feature request and problem discovery. To this end, we first employ ARDOC, a *user review classifier* developed in the previous work [8] which transforms user reviews into individual sentences

and then classifies these sentences into one of the aforementioned four categories. We then collect those sentences of the last two categories. ARDOC is built upon the functions of AR-Miner [11], which can filter noisy and uninformative user reviews in the first place. Such capability contributes another benefit to our approach.

To improve the accuracy of clustering, two tactics are employed, i.e., finer granularity review segmentation and textual processing, which will be elaborated in the following two subsections.

3.1.1 Fine-grained review segmentation

Clustering user reviews is usually conducted at the sentence level. We observe that, even inside an individual sentence, there still may be multiple topics involved which possibly address quite different concerns. As an example, one user review of AcDisplay reads "I wish there was a pattern lock feature and a camera shortcut for the lockscreen." Apparently, the user prefers two more features (a pattern lock and a shortcut utility). Moreover, for composite sentences in user reviews, if they contain adversative conjunctions such as 'but', the content after 'but' usually discloses the real information. As an example from K9-Mail[4], one user states that "This app is good, but it is lacking a key feature for anyone who uses mailing lists: Reply-To-List." In this case, for the purpose of localization, the content before 'but' is not informative at all, and may introduce noises to the follow-up process. As a result, we propose to have a more fine-grained text analysis. In particular, we split the composite sentences into ATOMIC ones each of *which expresses a single concern only*, and remove the irrelevant part of the sentence.

To achieve that, we employ a statistical parser from the Stanford NLP toolkit[5] to generate grammatical structures of sentences, i.e., phrase structure trees. We then traverse the leaf nodes of the phrase structure tree to determine whether or not the sentence contains conjunctions. Particularly, we focus on two types of conjunctions, i.e., copulative conjunctions and adversative conjunctions. The former (e.g., 'and', 'as well as' and 'moreover') mainly expresses the addition while the latter (e.g., 'but', 'yet') denotes contrasts.

For the first type, we recursively parse the nodes to identify the layer where the copulative conjunctions are located. We then obtain the copulative conjunction's sibling nodes. The two parts connected by the conjunction may be two sentences, two noun phrases, two verb phrases, etc. Given different conditions, we can generate two atomic sentences based on the parts which are connected by the conjunctions. As a concrete example, if the conjunction 'and' connects two noun objectives, then the two objectives are split as the only objective of each atomic sentence, but they share the same subjective and verb. (e.g. I wish there was a pattern lock feature and a camera shortcut for the lockscreen. → I wish there was a pattern lock feature for the lockscreen. I wish there was a camera shortcut for the lockscreen). If the conjunction 'and' connects two sentences, then the two sentences will be simply split into two atomic sentences (e.g. There are only 2 things I'd change for a 5 star review; I wish it had audio controls, and I wish there was a camera shortcut

from the lock screen. → There are only 2 things I'd change for a 5 star review; I wish it had audio controls. There are only 2 things I'd change for a 5 star review; I wish there was a camera shortcut from the lock screen).

For the second type, since we believe that the content after the adversative conjunctions convey the real information, we only preserve the leaf nodes after the conjunction nodes and simply leave out the other parts.

3.1.2 Textual processing

User reviews are generally informal and unstructured, mixing with typos, acronyms and even emojis [18]. The noisy data inevitably degrades the performance of clustering and localization which necessitates further textual processing. We first filter out the emoji characters and other punctuation content. Some emojis which were published as icons are stored in a text format, and their encoding appears as combination of question marks. Some others also use a combination of common punctuations, such as smiley faces. These patterns are matched by using regular expressions. Particularly, we propose two regular expressions to extract the pattern. The first one is $"\backslash\backslash p\{P\}\backslash\backslash s*"$. It removes all punctuations and replaces them with a space; the second one is $"[^a-zA-Z0-9\backslash\backslash s]*"$ which removes non-alphanumeric parts. Furthermore, we also convert all letters to lowercase uniformly.

Given the above steps, sentences are transformed into lists of words (i.e., tokens). We then use the Stanford NLP toolkit[6] to transform the inflected words to their lemmatization form. Here a dictionary-based instead of a rule-based approach is used to convert words into tokens which can avoid over-processing of words. (For instance, "images" is transformed correctly to image instead of to imag). User reviews may contain stopwords that could introduce noise for clustering and need to be removed. We note that the existing English stopword list cannot be well applied here for two reasons: first, a large number of user reviews contain irregular acronyms (e.g., asap–as soon as possible, cuz–cause) which cannot be processed by the existing stopword list. Second, some words are in the regular stopword list, but for specific apps, they may convey important information. For example, some words, such as "home", listed in strings.xml which encodes the string literals used by the GUI components, are of this kind. Therefore, we manually edit the English stopword list[7] accordingly (e.g., by adding some acronyms commonly used and removing some words that appear in strings.xml). We also delete the repeated words and the sentences which contain less than two words, because in short documents like user reviews, documents with less than two words hardly convey any useful information for evolution purpose.

Note that review segmentation is executed before textual processing, because the textual processing, which includes transforming the inflected words to their lemmatization form, removing stopwords, etc., would affect the grammatical structures of sentences which are crucial for review segmentation.

---

4. https://play.google.com/store/apps/details?id=com.fsck.k9
5. https://nlp.stanford.edu/software/lex-parser.shtml

6. https://stanfordnlp.github.io/CoreNLP/
7. The customized stopword list is also available online with the replication package.

3.1.3 User Review Clustering

Although ARDOC could classify reviews into "problem discovery" and "feature request", such coarse-grained classification provides limited guidance for developers when confronted with specific maintenance tasks. A more fine-grained approach is highly desirable. Firstly, it is not uncommon that the number of user reviews makes addressing every concern practically infeasible. Therefore, developers would like to identify the most common issues or requests raised by the end users, which are supposed to be treated with higher priority [14]. Secondly, not all user reviews are meaningful, especially in the problem discovery category. In practice, some complaints are actually caused by users' misunderstanding. By grouping similar issues together, such cases would be easier to be identified. Both of these motivate using clustering of pre-processed user reviews.

Construction of word-review matrix. We adopt the widely used Vector Space Model (VSM [35]) to represent the pre-processed texts. We fix a vocabulary Σ, each of which represents a feature in our approach. Let $n = |\Sigma|$, the size of the vocabulary, and m be the number of atomic sentences. We first construct a raw matrix $WR_{m \times n}$ where each entry $WR[r, w]$ is equal to the number of occurrences of the word w in the review r.

For each word $w \in \Sigma$, let f_w denote the occurrence of w in all reviews, i.e., $f_w := \sum_r WR[r, w]$, and we use logarithmically scaled document frequency ($df(w)$) as the weight assigned to the corresponding word:

$$df(w) = \log(1 + f_w)$$

Finally we can construct the scaled word-review matrix $R_{m \times n}$, where each entry

$$R[r, w] := WR[r, w] * df(w).$$

We remark that some related work uses traditional tf/idf as the weighting strategy [14], [36]. However, we use the document frequency (df) [35] for two reasons: (1) Clustering in our approach is done at the sentence level. Particularly, these sentences are short where an individual word usually occurs once, so tf would be meaningless for clustering in most cases. (2) In general, the purpose of idf is to give less weight to common words than to less common ones. In our preliminary experiments, we found that some words which only appear once do not make right suggestion to developers because they only represent personal feedback-/opinion and lack a general ground. However, they carry high idf simply because they are rare. On the other hand, those words which can indeed represent common issues encountered, or new functions required, by a majority of users carry low weights. To offset, we adopt df rather than more common tf/idf. Besides, one of the strong reasons to use idf is to reduce the weight of stop words which would have been removed in the data preprocessing steps.

