This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

Where2Change: Change Request Localization for App Reviews

Tao Zhang, Jiachi Chen, Xian Zhan, Xiapu Luo‡, David Lo, and He Jiang, *Member, IEEE*

Abstract—Million of mobile apps have been released to the market. Developers need to maintain these apps so that they can continue to benefit end users. Developers usually extract useful information from user reviews to maintain and evolve mobile apps. One of the important activities that developers need to do while reading user reviews is to locate the source code related to requested changes. Unfortunately, this manual work is costly and time consuming since: (1) an app can receive thousands of reviews, and (2) a mobile app can consist of hundreds of source code files. To address this challenge, Palomba et al. recently proposed `CHANGEADVISOR` that utilizes user reviews to locate source code to be changed. However, we find that it cannot identify real source code to be changed for part of reviews. In this work, we aim to advance Palomba et al.'s work by proposing a novel approach that can achieve higher accuracy in change localization. Our approach first extracts the informative sentences (*i.e.*, user feedback) from user reviews and identifies user feedback related to various problems and feature requests, and then cluster the corresponding user feedback into groups. Each group reports the similar users' needs. Next, these groups are mapped to issue reports by using $Word2Vec$. The resultant enriched text consisting of user feedback and their corresponding issue reports is used to identify source code classes that should be changed by using our novel *weight selection*-based cosine similarity metric. We have evaluated the new proposed change request localization approach (`Where2Change`) on 31,597 user reviews and 3,272 issue reports of 10 open source mobile apps. The experiments demonstrate that `Where2Change` can successfully locate more source code classes related to the change requests for more user feedback clusters than `CHANGEADVISOR` as demonstrated by higher Top-N and Recall values. The differences reach up to 17 for Top-1, 18.1 for Top-3, 17.9 for Top-5, and 50.08% for Recall. In addition, we also compare the performance of `Where2Change` and two previous Information Retrieval (IR)-based fault localization technologies:`BLUiR` and `BLIA`. The results showed that our approach performs better than them. As an important part of our work, we conduct an empirical study to investigate the value of using both user reviews and historical issue reports for change request localization; the results shown that historical issue reports can help to improve the performance of change localization.

Index Terms—User review, Issue report, Mobile app, Change Request localization, Software maintenance.

1 Introduction

As the number of mobile devices (*e.g.*, smartphones and tablet computers) and their applications (apps) increases, the task of maintaining mobile apps is becoming more important [1]. In online app stores such as Google Play Store, Apple Store, and Windows Phone App Store, users are allowed to evaluate each app by using scores (*i.e.*, five stars) and posting their reviews. These reviews are free-form text that may include important information such as bugs that need to be fixed for developers. These reviews express users' problems and suggestions, thus they can be used by developers to guide software maintenance activities for improving user experience. In the process of software maintenance, changing the source code to satisfy users' requrements is an important task [2]. In order to achieve the goal, developers' first priority is to locate source code that needs to be changed. However, it is difficult for developers to read each review in order to find the source files to be changed because popular apps usually receive hundreds of reviews every day. Undoubtedly, this is a time-consuming work.

Previous Information Retrieval (IR)-based fault localization technologies such as `BugScout` [3], `BugLocator` [4], `BLUiR` [5], and `BLIA` [6] tend to utilize issue reports to search the potential faulty source files or classes. However, these approaches focus on desktop software, which is different with mobile apps. According to the report at the literature [7], developers in apps first read user reviews, then resolve the issues reported in the user reviews and update the apps. To verify this process, we investigate top 200 most active developers in top 100 popular mobile apps, we find that 85.7% responders depend on user reviews to find and resolve issues from source code (See Section 6.2 for details). Therefore, when previous IR-based fault localization technologies are employed at mobile apps, they will be confronted with an important challenge: these technologies cannot automate the process of analysis on user reviews, thus they may ignore the problems reported by users because the publication time of user reviews is always ahead of the generation time of issue reports. This fact results in that developers still spend more time manually analyzing and understanding thousands of user reviews in an app. We start a follow-up investigation of the above-mentioned survey for the responders. In this investigation, we want to know the average time of change request localization for each new coming user review. 32 developers provide their answers, as the result, the average time per developer to find the classes that should be changed for a new user review is 1.9 working days. This fact shows manual change request localization is a time-consuming

‡*The corresponding author.*

- *Tao Zhang is with the Faculty of Information and Technology, Macau University of Science and Technology and the Department of Computing, Hong Kong Polytechnic University. E-mail: tazhang@must.edu.mo*
- *Jiachi Chen, Xian Zhan, and Xiapu Luo are with the Department of Computing, Hong Kong Polytechnic University. E-mail:{csjcchen,csxzhan,csxluo}@comp.polyu.edu.hk*
- *David Lo is with the School of Information Systems, Singapore Management University.*
- *He Jiang is with the School of Software, Dalian University of Technology.*

Manuscript received ; revised .

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

work due to a great number of user reviews posed every day. Therefore, for mobile apps, a new fully automated change request localization technology toward user reviews will be of great benefit to developers.

Recently, Palomba et al. proposed CHANGEADVISOR [8]. This approach clusters user reviews based on similar user requirements, and then locates the set of source code that needs to be changed for each cluster of reviews. CHANGEADVISOR measures the similarity between a cluster of reviews and source code. When the similarity value exceeds a threshold, the classes are returned. Unfortunately, many user reviews lack of detailed information. This can cause CHANGEADVISOR to miss the links between review clusters and the corresponding source code that should be changed. Indeed, our experiment finds that, using CHANGEADVISOR, a substantial proportion of reviews cannot be located to any class in source code (see Section 2 for details).

Let us consider a hypothetical scenario. As a developer of the popular mobile app K-9 Mail, *Mark*'s daily work is to fix faults and add features according to a great number of user reviews. When he finds a fault or a feature request-related review, he must locate the source code that should be changed. Obviously, this is a challenging and time-consuming task. *Mark* expects to use a useful tool which can locate all changes automatically. He first finds a tool CHANGEADVISOR. When using CHANGEADVISOR, *Mark* gets a ranked list of classes that link to some clusters of user feedback. But *Mark* finds that only a part of feedback clusters can be mapped to the corresponding classes. According to our investigation shown in Section 2.2 for top-10 popular mobile apps, there are 38 clusters of user feedback that cannot be linked to the classes by using CHANGEADVISOR. The miss rate reaches up to 33%. For a real case in K-9 Mail, there is a cluster-Topic-1 that describes the issue of notifications that cannot be turned off. CHANGEADVISOR cannot find any class which may link this issue. But in fact, the invited experts[1] verify that there are some classes such as GlobalSettings related to this issue. Therefore, for these clusters missed by CHANGEADVISOR, *Mark* needs to manually identify these classes. This takes much effort and thus an improved solution is desired. By analyzing CHANGEADVISOR, we find that this tool does not adopt the historical issue reports. This may be a major reason why CHANGEADVISOR cannot find the classes for some issues described in user reviews. According to our investigation shown in Section 6.2, 72.4% developers still need to depend on historical issue reports to locate source code classes related to change requests found in user reviews. Therefore, historical issue reports can help to change request localization due to the detailed descriptions for issues. In order to help apps' developers such as *Mark*, it is necessary to develop a new change request localization technology which adopts historical issue reports for reducing the miss rate.

To address the above-mentioned challenge, in this paper, we propose a novel approach named Where2Change to conduct change request localization for app reviews. First, we extract the informative sentences (*i.e.*, user feedback) contained in user reviews. Second, we use a popular review analysis tool-SURF [9] to automatically classify the user feedback into five categories: information giving, information seeking, *problem discovery*, *feature request*, and others. We focus on user feedback in the *problem discovery* and *feature request* categories. Next, after preprocessing the user feedback in above-mentioned two categories, we cluster them using HDP which presents the best clustering performance among six popular clustering algorithms. Third, we treat each cluster of user feedback as a query to search for the classes that should be changed in source code. Due to the small amount of information provided in user reviews, we introduce historical issue reports to enrich the user feedback extracted from user reviews. In particular, we build the multiple links between a cluster of user feedback and historical issue reports by computing their similarities via *Word2Vec* [10]. If the similarity value is larger than a threshold, an issue report can be used to enrich the cluster of user feedback. Finally, we propose a more accurate similarity metric named *weight-selection*-based cosine similarity to measure the similarity between an enriched version of comment cluster and the source code. This metric considers the influence of different terms' weights on the accuracy of change request localization so that the best weight values are used to measure the similarity. At the end, for each cluster of user feedback, Where2Change returns a ranked list of potentially classes to be changed. Overall, our approach can overcome the limitation of CHANGEADVISOR. For the case described in the last paragraph, by using our approach, the issue report # 1110 is used to enrich the cluster-Topic-1 due to its detailed description for the problem of notifications in K-9 Mail. Then Where2Change can easily find the class-GlobalSettings that should be changed for satisfying users' requirements. Therefore, it can reduce the workload of developers like *Mark* and consequently improve the efficiency of issue resolution.