Due to the large number of user reviews and the shortness nature of individual atomic sentences, the word vectors are of very high-dimension but very sparse. To reduce the dimension, we use the principal component analysis (PCA) technique [37], [38] which is one of the most widely used techniques for dimension reduction. Essentially, PCA replaces the original n features with an (usually much smaller) number r of features. The new features are linear combinations of the original ones that maximize the sample variance and try to make the new r features uncorrelated. The conversion between the two feature spaces captures the inherent variability of the data. Finally, the resulting matrix of $m \times r$ dimension gives rise to the data set D, which is the collection—as vectors—of rows of the matrix.

COP-Kmeans. After obtaining the vector models, we are in position to cluster similar texts based on their content. Existing approaches mainly employ automatic clustering algorithms to divide the reviews into multiple groups. However, we postulate that clustering would benefit from leveraging domain knowledge about the mobile app dataset. By investing limited human effort, the performance of clustering could be further boosted. For example, from AcDisplay, some reviews state "I do wish you could set a custom background, though." and "Would be nice to be able to customize the wallpaper too." As for traditional clustering algorithms, since the two keywords (i.e., background and wallpaper) are quite different in regular contexts, these two sentences would have a very low similarity score and thus be clustered into two different categories. However, professional developers would easily recognize that "wallpaper" and "background" refer to similar things in UI design, which suggests that the two reviews address the same issue and should be put into the same cluster.

On the other hand, some reviews might address quite irrelevant issues using the same words. For example, again in AcDisplay, two reviews are as below: "I would love the option of having different home screen.", and "First I'd like to suggest to disable that home button action because it turns the lock screen off ..., I hope you do it in next update.". These two reviews have completely different meanings, but since they both contain key words "home" and "screen", they are very likely to be clustered together by traditional clustering algorithms.

Domain knowledge of developers could potentially improve the precision of clustering, which has not been exploited by the traditional clustering algorithms. To remedy this shortcoming, we annotate a subset of instances with two types of link information, i.e., must-link and cannot-link constraints, as a priori knowledge and then apply the constrained K-means clustering technique [39]. The must-link constraints specify the instance pairs that discuss semantically similar or the same concerns, judged by professional developers with rich development expertise. Likewise, the cannot-link constraints specify the instance pairs that are *not* supposed to be clustered together. Besides, the must-link constraints define a transitive binary relation over the instances [39]. When making use of the constraints (of both kinds), we take a transitive closure over the constraints. (Note that although only the must-link constraints are transitive, the closure is performed over both kinds because, e.g., if d_i must link to d_j which cannot link to d_k, then we also know that d_i cannot link to d_k.)

To use the K-means family of algorithms, one needs to determine the value of the hyper-parameter K. There are some traditional, general-purpose approaches [40], [41], [42], but they did not take the topic distribution concerns into consideration so cannot provide a satisfactory solution

in our setting. We instead use a heuristic-based method to infer K. The heuristic is derived from the n-gram model of the review texts, since we believe the cluster number should strongly correlate to the topic distribution. N-gram denotes a sequence of n words in a particular sentence, which is a widely adopted statistical model to predict the occurrence of the n-th word using its previous $n-1$ words, based on the probability distribution of these words. Concretely, we obtain the 2-gram phrases of all user reviews. Then we merge the same phrases and record the number of occurrences of those phrases. If two phrases share the same word information, the less frequent phrase will be deleted. We also delete the phrases which occur once. K is then set to be the number of the remaining phrases. (2-gram is used as we empirically found that this yields the best performance.)

The COP-Kmeans algorithm takes the must-link and cannot-link dataset, K value and atomic sentence vectors as input and produces the clustering results. The pseudo-code is given in Algorithm 1. First, it randomly selects k samples $\{\mu_1, \ldots, \mu_k\}$ from the data set D as the initial cluster centers. Then, for each sample x_i in D, assign it to the closest cluster C_j such that it doesn't violate constraints in M and C. If no such cluster exists, an error message (line 4-21) would be returned. Then, for each cluster C_j, update its centroid by averaging all of the points $x \in C_j$ (line 22-24). This process iterates until the mean vectors no longer change.

3.2 Change file localization

For localizing potential change files, our approach combines the information from both the commit message and the source code. To get the commit messages of mobile apps, we exploit open-source projects to collect (i) the title, (ii) the description, (iii) the set of files involved, and (iv) the timestamp, for each commit. For source code, we mainly use the file path, class summary, method summary, method name and field declaration. Class summary and method summary can be extracted based on the javadoc tags. Method names and field declarations are parsed through abstract syntax tree (AST) analysis. In both cases, we remove non-textural information, split identifiers based on camel case styles, convert letters to lower case formats, stem, and remove stopwords/repetition words. Finally, the bag-of-words (BoW) model from the target app's source code and commit messages are generated respectively.

3.2.1 Tag Source Code Files

As mentioned earlier, we propose to leverage historical commit information to bridge the semantics gap between user reviews and source code. To this end, we first tag the source code with the historical change information. Particularly, for each commit, we extract the title, description, timestamps, and the involved file paths. From the file paths, we traverse the corresponding source code files in the project, and all the collected information, i.e., the title, description, and time stamps, is attached with the source file. As a result, each source code file can be regarded as a pair,

$$file = (code, commit)$$

where both $code$ and $commit$ are bag of words.

Input :
> The Data set $D = \{x_1, x_2, ..., x_m\}$;
> The Must-link constraints M;
> The Cannot-link constraints C;
> The K-value k;

Output:
> The clustering results $\{C_1, C_2, ..., C_k\}$;

1 Randomly select k samples $\{\mu_1, \mu_2, ..., \mu_k\}$ from D as the initial cluster centers;
2 **repeat**
3    $C_j = \varnothing (1 \le j \le k)$;
4   **for** $i = 1, 2, ..., m$ **do**
5   Calculate the distance between the sample x_i and each mean vector $\mu_j (1 \le j \le k) : d_{ij} = \|x_i - \mu_j\|_2$;
6      $\mathcal{K} = \{1, 2, ..., k\}$;
7   is_merged = false;
8   **while** $\neg$ *is_merged* **do**
9   Find the cluster closest to the sample x_i based on $\mathcal{K}$: $r = \arg\min_{j \in \mathcal{K}} d_{ij}$;
10   Detecting whether x_i is classified into cluster C_r violates constraints in M and C;
11   **if** $\neg$ *is_voilated* **then**
12          $C_r = C_r \cup \{x_i\}$;
13   is_merged=true;
14   **else**
15          $\mathcal{K} = \mathcal{K} \setminus \{r\}$;
16   **if** $\mathcal{K} = \varnothing$ **then**
17   **Break** Return error message;
18   **end**
19   **end**
20   **end**
21   **end**
22   **for** $j = 1, 2, ..., k$ **do**
23      $\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$;
24   **end**
25 **until** *Mean vectors are no longer updated*;

Algorithm 1: Constrained K-means Algorithm

Fig. 3 shows a commit example from AcDisplay. We extract title, description, timestamps (in blue rectangle) and relevant file paths (in red rectangle) information. All the files will be tagged with such information. Particularly, in this step we only consider the source code files and their related commit messages. The irrelevant commits (e.g., those do not involve source code files changes which are usually associated with '.html', '.properties', or '.md' files) are removed in the first place.

3.2.2 Localization

Similarity Computation. As mentioned earlier, due to the semantic gap between natural language and programming language, the direct similarity matching cannot precisely localize potential change files. We introduce the commit information to bridge the gap. Therefore, the similarity is attributed to the following two parts:

- the similarity between the user review and the code components extracted from one class of the target
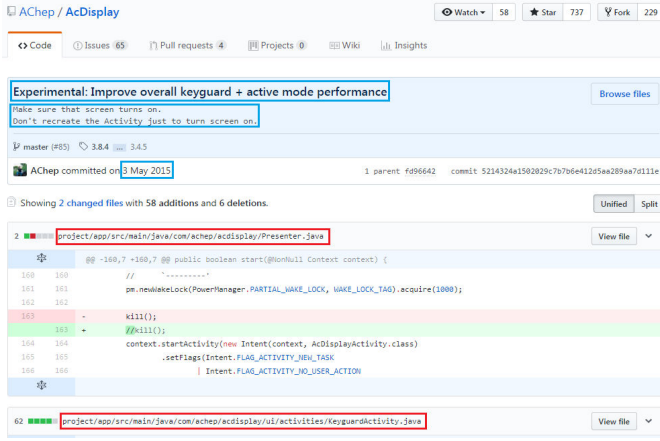
Fig. 3. Commit Message Illustration

app;

- the similarity between the user review and the commit tags of one class whose time stamps were earlier than the user review.

Palomba et al. [43] used the asymmetric Dice coefficient [35] to compute a textual similarity between a user review and a commit, as well as a textual similarity between a user review and an issue. Since user reviews are usually much shorter than source code files and commits, asymmetric Dice coefficient based similarity measures are usually employed (as opposed to other alternatives such as the cosine similarity or the Jaccard coefficient [44]). However, the original asymmetric Dice coefficient treats all the word equally and ignores those words which occur more frequently. Hence, we introduce a weighted asymmetric Dice coefficient as follows:

$$sim(r_i, code_j) = \frac{\sum\limits_{w_k \in W_{r_i} \cap W_{code_j}} df_{w_k}}{\min \left(\sum\limits_{w_r \in W_{r_i}} df_{w_r}, \sum\limits_{w_c \in W_{code_j}} df_{w_c} \right)} \quad (1)$$

where W_{r_i} is the set of words within the review r_i, W_{code_j} is the set of words within the code components of class j, df_{w_k} represents the document frequency (df) of the word w_k, and the $\min(\cdot, \cdot)$ function returns the argument whose value is smaller. In (1), we use df's value as the weight of the words. The intuition is that the more frequently a word occurs, the more important the word is.

The similarity between a user review and commit tags is computed analogously, by replacing W_{code_j} by W_{commit_j} as shown in (2), where W_{commit_j} is the set of words within the commit tags of class j.

$$sim(r_i, commit_j) = \frac{\sum\limits_{w_k \in W_{r_i} \cap W_{commit_j}} df_{w_k}}{\min \left(\sum\limits_{w_r \in W_{r_i}} df_{w_r}, \sum\limits_{w_c \in W_{commit_j}} df_{w_c} \right)} \quad (2)$$

Dynamic Interpolation Weights. The similarity score between user reviews and source code files is calculated by a linear combination of the similarity score between the

reviews and the source code contained in the files and the one between the reviews and the commit messages associated with the files (cf. Section 3.2.1). However, in the initial stage of the project life cycle, there is no enough commit information, a reminiscent of the cold-start problem. During the course of the project, commit messages accumulate. In light of this, we dynamically assign the weights to the two parts, inspired by dynamic interpolation weights [45], [46]:

$$sim(r_i, file_j) = \frac{L - \gamma}{L} sim(r_i, code_j) + \frac{\gamma}{L} sim(r_i, commit_j)$$

where γ is the number of common words which appear in both user review r_i and commit tags $commit_j$, and L is the number of words in user review r_i. We use L instead of the concentration parameter because we can determine the maximum number of γ. Based on the above equation, if $class_j$ does not have enough commit tags (when γ is small), then the code components of $class_j$ will be preferred, which can cope with the cold-start problem in which there are few commits or even no commits at the beginning of project life cycle. As the number of commit tags is growing, when γ is large, the commits will be preferred. This strategy could gradually increase the weight of commit messages during similarity calculation over time.

4 CASE STUDY

We collect the user reviews and commit messages of ten popular apps available from Google Play. The basic statistics of the selected projects are listed in Table 1. The selection

TABLE 1
Overview of selected apps

App Name	Category	Version	Comm. Msg. No.	Review No.
AcDisplay	Personalization	3.8.4	1096	8074
SMS Backup+	Tools	1.5.11	1687	1040
AnySoftKeyboard	Tools	1.9	4227	3043
Phonograph	Music&Audio	1.2.0	1470	6986
Terminal Emulator	Tools	1.0.70	1030	4351
SeriesGuide	Entertainment	44	9217	5287
ConnectBot	Communication	1.9.5	1714	4806
Signal	Communication	4.29.5	3846	6460
AntennaPod	Video Players	1.7.0	4562	3708
K-9 Mail	Communication	5.600	8005	8040
Total	-	-	36854	58775

criteria for Android apps are (i) open-source Android apps published on the Google Play market with version system and commit messages publicly accessible, and (ii) diversity in terms of app category (e.g., Personalization, Tools, Communication), size, and number of commit messages.

We developed two web scrapers to crawl the raw data: one is to extract user reviews from Google Play Store, and the other is to extract commit messages from GitHub. As for the source code, we download the latest version of the apps from GitHub. The version information is also shown in the table.

To evaluate how well our approach could help developers localize potential change files, we investigate the following two research questions.

RQ1: Does the constraint-based clustering algorithm perform better?

RQ2: Do commit messages improve the accuracy of localization?

RQ1. We implemented the approach and ran it on our dataset to address the above research questions. We chose ChangeAdvisor [15] as the baseline for two reasons. First-ly, ChangeAdvisor is the closest/most relevant approach with ours and the two address the same question largely; secondly, ChangeAdvisor is the state-of-the-art reference on clustering of user reviews in literature, and its supe-riority had been demonstrated compared to other similar approaches, such as BLUiR [47]. We observe that in the previous work [15], the authors did not distinguish the two general categories of user reviews, i.e., feature request and problem discovery, in the clustering process. Thus it is very likely that reviews of the two categories are clustered together. For example, in AcDisplay, the two reviews "The error was with my phone and other apps were crashing as well" and "It lacks UI customization like resizing the clock size or adding circle boarder ..." are grouped into the same cluster. Apparently, they express quite different concerns. To give a fair comparison, we differentiate the two categories, reuse the original prototype of ChangeAdvisor and the same parameter settings as published in [15].

TABLE 2
Apps' review information (FR: feature request; PD: problem discovery)

App Name	Review No.	FR No.			PD No.		
		Total	M-link	N-link	Total	M-link	N-link
AcDisplay	8074	1400	50	50	1437	50	50
SMS Backup+	1040	677	22	8	1425	32	21
AnySoftKeyboard	3043	280	25	3	290	16	6
Phonograph	6986	1094	63	28	960	42	53
Terminal Emulator	4351	248	13	9	372	10	28
SeriesGuide	5287	588	28	21	460	16	21
ConnectBot	4806	467	29	16	604	43	17
Signal	6460	629	36	23	792	32	45
AntennaPod	3708	336	25	21	359	16	22
K-9 Mail	8040	1018	65	36	1854	66	28
Total	58775	6737	356	215	8553	323	291

To annotate the constraints, we asked three researchers in software engineering (including the second author), to ran-domly select a small amount of reviews as samples from the whole dataset. These samples are inspected independently and the resulting constraints are cross validated. Table 2 shows the information of the input to the clustering stage. As mentioned before, we distinguish two categories, i.e., feature request (FR) and problem discovery (PD). The table also gives the human-annotated cluster constraints informa-tion of each app. In total, in the feature request category, the annotated must-link (M-link) takes up around 10.57% (356 pairs out of 6,737), and cannot-link (N-link) takes up around 6.38% (215 pairs out of 6,737); while in the problem discovery category, the percentage of must-link instances is around 7.55% (323 pairs out of 8,553), and of cannot-link is around 6.80% (291 pairs out of 8553). The marked instances are randomly selected from each app. In line with the metrics used in [15], we compare the *cohesiveness* and *separation* of clustering results between two approaches.

RISING incorporates domain knowledge to annotate the must-link and cannot-link information to the subset of user reviews, and leverages a heuristic-based method to infer the hyperparameter K. ChangeAdvisor directly applies the Hierarchical Dirichlet Process (HDP) algorithm to group the sentences of the reviews [48]. So we first need to compare the cluster numbers that the two approaches yield, and then the quality of each cluster.