We conduct experiments on 31,597 user reviews and 3,272 issue reports collected from 10 open source mobile apps on GitHub. The experimental results demonstrate that Where2Change can successfully locate more source code classes related to change requests for more user feedback clusters than CHANGEADVISOR due to the highest Top-N and Recall values. The differences reach up to 17 for Top-1, 18.1 for Top-3, 17.9 for Top-5, and 50.08% for Recall. We also conduct Wilcoxon test to further compare the performance between CHANGEADVISOR and Where2Change. The result indicates that our approach can significantly improve the performance of change request localization for user feedback by comparing with CHANGEADVISOR. Moreover, we conduct the performance comparison between our approach and two IR-based fault localization technologies-BLUiR and BLIA. The results show that our approach performs better than them. In order to explain why we use user feedback clusters as queries rather than issue reports, we conduct the empirical study for user reviews and issue reports, the results shown that issue reports can help to improve the performance of change request localization but cannot replace user reviews to conduct this task for mobile apps.

We summarize the contributions of our work as follows:

- Where2Change enriches user feedback extracted from user reviews via similar issue reports. This richer text enables Where2Change to perform better in change localization than CHANGEADVISOR and other fault localization approaches such as BLUiR and BLIA.
- We propose a new textual similarity metric (*i.e.*, weight selection-based cosine similarity) to produce an accurate result of change request localization.
- We implement Where2Change in a tool[2] to locate source code classes that should be changed according to user reviews on mobile apps. We evaluate the tool and compare

1. Please refer to Section 5.1: *Building Ground Truth*

2. It will be released after paper publication.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

it with CHANGEADVISOR. The result shows that our approach successfully locates more source code classes related to change requests for more user feedback clusters than CHANGEADVISOR and keeps the similar accuracy in top-5 ranking results.

Roadmap. Section 2 introduces the background knowledge related to our work and shows the motivation example in order to indicate why we need to propose a new change request localization approach based on user reviews for mobile apps. In Section 3, we detail the proposed change request localization technology. Section 4 describes the research questions that guide our experiment while Section 5 presents the experimental results and Section 6 discusses how the proposed approach performs better than the previous study. Section 7 introduces the threats to validity. In Section 8, we present the related work, and Section 9 concludes the paper and introduces our future work.

2 Background and Motivating Examples

2.1 Background

In this subsection, we introduce information retrieval (IR) based change request localization and a sample issue report for a mobile app.

2.1.1 Information retrieval (IR) based change request localization

Information retrieval (IR) based change request localization techniques attract wide attention due to their relatively low computation cost and external resources requirements (*i.e.*, only source code and software artifacts are required.) [11]. In these IR approaches, each software artifact (*e.g.*, issue report and user review) is treated as a query, and the source code (*e.g.*, source files, classes, and methods) to be searched as the document collection. Then, IR techniques rank the documents by computing the textual similarities between queries and documents. Finally, a list of ranked documents are returned.

Previous studies such as [3]–[6], [12]–[21] tend to utilize issue reports as queries to locate source code related to software faults in traditional desktop software (*e.g.*, Eclipse and Mozilla). For mobile apps, user reviews can also be used as queries to change request localization. The reason is described in Section 6.2.

Fig. 1 shows the examples of three user reviews in Wordpress. When the users find the problems of Notifications in Wordpress: "*the users cannot do anything (e.g., remove operation and read operation) when the notifications appear*", they posts some reviews in Google Play Store. With IR-based change request localization techniques, researchers treat these user reviews as queries to search the corresponding classes that should be changed in order to help developers resolve the reported issues. These classes related to three user reviews are presented as follows:

```
1  ui.notifications.NotificationsDetailActivity.java
2  ......
3  private NotificationDetailFragmentAdapter
       buildNoteListAdapterAndSetPosition(Note note,
       NotesAdapter.FILTERS filter)
4  {NotificationDetailFragmentAdapter adapter;
5  ArrayList<Note> notes = NotificationsTable.
       getLatestNotes();
6  ArrayList<Note> filteredNotes = new ArrayList<>();
7  NotesAdapter.buildFilteredNotesList(filteredNotes,
       notes, filter);
8  adapter = new NotificationDetailFragmentAdapter(
       getFragmentManager(), filteredNotes);
9  mViewPager.setAdapter(adapter); mViewPager.
       setCurrentItem(NotificationsUtils.
       findNoteInNoteArra y(filteredNotes, note.getId()
       ));
10 return adapter;}
```

Review Title: Notifications Blocking My View

Comment: Everytime I visit my blog through this app the **notification panel** just occupies the three-fourths of the screen and there is no way to remove it because everytime I tap it what is clicked is the blog underneath.

Review Title: Cannot be trusted

Comment: I used to love this app. Until things started changing for the worst. Doesn't refresh o sync, what I've done online sometimes doesn't show up here and vice versa. Cant even read **notifications**!!!!

Review Title: Latest update

Comment: I'm having problems with the latest update of this app. Once I open it by clicking on a **notification**, I cannot do anything else as it freezes. It's a shame, as the previous version has worked relatively well most of the times.

Fig. 1: User reviews in Wordpress

2.1.2 Issue reports

For each issue report in mobile apps, the main body is composed of title and description. Title briefs what is the issue while description details how the issue occurs. Fig. 2 shows an example of issue report (ID: #945) in WordPress. Its title is "*Crash on 2.6.1, IllegalStateException: Content view not yet created-NotificationsListFragment.requestMoreNotifications*" which indicates the crash problem when invoking NotificationsListFragment.requestMoreNotifications. The developer named "maxme" opened this issue report and posted the description. The description includes the information of stack traces in order to show the root reason how the issue occurs.

We find that the above-mentioned issue report describes a similar (yet different) issue to the reviews shown in Fig. 1. Therefore, the issue report #945 (which contains more detailed information) can help to locate source code to be changed corresponding to the user reviews. We discuss it in the next subsection.

2.2 Motivating example

In this subsection, we discuss a real-world example that motivates our research. We collected reviews, issue reports, and their corresponding fixed files from the open-source mobile apps. Since CHANGEADVISOR is, to the best of our knowledge, the state-of-the-art work on mapping user reviews to code, we run it and use the same data set to compare it and our approach in Section

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

Fig. 2: An example issue report from Wordpress

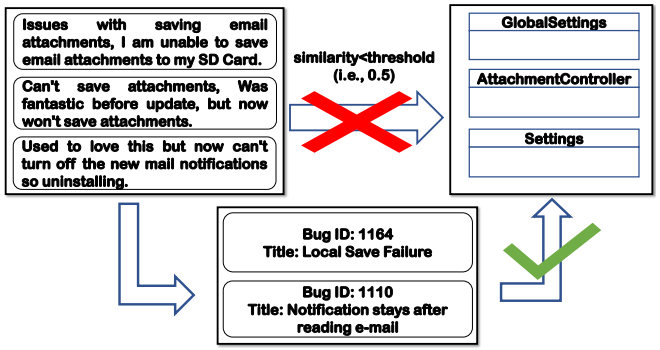

Fig. 3: Motivating example in K-9 Mail: change request localization

5. In detail, we downloaded 31,597 user reviews from Google Play Store, 3,272 issue reports and 4,207 classes from GitHub. Then we use HDP [22] to cluster user feedback extracted from user reviews and compute the similarities between each cluster and classes for reproducing CHANGEADVISOR. As a result, we observe the following issue shown in Fig. 3.