TABLE 3
Clustering information

App Name	ChangeAdvisor		RISING	
	FR No.	PD No.	FR No.	PD No.
AcDisplay	10	12	83	129
SMS Backup+	12	10	71	107
AnySoftKeyboard	11	11	46	41
Phonograph	8	7	105	106
Terminal Emulator	11	8	38	45
SeriesGuide	11	11	59	44
connectBot	8	11	47	75
Signal	12	10	85	97
AntennaPod	10	8	43	57
K-9 Mail	6	10	113	154
Total	99	98	690	855

Table 3 presents the comparison of clustering between our approach and ChangeAdvisor. From the table, we can observe that, the number of clusters yielded by ChangeAd-visor and our approach varies a lot. The median cluster values of feature request and problem discovery categories in the studied apps by ChangeAdvisor are 10.5 and 10, respectively. However, in our approach, the median cluster values of the two categories are 65 and 86, respectively. Moreover, we found that, the clustering result is quite un-balanced in ChangeAdvisor. For example, in AcDisplay, the number of clusters of ChangeAdvisor (Problem Discovery Category) is 12, but 1142 out of 1437 sentences are grouped into one cluster, which takes up more than 79%; while in our approach, the largest cluster contains 339 instance sentences, which takes up around 23.6%. This highlights that the clusters generated by our approach are of more practical use to developers compared to the one generated by ChangeAdvisor.

To compare the quality of clusters obtained by two approaches, we use the Davis-Bouldin index (DBI), a widely adopted method to evaluate clustering algorithms [49], as a metric to assess the cohesiveness of intra-clusters and the separation of inter-clusters. This is an internal evaluation scheme, where the validation of how well the clustering has been done is made using quantities and features inherent to the dataset. DBI consists of two parts, one is the measure of scatter within the cluster, and the other is of separation between clusters.

For a cluster $C = \{X_1, \cdots, X_T\}$, the measure of scatter S_C of C is defined as follows.

$$S_C = \left(\frac{1}{T} \sum_{j=1}^{T} |X_j - A|^2 \right)^{1/2} \quad (3)$$

where T is the size of the cluster and A is the centroid of the cluster C.

The measure $M_{i,j}$ of separation between clusters C_i and C_j is defined as follows.

$$M_{i,j} = \|A_i - A_j\|_2 = \left(\sum_{k=1}^{n} |a_{k,i} - a_{k,j}|^2 \right)^{\frac{1}{2}} \quad (4)$$

where $a_{k,i}$ is the kth element of the centriod A_i of cluster C_i. DBI can then be defined as

$$DBI := \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} \frac{S_i + S_j}{M_{i,j}} \quad (5)$$

where K is the number of clusters.

The DBI value is a standard measurement of the quality of clustering, i.e., the cohesiveness of intra-cluster and the separation of inter-clusters. Typically, a better clustering algorithm admits a *lower* value of DBI.

ChangeAdvisor uses HDP algorithm for clustering, which accepts text objects as input. To enable the adoption of DBI, we need to convert the dataset of ChangeAdvisor into a vector format. In order to ensure a fair comparison, we use the same method to convert the transformed sentences into the vector representation as in our approach, as detailed in Section 3.1.

TABLE 4
DBI results comparison

App Name	ChangeAdvisor		RISING	
	FR	PD	FR	PD
AcDisplay	0.493	0.361	0.035	0.020
SMS Backup+	0.321	0.444	0.047	0.042
AnySoftKeyboard	0.357	0.342	0.050	0.050
Phonograph	0.514	0.693	0.031	0.029
Terminal Emulator	0.300	0.557	0.105	0.060
SeriesGuide	0.440	0.303	0.075	0.057
ConnectBot	0.606	0.479	0.080	0.027
Signal	0.317	0.391	0.055	0.027
AntennaPod	0.447	0.548	0.048	0.046
K-9 Mail	0.928	0.538	0.040	0.022
Average	0.472	0.466	0.057	0.038

The results are summarized in Table 4. From the table, we can observe that our approach yields a considerably better DBI result compared with ChangeAdvisor. The average DBI values of feature request and problem discovery by ChangeAdvisor are 0.472 and 0.466 respectively; while by our approach, the average values are 0.057 and 0.038 respectively.

In addition, to demonstrate the effectiveness of review segmentation, we conduct ablation study for RISING and RISING without review segmentation. The effectiveness is also measured by DBI values. The results are shown in Table 5 which suggest that the review segmentation operation has indeed improved the clustering effect by reducing the average DBI values of feature request and problem discovery from 0.167 to 0.057 and 0.159 to 0.038, respectively.

TABLE 5
DBI results comparison for the review segmentation

App Name	RISING (no segmentation)		RISING	
	FR	PD	FR	PD
AcDisplay	0.235	0.247	0.035	0.020
SMS Backup+	0.094	0.110	0.047	0.042
AnySoftKeyboard	0.159	0.184	0.050	0.050
Phonograph	0.178	0.131	0.031	0.029
Terminal Emulator	0.156	0.120	0.105	0.060
SeriesGuide	0.159	0.125	0.075	0.057
ConnectBot	0.250	0.230	0.080	0.027
Signal	0.129	0.113	0.055	0.027
AntennaPod	0.151	0.164	0.048	0.046
K-9 Mail	0.159	0.165	0.040	0.022
Average	0.167	0.159	0.057	0.038

To further evaluate the quality of clustered reviews, we hired mobile app developers as external evaluators. Specifically, we contacted over 10 professional mobile developers from our personal network applying the following participants constraints: they were not familiar with the actual apps under study and they had at least three years of mobile development experience. This selection process allowed us to receive feedback by only external participants having the adequate experience to the context of the tasks. Considering also the developers availability and willingness to perform the study, we hired in total five developers, three to evaluate the quality of clustered reviews, while all of them participated in the evaluation tasks of RQ2 (as reported later in the paper). Thus, we asked the three of the mobile developers to look into the clustered sentences and to assess the coherence of content of each individual cluster as well as the semantics separation of different clusters. The assessment is given in Likert scale grades: "exactly related topics" (5), "mostly related topics" (4), "basically related topics" (3), "just a few related topics" (2), and "not relevant topics" (1). Different from the evaluation method in [15], we evaluate all the clusters, and calculate the average value as the final result.

The results are shown in Table 6. From the table, we observe that RISING yields a better value of Likert scale compared with ChangeAdvisor. The average values of feature request and problem discovery categories by ChangeAdvisor are 2.07 and 1.94 respectively; while by RISING, the average values are 4.20 and 4.26 respectively.

TABLE 6
Likert results comparison

App Name	ChangeAdvisor		RISING	
	FR	PD	FR	PD
AcDisplay	2.22	2.12	4.30	4.29
SMS Backup+	1.93	2.03	4.23	4.26
AnySoftKeyboard	2.50	2.47	4.23	4.09
Phonograph	2.35	1.55	4.40	4.35
Terminal Emulator	2.18	2.15	3.83	4.17
SeriesGuide	2.17	1.74	4.22	4.29
ConnectBot	1.43	2.05	4.20	4.35
Signal	1.96	1.70	4.26	4.31
AntennaPod	2.08	1.67	4.17	4.25
K-9 Mail	1.87	1.92	4.11	4.25
Average	2.07	1.94	4.20	4.26

The above objective and subjective measures answer RQ1 that our constraints-based clustering method, aided by more intensive (automated) data preprocessing and marginal human annotation efforts, could greatly boost the clustering performance.

RQ2. To address RQ2, we need to judge whether commit messages improve the accuracy of localization. In the experiments, we use the same ten Android apps in the preceding step (cf. Table 1). As the first step, we need to obtain the ground-truth for the localization result which requires human examination. As mentioned in the design of RQ1, to reduce personal bias, we hired two additional mobile app developers, both of whom have over three years of development experience as well. The five evaluators were asked to check the localization result individually. Only after the individual evaluation, they discussed jointly trying to reach a consensus. The discussion happened via conference calls involving all the five evaluators (as physical meetings turned out to be difficult to organize). During the meeting, the evaluator of the work led the discussion. All participants quickly separated the cases in which there was a clear agreement from those in which no agreement can be quickly drawn. Extensive discussions then focused on those more

difficult cases until a consensus was reached. Results of this evaluation process serves as the ground-truth of RQ2.

As the next step, we apply ChangeAdvisor and RISING to the reviews to compare the results returned from them against the ground-truth results from the previous step. For each category in each app, we randomly select 12-18 user reviews and then apply ChangeAdvisor and RISING separately to these sample reviews. (In ChangeAdvisor, a sample size of around 10 for each app category was used. We generally follow the guideline of the baseline.) Overall, we select 297 (150 + 147) user reviews from these 10 apps. RISING could return potential change files in all the cases. However, ChangeAdvisor could only give outputs when inputting 132 (82 + 50) user reviews, less than 50% of RISING. The details of the localizability comparison of the two approaches in the studied apps are given in Table 7.

To evaluate the localization result, we employed the Top-k accuracy and NDCG as metrics which are commonly used in recommendation systems [50], [51], [52]. Top-k accuracy can be calculated as

$$\text{Top-k } accuracy(U) = \frac{\sum\limits_{u \in U} \text{isCorrect}(u, \text{Top-k})}{|U|}$$

where U represents the set of all user feedbacks and the isCorrect(r, Top-k) function returns 1 if at least one of Top-k source code files actually is relevant to the user feedback u; and returns 0 otherwise.

Table 8 reports the Top-k accuracy of ChangeAdvisor and RISING for each of the considered apps where the value k is set to be 1, 3 and 5. From the table, we can observe that, in most cases, RISING substantially outperforms ChangeAdvisor in terms of Top-k hitting. On average, for feature request category, the Top-1, Top-3, Top-5 values can be improved from 44.64% to 76.74%, 70.54% to 91.77%, and 76.65% to 98.00% respectively; for problem discovery category, the Top-1, Top-3, Top-5 values are improved from 48.50% to 76.04%, 65.08% to 93.84%, and 76.00% to 98.04% respectively.

NDCG is defined as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \quad \left(DCG@k = \sum_{i=1}^{k} \frac{r_i}{\log_2(i+1)} \right)$$

where $r_i = 1$ if the i-th source code file is related to the user feedback, and $r_i = 0$ otherwise. IDCG is the ideal result of DCG, which means all related source code files are ranked higher than the unrelated ones. For example, if an algorithm recommends five source code files in which the 1-st, 3-rd and 5-th source code files are related, the results are represented as $\{1, 0, 1, 0, 1\}$, whereas the ideal result is $\{1, 1, 1, 0, 0\}$.

Table 9 reports the NDCG values of ChangeAdvisor and RISING for each of the considered apps where, similarly, the value k is set to be 1, 3 and 5. Based on the table, we observe that, in most cases of the studied apps, the NDCG value of RISING is greater than that of ChangeAdvisor, which indicates a better performance. On average, the NDCG@1, NDCG@3 and NDCG@5 values of ChangeAdvisor in the problem discovery category are 48.50%, 46.37%, and 59.69% respectively. In contrast, the corresponding values of RISING in this category are 76.03%, 74.46%, and 85.95%

respectively. In feature request category, the NDCG@1, NDCG@3 and NDCG@5 values of ChangeAdvisor are 44.64%, 53.52%, 60.94% respectively; while the values of RISING in this category are 76.74%, 74.11%, and 85.79% respectively.

To further demonstrate the impact of commit messages for localization, we localize the user reviews by merely using source code information. The localization results are shown in Table 10. By comparing the localization results between RISING (Source Code) and RISING, we can observe that, for the feature request category, the Top-1, Top-3, Top-5 values are improved from 67.53%, 83.91% and 91.48% to 76.74%, 91.77% and 98.00%, and the NDCG@1, NDCG@3, NDCG@5 are improved from 67.53%, 65.96%, 78.09% to 76.74%, 74.11%, 85.79%; for the problem discovery category, the Top-1, Top-3, Top-5 values are improved from 63.67%, 85.52%, 92.46% to 76.04%, 93.84%, 98.04%, and the NDCG@1, NDCG@3, NDCG@5 are improved from 63.67%, 64.76%, 77.20% to 76.03%, 74.46%, 85.95%. The results demonstrate that commit messages do contribute to improving the localization performance.

The experiment results answer RQ2 that, in terms of the localization accuracy, our approach which exploits commit messages to fill the lexicon gap could improve the performance greatly.

TABLE 7
Overview of Localization

App Name	ChangeAdvisor		RISING	
	FR No.	PD No.	FR No.	PD No.
AcDisplay	6	2	17	15
SMS Backup+	7	2	14	15
AnySoftKeyboard	7	5	14	13
Phonograph	9	4	18	16
Terminal Emulator	4	6	15	16
SeriesGuide	14	10	15	14
ConnectBot	10	4	14	16
Signal	7	8	16	14
AntennaPod	10	5	15	14
K-9 Mail	8	4	12	14
Sum	82	50	150	147

5 DISCUSSION

Identifying meaningful user reviews from app markets is a non-trivial task, since a majority of them are not informative. Furthermore, to link and localize potential change files based on those meaningful feedbacks would be highly desirable for software developers. Compared with the state-of-the-art baseline work like ChangeAdvisor, RISING could give more fine-grained clustering results and more accurate localization performance. Specifically, after a closer qualitative analysis of the clusters generated by both approaches we found interesting characteristics concerning the clusters generated by RISING, when compared to the one of ChangeAdvisor. First of all, ChangeAdvisor tends to discard a wide majority of reviews, clustering a small subset of informative reviews. For instance, if we consider the app AcDisplay, we observe that the total number of reviews included in all generated clusters, considering both feature requests and problem discovery categories, is 150. This value is drastically smaller than the amount of informative reviews composing the clusters generated by RISING for this app, i.e., 2,053 reviews. As a consequence, the number of clusters for ChangeAdvisor tends to be very small for most

TABLE 8
Top-k Accuracy of Localization

App Name	ChangeAdvisor						RISING					
	FR			PD			FR			PD		
	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
AcDisplay	0.8333	0.8333	0.8333	0.5000	0.5000	0.5000	1.0000	1.0000	1.0000	0.8000	1.0000	1.0000
SMS Backup+	0.2857	0.5714	0.7143	0.5000	0.5000	0.5000	0.7143	1.0000	1.0000	0.8000	1.0000	1.0000
AnySoftKeyboard	0.7143	0.8571	0.8571	0.8000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9231	1.0000	1.0000
Phonograph	0.6667	0.7778	0.7778	0.5000	0.7500	1.0000	0.8333	1.0000	1.0000	0.9375	0.9375	1.0000
Terminal Emulator	0.2500	0.7500	0.7500	0.5000	0.8333	1.0000	0.8667	0.9333	0.9333	0.6875	0.9375	0.9375
SeriesGuide	0.2857	0.6429	0.7857	0.4000	0.7000	0.8000	0.6000	0.8000	0.8667	0.7143	1.0000	1.0000
ConnectBot	0.3000	0.7000	0.7000	0.7500	0.7500	0.7500	0.6429	0.7857	1.0000	0.8125	0.9375	0.9375
Signal	0.4286	0.5714	0.5714	0.2500	0.3750	0.7500	0.5000	0.8750	1.0000	0.6429	0.8571	1.0000
AntennaPod	0.2000	0.6000	0.8000	0.4000	0.6000	0.8000	0.6000	0.8667	1.0000	0.7857	0.9286	0.9286
K-9 Mail	0.5000	0.7500	0.8750	0.2500	0.5000	0.5000	0.9167	0.9167	1.0000	0.5000	0.7857	1.0000
Average	0.4464	0.7054	0.7665	0.4850	0.6508	0.7600	0.7674	0.9177	0.9800	0.7604	0.9384	0.9804