Consider a developer *Mark* who is using CHANGEADVISOR. Even though *Mark* can verify the bug and feature request-related reviews correctly using CHANGEADVISOR, he will encounter a challenge. Fig. 3 shows an example of change request localization for Topic-1 of the app K-9 Mail. We suppose that *Mark* has selected the reviews that describe the real issues from all reviews in Topic-1. We note that the similarity value is less than a threshold (*i.e.*, 0.5[3]) for identifying potential classes that should be changed. Thus, none of the classes can be matched to Topic-1 via CHANGEADVISOR. However, *Mark* finds that the reviews in Topic-1 can be mapped to the classes by reading the related issue reports. He still needs to manually locate the classes to be changed for these reviews. For example, the review "*Used to love this but now can't turn off the new mail notifications so uninstalling*" is related to issue report # 1110. Both of them present the problem of mail notifications that cannot be turn off. According to the change history in GitHub, we find that the issue reported in # 1110 is resolved by changing the source code file corresponding to the class-GlobalSettings. Thus, the user feedback in Topic-1 could have been matched to the class, but CHANGEADVISOR cannot find it. According to our analysis for the results produced by CHANGEADVISOR, we find the number of clusters that are not linked to classes, which is shown in Table 1. The data on last column indicates the total number of issue reports (*i.e.*, unlinked and linked reports) in each app.

3. The threshold value does not appear in the paper, it is found when we run the replication package provided by Palomba et al.

TABLE 1: The status of number of feedback clusters linked to classes when using CHANGEADVISOR

Project	# feedback clusters linked to classes		
	unlinked	linked	total
AntennnaPod	4	8	12
Automattic	6	4	10
Cgeo	8	4	12
Chrislacy	3	5	8
K-9 Mail	4	10	14
OneBusAway	3	9	12
Twidere	3	9	12
UweTrottmann	3	11	14
WhisperSystems	2	91	11
Wordpress	2	8	10
Total	38	159	115

In Table 1, 33% (38/115=33%) of clusters are missed by using CHANGEADVISOR. Relatively high miss rate results in that developers like *Mark* need to spend additional time to find the classes that should be changed for the user feedback clusters missed by CHANGEADVISOR.

Based on the above observation and analysis, we have the following motivation for this study:

Motivation: The purpose of CHANGEADVISOR is to return a list of ranked classes to be changed for each cluster of user feedback. However, not all feedback clusters can be mapped to classes by CHANGEADVISOR. Developers still need to locate the classes to be changed. This observation motivates us to propose a new and effective approach to accurately identify the classes that should be changed for satisfying more user requests described in user reviews.

In order to achieve the above-mentioned goal, we conduct the investigation on real developers for the top-100 popular apps, and find that 72.4% developers still rely on historical issue reports to locate source code classes related to change requests found in user reviews. Therefore, historical issue reports can facilitate change request localization due to the detailed issues' descriptions. To help apps' developers such as *Mark*, it is necessary to develop a new change request localization technology which adopts historical issue reports for reducing the miss rate.

Due to the aforementioned motivation, we propose a two-phase approach to locate classes to be changed in mobile apps according to user reviews. Section 3 describes the design of this approach.

3 METHODOLOGY

In this section, we first show the overall framework of Where2Change. Next, we detail how to implement change request localization based on user reviews on mobile apps in Where2Change.

3.1 Overall framework

To locate source code classes related to change requests appearing in user reviews on mobile apps, we propose Where2Change, a two-phase method to retrieve the classes to be changed for resolving the bugs and feature requests described in user reviews. In the first phase, Where2Change extracts the informative sentences from user reviews as user feedback. It selects change requests-related user feedback and enriches them using issue reports; in

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

the second phase, `Where2Change` recommends a list of ranked classes that should be changed for each cluster of user feedback. Fig. 4 shows its overall framework. In this framework, the first phase includes the steps (1-3) while the step (4) belongs to the second phase.

In this framework, we first **(1)** utilize SURF [9], a state-of-the-art review analysis tool, to extract the informative sentences belonging to categories *feature request* and *problem discovery*. Then we **(2)** cluster pre-processed user feedback via Hierarchical Dirichlet Processes (HDP) which performs the best among six clustering algorithms. Next, we **(3)** compute the similarity between a cluster of user feedback and a pre-processed issue report via Word2Vec. If the similarity value exceeds the threshold, we treat the issue report as a change-related issue report. In other words, we build a link between a cluster of user feedback and the related issue report. We use these related issue reports to enrich the cluster of user feedback. These enriched versions of feedback clusters are treated as queries. After we pre-process the source code, we **(4)** compute the similarity between each query and the potential classes to be changed via the proposed weight selection-based cosine similarity. Finally, we get a ranked list of classes for the cluster of user feedback.

We give an example to further explain the process of our change request localization approach. For a user review "*Used to love this but not can't turn off the new mail notifications so uninstalling*" shown in Fig. 3, by using SURF, we learn that this review includes only one informative sentence which belongs to the category *Problem Discovery*. Then we extract it as user feedback. After pre-processing, we use HDP to group the user feedback and other similar feedback entries into *Topic 1*. Next, by using Word2Vec, we find that there are some issue reports such as #1662 and #1110 linked to them because the similarity score is more than a threshold value (*i.e.*, 0.4). We use these issue reports to enrich the cluster. Finally, we compute the similarity scores between the different versions of enriched feedback clusters and the classes that should be changed. As a result, we get a global ranked list of all potential classes with the similarity scores.

In the following subsections, we show how to implement the novel change request localization approach step by step.

3.2 Selecting change request-related user feedback

User reviews provide rich information that can facilitate the development and maintenance of mobile apps, however, they often include a lot of uninformative data that should be eliminated. To resolve the problem described in Section 1, we first choose change request-related user feedback extracted from user reviews that describe real faults or feature requests.

We utilize SURF [9] to remove the uninformative data in user reviews and find change-related user feedback. For ARDOC [23] used in `CHANGEADVISOR` [8], Sorbo et al. pointed out that this one-dimensional classification approach cannot sufficiently utilize the available review information [9]. Due to this reason, they developed SURF which can facilitate developers to understand the contents of user reviews. SURF relies on AR-miner [24] to filter out non-informative reviews in our dataset, then it employs an Intent Classifier [23] combining Natural Language Processing (NLP), Sentiment Analysis (SA) and Text Analysis (TA) techniques through a Machine Learning (ML) algorithm for detecting sentences from the five categories: *Feature Request*, *Problem Discovery*, *Information Seeking*, *Information Giving* and *Other*. Finally, SURF uses a sentence selection and scoring mechanism to generate the summaries.

Because we focus on change-related user feedback, we only collect the sentences in the categories *Problem Discovery* and *Feature Request* to generate the queries. The summaries produced by SURF are used to help the invited developers create the ground truth (See Section 5.1).

In order to ensure the classification is acceptable, we verify whether all user feedback classified into the categories *Problem Discovery* and *Feature Request* really belongs to them. The second author and the third author are responsible for verifying whether the user feedback describes a real fault or a feature request. They have more than 5-years experience in software testing. They are also familiar with the apps' user reviews and issue reports. The user feedback is divided into two groups. One person is invited to check one group and another person is responsible for checking the remainder. In order to reduce the possible bias, they exchange their data to conduct the verification again. We invite the software test expert from Alibaba Company to make a final decision when the verification results are inconsistent. He has more than 15-years software testing experience at Baidu and Alibaba. As a result, we get an accuracy of 95.64%. Therefore, the classification results is acceptable.

3.3 Clustering change-related user feedback

Before clustering change request-related user feedback the pre-processing for NLP is executed. Since user feedback is written by end-users, the language is generally informal and very noisy. The feedback is different from conventional software artifacts like issue reports. Thus, they should be further processed in order to better match issue reports by using our approach. For this reason, we adopt the text processing tools that contain python libraries NLTK[4] and TEXTBLOB[5], and the spell check tool-PYENCHANT library to implement the following steps:

- **Tokenization:** An issue report or a piece of user feedback is split into a list of words (*i.e.*, tokens), which can be used to compute the textual similarity.
- **Stop word removal:** Stop words like "the", "a", and "are" are common words but they make no sense to change localization. Therefore, these words are removed according to the list of WordNet English stop words. We maintain this list at https://github.com/ReviewBugLocalization/ReviewBugLocalization. Note that the predicate negatives such as "aren't", "isn't", "can't" are also appeared in this list, which are removed.
- **Stemming:** The words are transformed to their basic forms (*i.e.*, stems). For example, "running" is changed to "run", and "bugs" is changed to "bug".
- **Lemmatization:** It is the process of grouping together the inflected forms of a word so that they can be analyzed as a single item, identified by the word's lemma. Lemmatization is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. For example, a word "better" can be transformed to "good" when we use lemmatization. But for stemming, it cannot conduct the transformation.

4. http://www.nltk.org
5. http://textblob.readthedocs.org/en/dev/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

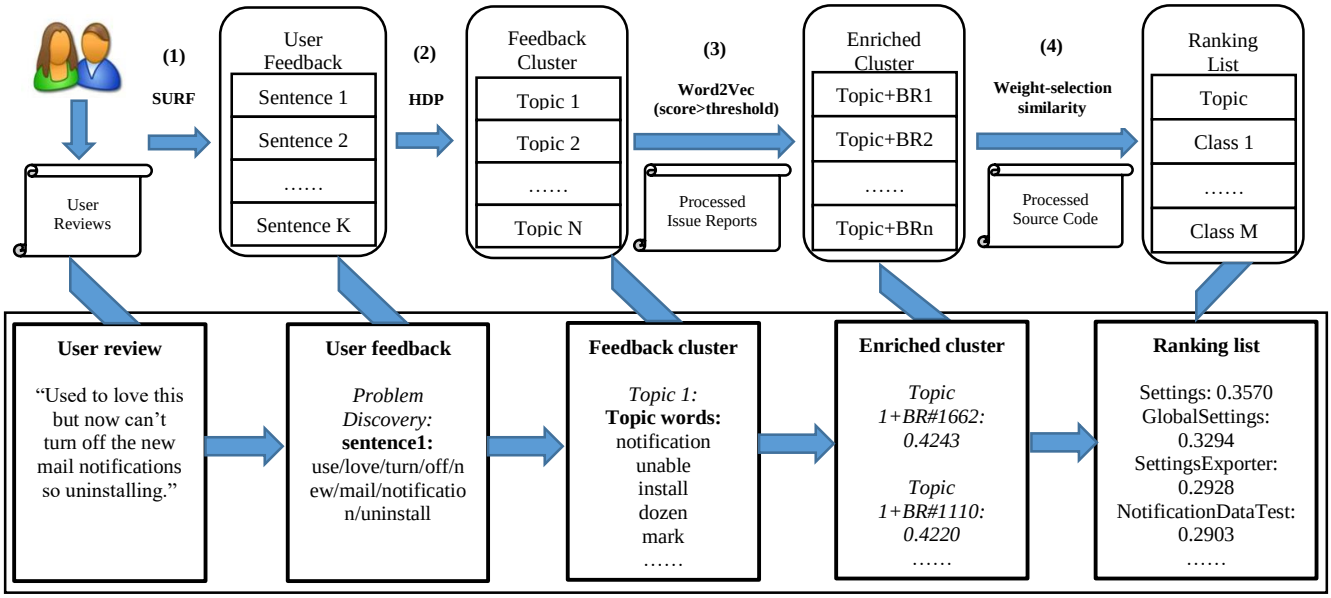

Fig. 4: Overall framework of `Where2Change` for user reviews in mobile apps

- **Spelling correction:** By using the spell check tool-PYENCHANT library, the misspelled words appearing in user feedback are corrected.
- **Contractions expansion:** We extend the possible contractions of English words in user feedback. For example, the abbreviated form "It's" becomes "It is".
- **Nouns and verbs filtering:** We adopt a part of speech (POS) tagging classification to identify the nouns and verbs from user feedback and issue reports. Only these words are considered to compute the following textual similarity because they are the most representative words in the documents.
- **Non-English characters filtering:** We find that the ASCII codes of the non-English characters are out of the range of 65-90 and 97-122. Therefore, we utilize regular expression to filter the non-English characters existing in each word by matching the range of the ASCII codes.

As a result, the process returns bag-of-words for user feedback. We use them as the input of the following steps.

We find that there are three clustering algorithms frequently utilized in the research articles[6] published in Software Engineering (SE) field at the last 5 years. These algorithms include Latent Dirichlet Allocation (LDA), K-means, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). Moreover, we also consider to verify the performance of the three extended versions of LDA: SentenceLDA, CopulaLDA, and HDP. By comparing their performance on our data set, the result shows that HDP performs the best among the above-mentioned clustering algorithms. Therefore, we select HDP to cluster the user feedback. With regard to comparing process and result, please refer to Section 5.

When the process of topic modelling is finished, each cluster is treated as a query to search the classes that need to be changed.

6. We only consider the research papers published in ICSE, FSE, ASE, TSE, TOSEM, and EMSE

3.4 Building a link between feedback clusters and issue reports

Most of users have not enough knowledge to understand program development and bug fixing, thus user feedback usually includes inadequate information related to change requests. Therefore, directly computing the textual similarity between the user feedback and classes can lead to low similarity scores so that it is possible to result in the result like `CHANGEADVISOR` that some change request-related reviews cannot match the correct classes because the similarity score is lower than the threshold. To avoid this situation, we use issue reports that provide the detailed information of bugs and feature requests as a bridge to build a link between feedback clusters and classes to be changed so that we can use issue reports to enrich user feedback.

Before enriching feedback clusters via issue reports, it is necessary to refine these clusters because the clustering results via algorithms are still far away from the manual classification results (See Section 5). Note that we do not use the manual classification results to replace the automated clustering results due to the following two reasons. First, `Where2Change` is a semi-automated change request localization tool which is developed to reduce the developers' workload. Thus, we only conduct the moderate manual refinement process for feedback clusters; second, our purpose is to build the accurate links between feedback clusters and issue reports rather than directly compute the similarity scores between feedback clusters and source code. By referring to the manual classification results, we conduct the refinement process as the following two steps:

- We remove the feedback entries which are very different from others in the same cluster.
- We add the appropriate feedback entries which have the stronger relations with the most of feedback entries in a cluster.

We adopt a method named "*full voting*" to complete the aforementioned steps. In detail, we convoke the persons who have the qualification to decide which feedback entries should be removed/added. We grant seven persons the right to vote. These

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

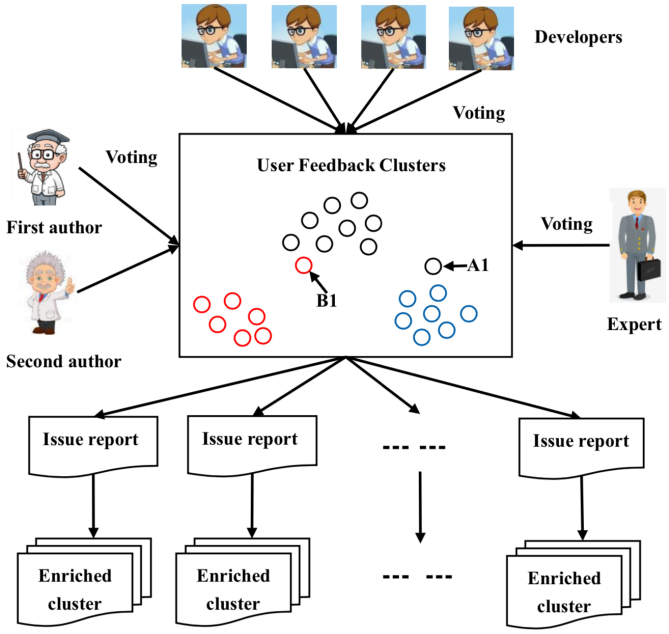

Fig. 5: Refinement process for the clustering results

persons include the first two authors of our article, the developers who are invited to manually classify user feedback, and the expert who are responsible for verifying the classification results. When all seven persons come to an agreement, we conduct the refinement process by removing/adding the corresponding feedback entries. Fig. 5 shows the refinement process for the clustering results via HDP. The small circles present the user feedback entries which are clustered into the three difference groups with the difference colors. Note that A1 is far from the original cluster marked by the black color and close to another cluster marked by the blue color. We have the same finding for B1. When all seven persons also vote for removing them from their original clusters and adding them into other clusters, the refinement process is conducted. For example, the user feedback entry "*In other words, you cant reverse sort or manually configure it either*" describes the different problem with other feedback entries such as "*It doesn't show up on my phone except to say it's been successfully installed*" in the same cluster. The former indicates a sorting problem of AntennaPod while the latter entries describe the problems of AntennaPod on installation and downloading. All seven persons also agree to remove the former entry and add it into a new cluster. In the new cluster, this entry describes the similar problem with other ones such as "*No sorting You aren't able to sort podcast subscriptions or podcast episodes*". These entries also describe the sorting problem of AntennaPod.

After conducting the refinement process, we build the links between the user feedback clusters and the appropriate issue reports to produce the enriched versions of feedback clusters in order to improve the accuracy of change request localization.