TABLE 9
NDCG@k of Localization

App Name	ChangeAdvisor						RISING					
	FR			PD			FR			PD		
	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5
AcDisplay	0.8333	0.7675	0.8012	0.5000	0.3520	0.4427	1.0000	0.8843	0.9565	0.8000	0.7996	0.8973
SMS Backup+	0.2857	0.3733	0.4726	0.5000	0.3066	0.4386	0.7143	0.6598	0.8180	0.8000	0.6837	0.8706
AnySoftKeyboard	0.7143	0.7672	0.8000	0.8000	0.6922	0.8692	1.0000	0.8630	0.9611	0.9231	0.8989	0.9507
Phonograph	0.6667	0.6169	0.7021	0.5000	0.5180	0.7303	0.8333	0.8010	0.9120	0.9375	0.7896	0.9242
Terminal Emulator	0.2500	0.5044	0.5705	0.5000	0.6087	0.7509	0.8667	0.7994	0.8872	0.6875	0.6753	0.8127
SeriesGuide	0.2857	0.4422	0.5443	0.4000	0.3981	0.5572	0.6000	0.6013	0.7328	0.7143	0.7538	0.8666
ConnectBot	0.3000	0.4429	0.5279	0.7500	0.7500	0.7500	0.6429	0.6456	0.7962	0.8125	0.8265	0.8877
Signal	0.4286	0.4885	0.4885	0.2500	0.2984	0.4874	0.5000	0.6676	0.7693	0.6429	0.7113	0.8188
AntennaPod	0.2000	0.4028	0.5406	0.4000	0.3860	0.5564	0.6000	0.6756	0.8246	0.7857	0.7262	0.8232
K-9 Mail	0.5000	0.5460	0.6467	0.2500	0.3266	0.3859	0.9167	0.8131	0.9208	0.5000	0.5808	0.7430
Average	0.4464	0.5352	0.6094	0.4850	0.4637	0.5969	0.7674	0.7411	0.8579	0.7603	0.7446	0.8595

TABLE 10
Localization of RISING (Source Code Only)

App Name	RISING(sourceCodeOnly)											
	FR			PD			FR			PD		
	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5
AcDisplay	0.9412	1.0000	1.0000	0.7333	0.9333	1.0000	0.9412	0.8631	0.9394	0.7333	0.7124	0.8547
SMS Backup+	0.5714	1.0000	1.0000	0.6000	0.9333	0.9333	0.5714	0.6567	0.7974	0.6000	0.6387	0.7676
AnySoftKeyboard	0.9286	0.9286	1.0000	0.8462	0.8462	1.0000	0.9286	0.7975	0.9189	0.8462	0.7396	0.8717
Phonograph	0.7778	0.8889	0.9444	0.8125	0.9375	1.0000	0.7778	0.7009	0.8359	0.8125	0.7591	0.8807
Terminal Emulator	0.6667	0.8000	0.8667	0.6875	0.8125	0.8750	0.6667	0.6536	0.7604	0.6875	0.6658	0.7618
SeriesGuide	0.4667	0.6000	0.7333	0.4286	0.8571	0.9286	0.4667	0.5020	0.5936	0.4286	0.6107	0.7246
ConnectBot	0.5714	0.8571	0.9286	0.6875	0.8750	0.9375	0.5714	0.6202	0.7668	0.6875	0.6666	0.7894
Signal	0.5625	0.7500	0.8750	0.6429	0.7857	0.8571	0.5625	0.5224	0.6896	0.6429	0.6154	0.7309
AntennaPod	0.6000	0.7333	0.8000	0.5714	0.8571	0.9286	0.6000	0.5532	0.6810	0.5714	0.5815	0.7349
K-9 Mail	0.6667	0.8333	1.0000	0.3571	0.7143	0.7857	0.6667	0.7261	0.8262	0.3571	0.4858	0.6032
Average	0.6753	0.8391	0.9148	0.6367	0.8552	0.9246	0.6753	0.6596	0.7809	0.6367	0.6476	0.7720

projects, compared to our approach, as reported in Table 3. On the other hand, the sizes of the clusters generated by RISING tend to be more balanced, i.e., not too large. Indeed, for RISING, the average number of reviews for each cluster tends to be smaller (11-12 reviews on average v.s. 18-24 for ChangeAdvisor). In our experiments, we also observe that, distinct runs of ChangeAdvisor give noticeably different clustering results, making the clustering less stable or deterministic and less accurate (they tend to be less semantically related). The clustering result of RISING is much more stable. To see this, we made two new runs of RISING and ChangeAsdvisor respectively for the app AcDisplay. The results show that the size of the clusters generated by RISING tend to be still very balanced, i.e., not too large, keeping a similar number of clusters, and almost the same average number of reviews for each cluster (11-12 reviews in average). Vice versa, for ChangeAdvisor we observe for the same app, still very large clusters. Interestingly, in the ChangeAdvisor re-runs, the number of clusters was in one case reduced and in another increased, observing in some cases higher or similar averages number of reviews for each cluster. In the localization phase, RISING leverages the commit information to bridge the lexicon gap. Note that commit history contains all the relevant files for the change transaction including not only source files but also configuration related files (such as XML files). Our approach is thus advantageous over other similar approaches to be able to locate multiple files which are necessary for problem fix or feature request. In contrast, ChangeAdvisor does not take into account the association between files, which would miss, for instance, configuration files.

Threats to Validity

Internal validity. We conclude that, with domain knowledge, marginal human effort could greatly boost the clustering performance. Such effectiveness has already been demonstrated in various scenarios [53], [54]. In the clustering phase, we only annotate a small portion of the whole re-

view set with must-link and cannot-link, reducing the threat of over-fitting. The recovery of missing traceability links between various software artefacts has also been actively studied in the literature [55]. Commit messages contain rich information about the change history and the motivation of the change itself. Thus the information could bring benefits to bridge the vocabulary gap between professional developers and ordinary users. Another threat arises from the selection bias of the dataset. In our experiments, we strive to reuse the same apps in the baseline work as many as possible. To reduce the noise from the raw data and bias in the result, we take the standard measures to pre-process the raw texts, and include five professional developers with over three years of mobile app development experience to solve subjective conflicts.

External validity. In our case study, we deliberately selected 10 apps across different categories instead of being limited within a narrow domain. To give a fair comparison, we use a combination of multiple evaluation metrics, including both objective and subjective ones. Similar to other empirical studies, no evidence could theoretically prove our approach can always accurately localize change files in all scenarios. But we believe that, since our approach is open to different scenarios, domain knowledge could be leveraged via new constraints and heuristics incorporated into our approach which could improve the clustering and localization performance as well in the new dataset.

Finally, even if the underlying assumption of our work is that commit messages contribute to increase the number of common terms matching the use review vocabulary, there could be a mismatch between commit message vocabulary and user review vocabulary too. For instance, commit messages terms like bug ids (*'this commit fixe the bug X'*) are not present in user reviews. Hence, for future work, we plan to investigate the potential mismatch between such vocabularies, with the goal to improve the results of our approach.

6 RELATED WORK

The concept of app store mining was introduced by Harman *et al.* [7] in 2012, and several researchers focused on mining mobile apps and app store data to support developers during the maintenance and evolution of mobile applications, with the goal to achieve a higher app success [2].

6.1 The Role of User Feedback Analysis in the Mobile App Success

App Rating & App Success. Previous research widely investigated the relationship between the rating and a particular characteristic (or feature) of mobile applications [56], [57], [58], [59], [60]. Recent research efforts have been devoted to investigating the reliability of app rating when used as a proxy to represent user satisfaction. For example, Luiz *et al.* [61] proposed a framework performing sentiment analysis on a set of relevant features extracted from user reviews. Despite the star rating was considered to be a good metric for user satisfaction, their results suggest that sentiment analysis might be more accurate in capturing the sentiment transmitted by the users. Hu *et al.* [62] studied the

consistency of reviews and star ratings for hybrid Android and iOS apps discovering that they are not consistent across different app markets. Finally, Catolino [63] preliminarily investigated the extent to which source code quality can be used as a predictor of commercial success of mobile apps.

User Feedback Analysis & App Success. Several approaches have been proposed with the aim to classify useful user reviews for app success. AR-MINER [11] was the first one able to classify informative reviews. Panichella *et al.* adopted natural language processing and text and sentiment analysis to automatically classify user reviews [8], [64] according to a User Review Model (URM). Gu and Kim [65] proposed an approach that summarizes sentiments and opinions of reviews.