We compute the textual similarity between a cluster of user feedback and an issue report by utilizing Word2Vec [10] which can help to locate more source code classes related to change requests for more user feedback than other similarity metrics such as Dice coefficient [25], tf·idf [26], and Microsoft Concept Graph (MCG) [27]. With regard to comparing process and result, please refer to Section 5. Word2Vec aims to map a word into semantic word embedding. It takes a large corpus of text as its input and produces a vector space, with each unique word in the corpus being assigned a corresponding vector in the space. We utilize Word2Vec with the skip-gram model [28]. In k dimensions (k=100 in our work), each word can be represented as the vector defined as follows:

$$\overrightarrow{vec(word)} = \langle v_1, v_2, \ldots, v_k \rangle \tag{1}$$

Thus, a document can then be mapped into the space by:

$$C_s = \theta^T \cdot H^W \tag{2}$$

Here, θ^T is the vector of the TF-IDF weights of the words in the document computed by the following formula:

$$TF - IDF\, weight = tf_{t,d} \times log\frac{N}{n_t}, \tag{3}$$

where $tf_{t,d}$ is a frequency of term t in the document d. $log\frac{N}{n_t}$ presents the inverse document frequency which is a measure of how much information the word provides. N is the total number of documents while n_t is the number of documents which contain term t.

H^W is the word vector matrix. In this matrix, the i-th line represents the word vector of the word i. The matrix is constructed by concatenating the word vectors of all words in the document. Via matrix multiplication, a document is transferred to a vector of semantic categories, denoted as C_s.

When we get the word vectors of the cluster of feedback $cluster_i$ and the issue report IR_j, we use the cosine similarity to compute their semantic similarity, which is defined by:

$$CosineSim(cluster_i, IR_j) = \frac{\sum_{k=1}^{n} \omega_{ki}\omega_{kj}}{\sqrt{\sum_{k=1}^{n} \omega_{ki}^2} \times \sqrt{\sum_{k=1}^{n} \omega_{kj}^2}}, \tag{4}$$

where ω_{ki} and ω_{kj} indicate the weight of k^{th} word in $cluster_i$ and IR_j, respectively. They are computed by formula (2).

When the similarity score is more than the threshold, we treat the issue report as the feedback-related issue report. Thus, the link between the cluster of user feedback and the issue report is built. Note that the feedback cluster may link to multiple issue reports.

After we get the links between the feedback cluster and the issue reports, we use the issue reports to enrich the feedback cluster. As a result, we can get the multiple enriched versions of this cluster of feedback. For example, if we use the issue report IR_1 to enrich the feedback cluster $cluster_i$, we can get the enriched feedback cluster Ec_{i1} which is one of the multiple enriched versions.

3.5 Change request localization using enriched cluster of user feedback

Before conducting change request localization through the enriched feedback clusters, we start a process of pre-processing to remove noise data contained in source code. We implement the same steps of pre-processing with user feedback and issue reports. More precisely, we 1) separate composed identifiers using the camel case splitting which separates words on underscores, capital letters, and numerical digits base, 2) return capital letters to lower case, and 3) remove special characters.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

Classical cosine similarity does not consider the influence of different words' weights on the performance when we use it to implement change request localization. Undoubtedly, some important words that may have low weights[7] so that the accuracy of change request localization is reduced.

In order to resolve this problem, we propose a weight selection-based cosine similarity to select the best weight value for each word in enriched cluster of user feedback and classes to be changed so that we can obtain the best performance of change localization. We present the new similarity metric in Algorithm 1.

Algorithm 1 Weight selection-based cosine similarity

Input: Ec: A set of enriched clusters of user feedback; C: A set of potential classes that should be changed; Ir: A set of issue reports; $W_{init_i^j}$: initial weight of word j in C_i; $stepsize$: used to adjust the weight of words; K: the number of classes we will recommend.

Output: A ranked list of top-K classes to be changed.

$W_{last_i^j}$=$W_{init_i^j}$; //Initialization
While iteration times $iter < 100$ Do:
  $W_{current_i^j}$=$W_{last_i^j}$;
  For each issue report Ir_i in Ir:
    CC_i=correct classes to be changed (Ground Truth List);
    Compute cosine similarity scores between Ir_i and all classes using $W_{current_i^j}$;
    For C_i is ranked at top-K in the output list:
      $W_{current_i}$ is the weight of each word in C_i;
      If C_i is in CC_i: Do nothing;
      Else the weight of all common words=$W_{current_i}$-$stepsize$;
    End For
    For C_i is ranked at from K+1 to the maximal number of classes:
      $W_{current_i}$ is the weight of each word in C_i.
      If C_i is in CC_i:
        the weight of all common words=$W_{current_i}$+$stepsize$;
      Else Do nothing;
    End For
  End For;
  $W_{last_i^j}$=$W_{current_i^j}$;
  If the $MRR_{current} > MRR_{previous}$:
    $W_{best_i^j}$=$W_{current_i^j}$;
End While and get $W_{best_i^j}$
Compute cosine similarity between Ec_i and all classes using $W_{best_i^j}$.
return A ranked list of top-K classes to be changed;

Since each enriched cluster of user feedback is generated by linking them with issue reports, we first compute the textual similarity scores between issue reports linked to user feedback and source code classes to decide the best weights of words when the number of iteration times achieves 100[8]. At each iteration, when a top-K class C_i in the output list is not a correct class to be changed, we reduce the weight of all common words by subtracting a $stepsize$ (*i.e.*, 0.05). In the other case, if C_i ranked at behind top-K (*i.e.*, ranked at from top K+1 to the maximal number) is a correct class, we add the weight of all common words by adding a $stepsize$. This process terminates till the MRR score achieves the highest. MRR is a frequently-used evaluation function in Information Retrieval-based Change Request (or Fault) Localization. We show the detailed definition and explanation at Section 5.1.

We use the best weight of each word to compute cosine similarity between each feedback cluster and source code classes in order to achieve the optimum performance of change request localization so that it can ensure that more feedback clusters can be linked to the correct classes that should be changed.

For each enriched version[9] (*i.e.*, Ec_{11},...,Ec_{1i}) of the feedback cluster-$cluster_1$, we can get the ranked list of classes to be changed. For all enriched versions of the feedback cluster, we get the final result as follows:

$$CL(cluster_1) = CL(Ec_{11}) \cap CL(Ec_{12}) \cap \ ... \ \cap CL(Ec_{1i}), \tag{5}$$

where $CL(cluster_1)$ represents the final list of classes to be changed for the cluster of user feedback $cluster_1$. $CL(Ec_{1i})$ stands for the list of ranked classes to be changed for an enriched version Ec_{1i} of the feedback cluster. Note that Ec_{1i} is enriched by the issue report IR_i.

To fairly compare the performance of our approach and `CHANGEADVISOR`, for each class in the final ranking list of a feedback cluster (*e.g.*, $cluster_1$), we choose the highest similarity score between it and all enriched versions as the final ranking score.

4 Research Questions

We evaluate the proposed change localization approach from four aspects. First, we examine whether we have selected the best approach for each step (*i.e.*, user feedback clustering, feedback cluster enrichment, and class ranking) in our approach using our data set (*i.e.*, RQ1). Second, we evaluate whether the proposed approach performs better than the previous studies including `CHANGEADVISOR`, `BLUiR`, and `BLIA` (*i.e.*, RQ2). Next, we analyze the importance of issue reports in our approach (*i.e.*, RQ3). Finally, we explain why we select user feedback as queries rather than issue reports to conduct change request localization (*i.e.*, RQ4).

We answer RQ1 and RQ2 in Section 5, and answer RQ3 and RQ4 in Section 6.

- **RQ1:** ***For each step in our approach, do we select the best method among alternative ones?***
  Motivation: 1) In the process of user feedback clustering, we implement six clustering algorithms *(i.e.*, LDA, sentenceLDA, CopulaLDA, HDP, K-means, and DBSCAN) to group them. Thus, we need to find which one performs the best and use it to generate the clusters as the queries to search the classes that need to be changed; 2) In the process of user feedback enrichment, we introduce three similarity metrics including Dice coefficient, tf·idf, and MCG to build a link between a cluster of feedback and the issue reports by

7. Due to the fewer occurrence, some important words may have lower weight than other unimportant words.

8. We find that the results are not almost changed when the number of iteration times is more than 100, thus we set the number of iteration times is 100.

9. Each enriched version is produced by a cluster of user feedback and each issue report which links to the feedback cluster.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

replacing Word2Vec. Therefore, we need to evaluate whether Word2Vec performs the best when we use it to enrich user feedback clusters; 3) As a key part of Where2Change, we propose the new similarity function named by weight selection-based cosine similarity to measure the similarity between an enriched user feedback cluster and the classes to be changed. This novel similarity function considers the influence of different weights of terms on the performance of change localization so that the best weight values are used to measure the similarity. It is necessary to verify whether the newly proposed similarity metric performs better than classic cosine similarity.

Method: For Motivation 1), we use the evaluation metrics $Homogeneity$, $Completeness$, and V score [29] to compare the results of clustering algorithms with *gold standard* which consists of the clusters generated by the experienced developers manually. Then we select the best one to cluster the user feedback. For Motivation 2) and 3), we invite experienced developers to manually build the ground truth which reports the actual links between user feedback clusters and the source code, then we can use these classes contained in the ground truth to compare the $Top-N$, $Precision$, $Recall$, MRR, MAP values for each project in our data sets. According to the results of change localization, we can know which metric is more suitable to find correct issue reports for enriching user feedback clusters and whether the weight selection-based cosine similarity performs better than classic cosine similarity.

- **RQ2:** ***Does our approach outperforms others in terms of the accuracy of change localization?***
  Motivation: 1) By using issue reports to enrich user feedback contained in the clusters and utilizing the proposed weight selection-based cosine similarity function, we successfully implement Where2Change to locate classes to be changed according to user reviews. We should evaluate whether these new characteristics in Where2Change can lead to higher accuracy of change localization than CHANGEADVISOR. 2) In the literature [8], Palomba et al. compare the performance of CHANGEADVISOR and BLUiR which is a structured IR-based fault localization approach. Youm et al. proposed BLIA which performs better than BLUiR. Therefore, it is necessary to evaluate whether our approach performs better than BLUiR and BLIA.
  Method: We can answer this research question by comparing the $Top-N$, $Precision$, $Recall$, MRR, MAP values between our approach and the previous studies that include CHANGEADVISOR, BLUiR, and BLIA for each project in our data sets.
- **RQ3:** ***Does the incorporation of issue reports help to boost the effectiveness of Where2Change?***
  Motivation: By using Where2Change, we can locate more source code classes that should be changed for more user feedback clusters. Researchers may be interested in the root causes. Since we adopt issue reports to enrich user feedback contained in the clusters, it is necessary to analyze how issue reports can help to improve the performance of the proposed approach.
  Method: We analyze the compositions of issue reports in order to show how these compositions can help to improve the performance of change request localization. In order to demonstrate the importance of issue reports in our approach, we compare the performance of change request localization using our approach that utilizes the enriched user feedback clusters via issue reports and the performance of the approach that only adopts original user feedback.
- **RQ4:** ***Why do we adopt user feedback rather than issue reports as queries to conduct change localization?***
  Motivation: In the previous studies, researchers tend to utilize issue reports as queries to perform change request localization. Even though CHANGEADVISOR also used user feedback to implement the same goal, it does not give an answer for this question. Thus, in our work, we should investigate the reason by deeply analyzing the different characteristics of user feedback and issue reports for mobile apps.
  Method: We analyze the different characteristics of user feedback and issue reports, and investigate their generation frequency in each mobile app. According to the result, we can explain why we adopt user feedback as queries for locating classes related to change requests.

TABLE 2: The scale of our data set

Project	# RE	# RP	# CS	# FB	Period
AntennaPod	2,089	114	350	125	21/09/12-21/12/16
Automattic	1,404	95	66	225	18/06/13-29/11/16
Cgeo	4,480	1,488	790	414	18/07/11-05/02/17
Chrislacy	1,477	153	152	200	16/02/13-27/06/14
K-9 Mail	4,480	58	529	1,109	18/03/15-05/02/17
OneBusAway	2,107	271	293	306	14/08/12-25/01/17
Twidere	2,120	117	610	486	07/07/14-05/02/17
UweTrottmann	4,480	114	335	369	03/07/11-26/01/17
WhisperSystems	4,480	209	702	346	22/12/11-06/02/17
Wordpress	4,480	653	612	1,339	07/03/13-08/02/17
All	31,597	3,272	4,439	4,919	

5 EXPERIMENT

5.1 Experiment setup

We collect the user views, the issue reports, and the classes from 10 open-source mobile app projects in GitHub. Note that we only consider the closed (*i.e.*, fixed) issue reports because their descriptions are confirmed and effective. We first download top-100 popular open source mobile apps according to the stars' ranking list in GitHub as our candidate projects, and then we filter out the projects which have less than 50 issue reports and 1,400 reviews because a small number of issue reports and reviews are not sufficient to evaluate the performance of our approach and other methods. Finally, we select top-10 projects from the remaining mobile apps. The scale of our data set is shown in Table 2. In the first row, **RE**, **RP**, **CS**, and **FB** stand for user reviews, issue reports, classes, and user feedback, respectively. For **Period**, the format is *day/month/year* where "year" indicates the time line in the CENTURY 21.

In our data sets, we have two projects (*i.e.*, K-9 Mail and WordPress) that have been used to evaluate CHANGEADVISOR. We do not collect the data from other projects adopted by CHANGEADVISOR because they do not meet our selection criteria (*e.g.*, small number of issue reports). The experimental result shows that our approach outperforms CHANGEADVISOR in both projects that have been used in the evaluation of CHANGEADVISOR and new projects.

In order to fairly compare our approach-Where2Change and the baselines that include CHANGEADVISOR [8], BLUiR [5], and BLIA [6], we adopt the same user feedback in the categories *Problem Discovery* and *Feature Request* to implement them.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

After we extracted user feedback from user reviews, we cluster the user feedback as queries to implement the baselines. Specially for CHANGEADVISOR, we do not adopt the default value defined in the online appendix [30]. Instead, we adjust the threshold value from 0.1 to 1 in order to compare the performance of our approach and the best performance of CHANGEADVISOR on our data set.

To compare Where2Change and baselines, we should build a benchmark data set (*i.e.*, ground truth). Since Palomba et al. do not open the ground truth at any forum[10], it is difficult to compare the results. Therefore, it is necessary to build a high-quality ground truth which can be used to evaluate the performance of our approach and baselines. Fortunately, we find the following way to create the ground truth:

Building Ground Truth: We also adopt the manual verification method which is the same as the approach used in CHANGEADVISOR. In detail, we invite four developers from Zhuhai Duozhi Science and Technology Company Limited to help us build the links between user feedback clusters and source code classes. These developers have more than 10-years software programming and testing experience. In addition, they are familiar with the mobile apps' software maintenance and testing process. We pay 100 RMB (equal to 14.5 USD) to each developer per working day. They rely on the summaries produced by SURF to understand the user feedback and build the links. In addition, to avoid the possible omission for the links, they also refer to the contents of issue reports and commits which can help them understand which issues were mentioned and which source code classes were changed by developers in the historical software maintenance process. After one month (22 working days), they had finished the task. Then we invite the senior software test specialist from Alibaba Company to verify whether these source code classes are linked to the given user feedback clusters accurately. He has more than 15-years software testing experience at Baidu and Alibaba. He has the right to modify the errors by discussing with the above-mentioned four developers. Finally, we get the ground truth for these projects. Table 3 shows the scale of the ground truth. In this table, CS_{link} shows the number of classes linked to the user feedback clusters while $CS_{overall}$ represents the total number of classes in each app of our data set. CL_{HDP} indicates the number of user feedback clusters in each app. $AvgCS_{link}$ is defined by $\frac{CS_{link}}{CL_{HDP}}$, which stands for the average number of linked classes per cluster. $R_{AvgCS_{link}}$ shows the ratio of the average number of linked classes per cluster to the overall number of classes in each app, which is defined by $\frac{AvgCS_{link}}{CS_{overall}}$. We note that the ratio is less than 5% for most of apps except Chrislacy and WordPress in which the ratio is less than 8%. This fact indicates that manual change request localization is a difficult work because developers should select the small number of ones linked to user feedback from plenty of classes. Obviously, it is a time-consuming task, fortunately, our approach can automate this process so that it becomes easy for developers.