Following the general idea of incorporating user feedback into typical development process, Di Sorbo *et al.* [16], [17] and Scalabrino *et al.* [14], [66] proposed SURF and CLAP, two approaches aiming at recommending the most important reviews to take into account while planning a new release of a mobile application. CLAP improves AR-MINER by clustering reviews into specific categories (*e.g.,* reports of security issues) and by learning from the app history (or from similar apps) which reviews should be addressed [66]. SURF proposed a first strategy to automatically summarize user feedback in more structured and recurrent topics [17], [34] (e.g., GUI, app pricing, app content, bugs, etc.). Finally, Palomba *et al.* [15], inspired by the work by Scalabrino *et al.*, proposed ChangeAdvisor, a tool that cluster user reviews of mobile applications. In this paper we considered as baseline ChangeAdvisor since, similarly to our approach, it is based on a clustering approach for user review feedback. In evaluating our approach, we discovered that ChangeAdvisor tends to generate rather different user review clusters with the same study setting and user review data, which highlights higher reliability of our approach compared to this state-of-the-art tool.

6.2 Information Retrieval in SE & the Mobile Context

Information Retrieval techniques have been widely adopted to handle several SE problems. Specifically, strategies for recovery traceability links between textual artefacts and the source code were widely studied in the past [55], [67]. In the same way, several approaches to locating features in the source code [68], and tracing informal textual documentation, such as e-mails [69], [70], [71], forum discussions [72], [73], [74], and bug reports [47] to the source code have been proposed. However, as previously demonstrated by Panichella *et al.* [75], the configuration used to set the clustering algorithm is an important component of topic modeling techniques used in several traceability recovery approaches, and an optimal choice of the parameters generally results in better performance.

Duan *et al.* [76] proposed a consensus-based approach to constrained clustering requirement documents. This is a different software engineering task than ours, but both approaches employ the semi-supervised clustering technique at a high level. In [76], consensus clustering is firstly performed to generate an ensemble of multiple clusters, and then a voting mechanism is applied to select the constraints. In our approach, we leverage domain knowledge to help

generate the constraints. In the context of mobile computing research, two pieces of work are closer to ours. Ciurumelea *et al.* [77], [36] employed machine learning techniques for the automatic categorization of user reviews on a two-level taxonomy adopting a modified Version of Vector Space Model (VSM) to automatically link user reviews to code artefacts. Similarly, Palomba *et al.* [15] cluster user reviews of mobile applications and suggest the source-code artefacts to maintain. We mainly compare our approach against ChangeAdvisor [15] as, similar to our approach, it leverages clustering approaches for user review feedback analysis and IR-based methods for suggesting the source-code artefacts to maintain according to user change-requests. However, different from the work [15] and the work [77], [36], the similarity score between user reviews and source code files in our approach is calculated by a linear combination of the similarity score between the reviews and the source code and the similarity score between reviews and the commit messages. Indeed the work [15], [77], [36] mainly relies on textual analysis techniques such as VSM and Dice to compute directly the similarity among reviews and source code files. Moreover, the word-review matrix is build on a subset of textual features, selected using PCA. This allows to select more meaningful features from user review textual content.

7 CONCLUSIONS AND FUTURE WORK

User reviews convey client-side requirements for mobile app products. Accurate recovery of the user concerns and automatic localization of relevant source code based on these feedbacks is of great importance to facilitate rapid development. In this paper, we present an approach to localize potential change files based on user reviews for mobile applications. We conducted experiments on 10 popular mobile apps and used a comprehensive set of metrics to assess the performance of our approach. Experimental results show that our approach greatly outperforms the state-of-the-art baseline work.

In the immediate future work, we plan to develop a comprehensive environmental support for change file localization so as to give a better applicability of our approach. Moreover, our current case studies are all about open-source apps, while our future plan includes collaboration with commercial app developers and applying our approach to these industry cases.

ACKNOWLEDGEMENTS

REFERENCES

[1] M. Dehghani, "An assessment towards adoption and diffusion of smart wearable technologies by consumers: the cases of smart watch and fitness wristband products," ser. CEUR Workshop Proceedings, vol. 1628. CEUR-WS.org, 2016.

[2] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2016.

[3] Statista. (2018, Mar.) Number of apps available in leading app stores as of october 2018. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[4] A. Boxall, "There are 12 million mobile developers worldwide, and nearly half develop for android first," *Retrieved from businessofapps: http://www. businessofapps. com/12-million-mobile-developers-worldwide-nearly-half-develop-android-first*, 2016.

[5] F. by Robert Stroud, "Devops: From unicorns to mainstream," https://go.forrester.com/blogs/17-07-09-devops_from_unicorns_to_mainstream/.

[6] Engine Yard - by Christopher Rigor , "How the role of devops will change in 2018," https://www.engineyard.com/blog/how-the-role-of-devops-will-change-in-2018.

[7] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, June 2012, pp. 108–111.

[8] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME '15, 2015, pp. 281–290.

[9] G. Grano, A. Ciurumelea, F. Palomba, S. Panichella, and H. Gall, "Exploring the integration of user feedback in automated testing of android applications," in *Software Analysis, Evolution and Reengineering, 2018 IEEE 25th International Conference on*, 2018.

[10] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 224–234.

[11] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Arminer: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 767–778.

[12] S. Mcilroy, W. Shang, N. Ali, and A. E. Hassan, "User reviews of top mobile apps in apple and google app stores," *Communications of the ACM*, vol. 60, no. 11, pp. 62–67, 2017.

[13] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *SANER*. IEEE Computer Society, 2017, pp. 91–102.

[14] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16, 2016, pp. 14–24.

[15] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. C. Gall, F. Ferrucci, and A. D. Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proceedings of the 39th International Conference on Software Engineering*, 2017, pp. 106–117.

[16] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 499–510.

[17] A. D. Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora, "SURF: summarizer of user reviews feedback," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, 2017, pp. 55–58.

[18] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 749–759.

[19] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto

for agile software development," 2001. [Online]. Available: http://www.agilemanifesto.org/

[20] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk.* Addison-Wesley, 2007.

[21] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, "Devops in regulated software development: Case medical devices," in *39th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track, ICSE-NIER 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017, pp. 15–18.

[22] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[23] M. R. Islam and M. F. Zibran, "Insights into continuous integration build failures," in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017, pp. 467–470.

[24] C. Ziftci and J. Reardon, "Who broke the build? Automatically identifying changes that induce test failures in continuous integration at google scale," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017*, 2017, pp. 113–122.

[25] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. D. Penta, and S. Panichella, "A tale of CI build failures: An open source and a financial organization perspective," in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*, 2017, pp. 183–193.

[26] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in My app? Summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 499–510. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950299

[27] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman, "App store effects on software engineering practices," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[28] M. Nayebi, H. Farrahi, and G. Ruhe, "Which version should be released to app store?" in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*, 2017, pp. 324–333. [Online]. Available: https://doi.org/10.1109/ESEM.2017.46

[29] M. Nayebi, B. Adams, and G. Ruhe, "Release practices for mobile apps - what do users and developers think?" in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, 2016, pp. 552–562. [Online]. Available: https://doi.org/10.1109/SANER.2016.116

[30] E. Noei, D. A. Da Costa, and Y. Zou, "Winning the app production rally," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 283–294.

[31] M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016*, 2016, pp. 21–32.

[32] G. Grano, A. Di Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: A dataset for software evolution and quality improvement," in *Proceedings of the 2Nd ACM SIGSOFT International Workshop on App Market Analytics*, ser. WAMA 2017, 2017, pp. 8–11.

[33] L. Pelloni, G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall, "Beccloma: Augmenting stack traces with user review information," in *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 522–526.

[34] S. Panichella, "Summarization techniques for code, change, testing, and user feedback (invited paper)," in *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*, C. Artho and R. Ramler, Eds. IEEE, 2018, pp. 1–5.