In order to guarantee the quality of the ground truth, we invite the top-10 active developers who posted the greatest number of comments in each app of our data set to verify the correct links between user feedback clusters and source code classes. We define two metrics that include hitting rate and missing rate to evaluate the quality. The hitting rate indicates how many classes are correctly linked to user feedback clusters in the ground truth, which is defined by the ratio of the number of correct links to the overall number of links in the ground truth; and the missing rate presents how many links between classes and user feedback clusters are missed, which is defined by the ratio of the number of correct links missed by the developers when they build the ground truth to the overall number of classes that should be correctly linked to the user feedback clusters. As a result, only one developer from Automattic gives a positive response and are willing to help us verify whether all 34 classes are correctly linked to 12 user feedback clusters produced by HDP in the ground truth. Moreover, he also help to check whether the ground truth may miss some links in the remaining 32 (66-34=32) classes which are not linked to any clusters by the developers. In the letter of reply, he wrote "*I have very interested in your current work so that I am willing to help you check these links in Automattic. I want to try Where2Change as soon as possible.*" In the end, he found that 33 classes are correct linked to the user feedback clusters (*i.e.*, the hitting rate=33/34=97.06%). In the remaining 32 classes, he found that there are 2 classes that should be linked to the clusters (*i.e.*, the missing rate=2/(33+2)=5.71%). Overall, the result is acceptable and the low missing rate only has the slight influence for the results produced by our approach. Therefore, we confirm the developers' capacity for building the ground truth. When we invite the developer from Automattic to continue checking the links in other apps, he told us he is very busy and he is not familiar with other apps. Therefore, it may be a threat that we are not sure what the hitting rate and the missing rate are in other apps. However, because of the case of Automattic which has the similar characteristics with other mobile apps [1], we believe that the threat is not big.

TABLE 3: Data scale of ground truth

Project	CS_{link}	$CS_{overall}$	CL_{HDP}	$AvgCS_{link}$	$R_{AvgCS_{link}}$
AntennaPod	140	350	36	3.9	1.1%
Automattic	34	66	12	2.8	4.2%
Cgeo	726	790	20	36.3	4.6%
Chrislacy	88	152	8	11.0	7.2%
K-9 Mail	71	529	8	8.9	1.7%
OneBusAway	235	293	34	6.9	2.4%
Twidere	264	610	30	8.8	1.4%
UweTrottmann	120	335	20	6.0	1.8%
WhisperSystems	185	702	16	11.6	1.7%
WordPress	577	612	12	48.1	7.9%

Pre-training Word2Vec: Mikolov et al. [10] point out that Word2Vec should be trained on a large-scale data corpus. Therefore, we cannot utilize our data set to train Word2Vec due to its relatively small data scale (only 31,597 user reviews and 3,272 issue reports). In order to guarantee Word2Vec worked well, we collect a 12.2G data corpus from Wikipedia [11] to train Word2Vec. This large-scale data corpus includes abundant words and their sematic forms. Then we use pre-defined Word2Vec to implement cosine similarity measure which is used to build the links between user feedback clusters and issue reports.

We evaluate the performance of our approach and baselines by using the following metrics:

- **Top-N**: this metric counts the number of feedback clusters in which at least one source code class related to change request was found and ranked in top-N (N=1, 3, 5). For examples, given a cluster of user feedback, if the top-N ranking results contain at least one class in the ground truth, we regard that

10. These forums include journal articles, conference/workshop papers, books, blogs, emails, etc.

11. https://dumps.wikimedia.org/enwiki/latest/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

the bug or the feature request has be localized successfully in the top-N rank.

- **Precision**: the metric is defined by $\frac{TP}{TP+FP}$. TP (*i.e.*, True Positive instances) indicates the number of top-5 instances (*e.g.*, classes to be changed) recommended correctly, FP (*i.e.*, False Positive instances) represents the number of top-5 instances recommended incorrectly.
- **Recall**: the metric is defined by $\frac{TP}{TP+FN}$. FN (*i.e.*, False Negative instances) means the number of correct instances that are not recommended at top-5 ranking list by the approach.
- **Mean Reciprocal Rank (MRR)**: this metric is defined as the multiplicative inverse of the rank of first correctly returned class within the top-5 results. Therefore, MRR averages such measures for all queries in the dataset, *i.e.*, $\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{Rank_i}$. Here, $Rank_i$ is the rank of first correctly returned class within the top-5 results. $|Q|$ is the total number of queries (*i.e.*, clusters of user feedback). The higher the MRR value is, the better the performance of change localization is.
- **Mean Average Precision (MAP)**: this metric is different with MRR because it considers all ranked classes rather than the first correct class for each query. MAP is the mean of the average precision values for all queries. The average precision of a single query q is computed as $AvgP(q) = \sum_{i=1}^{M}\frac{P(j)\times rel(j)}{N_{positive}}$. Here, j is the rank in the list of returned top-5 results; M is the total number of retrieved classes; $rel(j)$ is a binary indicator to verify whether the i^{th} class is a correct object (*i.e.*, the value is 1) or not (*i.e.*, the value is 0); $N_{positive}$ presents the number of positive instances (*i.e.*, TP); and $P(j)$ is the precision at the given cut-off rank j, which is defined as $\frac{N_{positive}\ in\ top\ j\ ranks}{j}$. Then, MAP is defined as $\frac{\sum_{k=1}^{|Q|}AvgP(q_k)}{|Q|}$ where $|Q|$ is the total number of queries.

Among these evaluation measures, Top-N verifies whether an approach can successfully locate the source code classes related to change requests for more user feedback clusters while Recall verifies whether an approach can successfully locate more correct classes for the user feedback clusters. Precision, MRR, and MAP also mention the accuracy at top-5 ranking results. In our work, we care more about the Top-N and Recall values because the results are related to our motivation to verify whether our approach can locate more source code classes related to change request for more user feedback clusters than `CHANGEADVISOR`.

Parameter adjusting: In order to avoid adjusting excessive parameters that may increase additional computing cost, we design our approach with less parameters. In the first phase, we adopt HDP which shows the best clustering performance (See Section 5.2) to group user feedback. In HDP, we do not mention (or set) the number of topics in advance because it can group the elements based on their probability distribution. Therefore, we need not to adjust the number of topics. In fact, in our algorithm, we only tune one parameter, *i.e.*, $\theta_{feedback}$ which is a threshold value to decide whether the issue reports should be selected to link (or enrich) the user feedback. We adjust it in the light of the performance (*i.e.*, MRR) of change request localization. We set the value to 0.4 for achieving the highest arithmetic mean of MRR scores for all projects in our data set.

Our experiment includes two parts: *pilot study* (See Tables 6-12) and *real experiment* (See Table 21). In *pilot study*, we verify whether issue reports can link user feedback and whether enriched user feedback can improve the performance of change request localization. Thus, we do not group the data according to the time frame. In *real experiment*, we focus on implementing `Where2Change` in the real development environment. We utilize the historical issue reports to enrich the newly submitted user feedback in `Where2Change`. In the following sections, we show the experimental results.

5.2 Answer to RQ1: best decision

In the process of user feedback clustering, we select the following six popular clustering algorithms to cluster the user feedback:

- **LDA:** It is a type of topic model that uses groups of topic words to explain sets of documents. In LDA, each document in the corpus collection is presented as a mixture of latent topics, and each topic is represented by a series of words and their occurrence probability.
- **SentenceLDA:** It is an extension of LDA for incorporating part of text structure in the topic model. LDA and SentenceLDA differ in that the latter assumes a very strong dependence of the latent topics between the words of sentences, whereas the former assumes independence between the words of documents in general. In SentenceLDA, the text spans can vary from paragraphs to sentences and phrases depending on the different tasks' purposes. Therefore it can control the number of topics according to the different text spans in the documents.
- **CopulaLDA:** It extends LDA to incorporate the topical dependencies within sentences and noun-phrases using copulas which include a family of distribution functions which can offer a flexible way to model the joint probability of random variables using only their marginals. Using copulas can result in decoupling the marginal distributions by the underlying dependency so that it can help to improve the performance of LDA by integrating simple text structure in the topic model. Due to copulas that result in more flexibility than assigning the same topic in each term of the sentence which is illustrated in the performance difference between CopulaLDA and SentenceLDA. The former is more flexible and performs better.
- **HDP:** It is an extension of LDA. Different from LDA, HDP does not need to confirm the number of topics before starting a clustering process. It implements a nonparametric Bayesian approach which iteratively groups documents based on a probability distribution.
- **K-means:** As a classical clustering algorithm, it aims to group the documents into k clusters in which each document belongs to the cluster with the nearest mean.
- **DBSCAN:** It is a density-based clustering algorithm, which groups the documents lied in high-density regions in the vector space. Different from K-means, it does not require a parameter to define the number of clusters before starting a clustering process. Moreover, it can find irregular-shaped clusters.