[35] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.

[36] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *IEEE 24th International Conference on Software Anal-*

[37] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu, "Dimensionality reduction for k-means clustering and low rank approximation," *CoRR*, vol. abs/1410.6801, 2014.

[38] C. H. Q. Ding and X. He, "K-means clustering via principal component analysis," in *Machine Learning, Proceedings of the Twenty-first International Conference (ICML)*, 2004.

[39] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl *et al.*, "Constrained k-means clustering with background knowledge," in *ICML*, vol. 1, 2001, pp. 577–584.

[40] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, 2003, pp. 281–288.

[41] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a dataset via the gap statistic," vol. 63, pp. 411–423, 2000.

[42] D. Pelleg and A. W. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, 2000, pp. 727–734.

[43] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 291–300.

[44] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[45] Z. Tu, Z. Su, and P. T. Devanbu, "On the localness of software," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, 2014, pp. 269–280.

[46] K. Knight, "Bayesian Inference with Tears," Tech. Rep., 2009.

[47] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 345–355.

[48] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Sharing clusters among related groups: Hierarchical dirichlet processes," in *Advances in neural information processing systems*, 2005, pp. 1385–1392.

[49] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, April 1979.

[50] C. Tantithamthavorn, R. Teekavanich, A. Ihara, and K. Matsumoto, "Mining A change history to quickly identify bug locations : A case study of the eclipse project," in *IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013, Pasadena, CA, USA, November 4-7, 2013 - Supplemental Proceedings*, 2013, pp. 108–113.

[51] C. Tantithamthavorn, A. Ihara, and K. Matsumoto, "Using co-change histories to improve bug localization performance," in *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2013, Honolulu, Hawaii, USA, 1-3 July, 2013*, 2013, pp. 543–548.

[52] X. Li, H. Jiang, Y. Kamei, and X. Chen, "Bridging semantic gaps between natural languages and apis with word embedding," *CoRR*, vol. abs/1810.09723, 2018. [Online]. Available: http://arxiv.org/abs/1810.09723

[53] M. Bilenko, S. Basu, and R. J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 11.

[54] S. Basu, I. Davidson, and K. Wagstaff, *Constrained clustering: Advances in algorithms, theory, and applications*. CRC Press, 2008.

[55] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[56] L. Corral and I. Fronza, "Better code for better apps: A study on source code quality and market success of android applications," in *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '15, 2015, pp. 22–32.

ysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017, 2017, pp. 91–102.

[57] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "The impact of api change- and fault-proneness on the user ratings of android apps," *Software Engineering, IEEE Transactions on*, vol. 41, no. 4, pp. 384–407, 2015.

[58] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: A threat to the success of android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 477–487.

[59] S. E. S. Taba, I. Keivanloo, Y. Zou, J. Ng, and T. Ng, *An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality*, 2014.

[60] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? A case study on free android applications," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, 2015, pp. 301–310.

[61] W. Luiz, F. Viegas, R. Alencar, F. Mourão, T. Salles, D. Carvalho, M. A. Gonçalves, and L. Rocha, "A feature-oriented sentiment rating for mobile app reviews," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18, 2018, pp. 1909–1918.

[62] H. Hu, S. Wang, C.-P. Bezemer, and A. E. Hassan, "Studying the consistency of star ratings and reviews of popular free hybrid android and ios apps," *Empirical Software Engineering*, 2018.

[63] G. Catolino, "Does source code quality reflect the ratings of apps?" in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. ACM, 2018, pp. 43–44.

[64] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "Ardoc: App reviews development oriented classifier," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, 2016.

[65] X. Gu and S. Kim, "What parts of your apps are loved by users?" in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, 2015, pp. 760–770.

[66] S. Scalabrino, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Listening to the crowd for the release planning of mobile apps," *IEEE Transactions on Software Engineering*, 2017.

[67] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, *Software and Systems Traceability*, 2012, ch. Information Retrieval Methods for Automated Traceability Recovery.

[68] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[69] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, 2010, pp. 375–384.

[70] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 12–23.

[71] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "DECA: development emails content analyzer," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pp. 641–644.

[72] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and dynamics of API discussions on stack overflow," Georgia Tech, Tech. Rep. GIT-CS-12-05, 2012.

[73] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *IEEE 20th International Conference on Program Comprehension (ICPC'12)*, 2012, pp. 63–72.

[74] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, "Codes: Mining source code descriptions from developers discussions," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014, 2014, pp. 106–109.

[75] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013.

[76] C. Duan, J. Cleland-Huang, and B. Mobasher, "A consensus based approach to constrained clustering of software requirements," in *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008, pp. 1073–1082.

[77] A. Ciurumelea, S. Panichella, and H. C. Gall, "Automated user reviews analyser," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 317–318.

Yu Zhou is a full professor in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics (NUAA). He received his BSc degree in 2004 and PhD degree in 2009, both in Computer Science from Nanjing University China. Before joining NUAA in 2011, he conducted PostDoc research on software engineering at Politechnico di Milano, Italy. From 2015-2016, he visited the SEAL lab at University of Zurich Switzerland, where he is also an adjunct researcher. His research interests mainly include software evolution analysis, mining software repositories, software architecture, and reliability analysis. He has been supported by several national research programs in China.

Yanqi Su received her BSc degree in Computer Science and Technology, from Nanjing University of Aeronautics and Astronautics (NUAA), China, in 2017. She is currently a MSc student in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. Her research interests include software evolution analysis, artificial intelligence, and natural language processing.

Taolue Chen received the Bachelor's and Master's degrees from the Nanjing University, China, both in computer science. He was a junior researcher at the CWI and acquired the PhD degree from the Free University Amsterdam, The Netherlands. He is currently a lecturer at the Department of Computer Science and Information Systems, Birkbeck, University of London. Prior to this post, he was a (senior) lecturer at Middlesex University London, a research assistant at the University of Oxford, and a postdoctoral researcher at the University of Twente, The Netherlands. His research interests include formal verification and synthesis, program analysis, logic in computer science, and software engineering in which areas he has published over 90 peer-reviewed papers.

Zhiqiu Huang is a full professor of Nanjing University of Aeronautics and Astronautics. He received his BSc. and MSc degrees in Computer Science from National University of Defense Technology of China. He received his Ph.D degree in Computer Science from Nanjing University of Aeronautics and Astronautics of China. His research interests include big data analysis, cloud computing, and web services.

Harald Gall is Dean of the Faculty of Business, Economics, and Informatics at the University of Zurich, Switzerland (UZH), and professor of software engineering in the Department of Informatics at UZH. His research interests are in evidence-based software engineering with focus on quality in software products and processes. This focuses on long-term software evolution, software architectures, software quality analysis, data mining of software repositories, cloud-based software development, and empirical software engineering. He is probably best known for his work on software evolution analysis and mining software archives. Since 1997 he has worked on devising ways in which mining these repositories can help to better understand software development, to devise predictions about quality attributes, and to exploit this knowledge in software analysis tools such as Evolizer or ChangeDistiller.

Sebastiano Panichella is a senior researcher at Zurich University of Applied Sciences (ZHAW), former at University of Zurich (UZH) when this work was carried out. His research interests are in software engineering (SE) and cloud computing (CC): Continuous Delivery, Continuous integration, Software maintenance and evolution (with particular focus on Cloud Applications), Code Review, Mobile Computing, Summarization Techniques for Code, Changes and Testing. His research interests also include Textual Analysis, Machine Learning and Genetic Algorithms applied to SE problems. His research is funded by one Swiss National Science Foundation Grants. He is author of several papers that appeared in International Conferences (ICSE, ASE, FSE, ICSME, etc.) and Journals (ESE, IST, etc.). These research works involved studies with industrial companies and open source projects and received best paper awards. He serves as program committee member of various international conferences (e.g., ICSE, ASE, ICPC, ICSME, SANER, MSR). He is a member of IEEE. He is Editorial Board Member of the Journal of Software: Evolution and Process (JSEP).