We implement the above-mentioned clustering algorithms and apply them to our data set. Then, followed by the description at the literature [29], we compare their clustering performance with *gold standard* which consists of the clusters of user feedback generated by the manual way. We first invite the developers who worked in the appropriate app's team to help us cluster the user feedback via publicity mail addresses. However, no one is willing to do

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

this work. We receive a developer's response: "*It looks like a very complicated and time-consuming task, so I am afraid I cannot complete it even though you can pay.*" Therefore we adopt an alternative solution. In detail, we invite the same four developers who help us to create the ground truth (See Section 5.1). We also pay 100 RMB (equal to 14.5 USD) to each developer per day. After two weeks, they had clustered all selected user feedback. Then we invite the senior software test specialist from Alibaba Company to verify whether these user feedback entries were clustered accurately. He has the right to modify the errors by discussing with the above-mentioned four developers. Finally, the clustering result is called "*gold standard*".

For the evaluation process, we first use three external metrics that include Homogeneity, Completeness, and V score [29] to evaluate the algorithms' performance. The metrics are introduced as follows:

- **Homogeneity, Completeness, and V score:** Homogeneity is the ratio of user feedback in a single cluster belonging to the same cluster of *gold standard*. Completeness measures the ratio of user feedback in a same cluster of *gold standard* which are assigned to the same cluster produced by algorithms. The lower bound is 0 and the upper bound is 1. V score is the harmonic mean between Homogeneity and Completeness, which is defined by $2 \times \frac{Homogeneity \times Completeness}{Homogeneity + Completeness}$.

Table 4 shows the best results of the different clustering algorithms adopting the appropriate number of clusters shown in Table 5. Note that all values keep one decimal place. For example, the value of Homogeneity for DBSCAN in Automattic is 1.54e-016, and we list its approximate value-0.0 in the table. The values in the last line show the arithmetic means of all apps for each algorithm. Since the arithmetic means consider the influence of the different number of clusters to the performance of the different clustering algorithms, we use it to evaluate which cluster algorithm performs the best. Among the six clustering algorithms, HDP and DBSCAN can automatically select the appropriate number of clusters to group the user feedback. We find that HDP performs better than DBSCAN. The values of Homogeneity and V score of the former improve that of the latter by up to 10.9% and 5.9%, respectively. For some projects such as Automattic, UweTrottmann, and Wordpress, the values of Homogeneity are very close to 0 when DBSCAN is adopted. We find that this algorithm produces only one cluster for each of the above-mentioned apps while HDP produces relatively more number of clusters which result in better clustering performance. By comparing the best performances of other topic models that include LDA, SentenceLDA, CopulaLDA when they select the appropriate numbers of clusters, HDP also performs better than them. Specially, the values of Homogeneity, Completeness, and V score of HDP improve that of LDA by up to 11.9%, 7.3%, and 9.4%, which performs the lowest among all clustering algorithms.

Based on the above analysis and the values in Table 4, we can get a conclusion that HDP performs the best among six popular clustering algorithms, we select it to group the user feedback.

Note that the arithmetic mean values of Homogeneity, Completeness, and V score are all less than 25%. This indicates that for lots of user feedback, the results produced by the clustering algorithms are different from manual classification results. For example, the following two entries of user feedback "*It doesn't show up on my phone except to say it's been successfully installed*" (**R1**) and "In other words, you cant reverse sort order or manually configure it either" (**R2**) for AntennaPod are grouped by HDP into the same cluster while they belong to different groups in the gold standard (*i.e.*, produced by manual clustering). On the contrary, **R1** and another entry of user feedback "*Gpodder integration stopped working long ago, and now episode auto download doesn't work anymore*" (R3) are grouped by HDP into the same cluster in the gold standard, however HDP does not group them together. Thus, the results produced by the clustering algorithms are still not optimal. The quality and style of user reviews vary greatly [7], [24], [31], which makes automatic clustering of user feedback a difficult problem. However, this finding does not influence our conclusion on which clustering algorithm can produce the closest result with the gold standard.

TABLE 5: The number of clusters when the clustering algorithms achieve the best performance

Project	The number of clusters					
	LDA	SenLDA	CopLDA	HDP	K-means	DBSCAN
AntennaPod	6	16	10	36	24	10
Automattic	2	10	10	12	10	1
Cgeo	8	12	10	20	38	6
Chrislacy	16	6	14	8	14	15
K-9 mail	6	4	12	8	4	26
OneBusAway	4	16	16	34	16	14
Twidere	6	22	22	30	6	14
UweTrottmann	10	14	18	20	8	1
WhisperSystems	6	12	24	16	8	29
Wordpress	10	6	14	12	12	1

Although the clustering algorithms that we consider in this work are not optimal, the use of HDP helps in improving the accuracy of the main task that we investigate in this work, *i.e.*, localization of change requests for app reviews.

In order to verify which metric is the best to build a link between a cluster of user feedback and the issue reports, we first conduct a *pilot study* described in Section 5.1. Specifically, we use three metrics including Dice coefficient [25], tf·idf [26], and MCG [27] to re-implement `Where2Change` by replacing Word2Vec. Dice coefficient can be directly used to compute the similarity between two documents, and other three metrics are introduced to transfer the documents to different kinds of vectors so that they can be input into cosine similarity measure [32] to compute the similarity scores. We introduce them one by one as follows:

- **Dice coefficient:** Dice coefficient is a statistic used for comparing the similarity of two samples, thus it can be used to compute the similarity between a cluster of change-related user feedback and an issue report. It is defined as follows:

$$Sim(cluster_i, IR_j) = \frac{|Word_{cluster_i} \cap Word_{IR_j}|}{min(|Word_{cluster_i}|, |Word_{IR_j}|)}, \quad (6)$$

  where $Word_{cluster_i}$ is the set of words contained in the cluster i, $Word_{IR_j}$ is the set of words contained in issue report j, and the function min shown in the denominator normalizes the similarity between the cluster of feedback and the issue report via the number of words contained in the shortest document containing the fewest words.
- **tf·idf:** tf·idf is a popular metric to represent documents as vectors of words. The value for each word is its TF-IDF weight which is computed by formula (3) shown in Section 3.4. When we get all words' TF-IDF weights, a document can be transferred to a vector of the TF-IDF weights. Then we can use the cosine similarity measure (*i.e.*, formula (4)) to compute the textual similarity between the cluster of feedback $cluster_i$ and the issue report IR_j.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 4: Homogeneity, Completeness, V score (x%) of different clustering algorithms

Project	LDA			SenLDA			CopLDA			HDP			K-means			DBSCAN		
	H%	C%	V%	H%	C%	V%	H%	C%	V%	H%	C%	V%	H%	C%	V%	H%	C%	V%
AntennaPod	9.1	12.9	10.7	29.5	24.5	26.7	20.3	20.9	20.6	48.3	32.6	38.9	37.0	27.3	31.5	15.3	28.2	19.9
Automattic	2.9	8.5	4.3	15.8	13.9	14.8	13.6	12.5	13.1	15.3	16.1	15.7	14.8	13.2	14.0	0.0	100.0	0.0
Cgeo	5.3	6.9	5.9	10.0	8.4	9.1	7.4	6.8	7.1	13.7	10.6	12.0	23.9	14.2	17.8	2.9	7.5	4.1
Chrislacy	24.3	16.9	19.9	7.2	7.2	7.2	15.2	10.9	12.7	7.6	10.8	8.9	23.5	17.0	19.8	20.0	23.2	21.5
K-9 mail	2.3	3.1	2.6	1.5	2.2	1.8	3.7	3.3	3.5	2.8	3.1	2.9	1.1	2.1	1.4	4.5	10.0	6.2
OneBusAway	4.0	6.8	5.0	13.9	11.0	12.3	13.5	11.1	12.2	23.1	16.6	19.3	14.7	12.2	13.3	7.7	13.8	9.9
Twidere	4.9	5.8	5.3	14.2	10.3	11.9	15.2	11.4	13.0	17.9	13.9	15.7	4.1	6.1	5.0	4.5	9.8	6.1
UweTrottmann	7.5	7.7	7.6	12.7	10.1	11.3	13.0	10.1	11.3	15.5	12.3	13.7	6.6	7.0	6.8	0.0	100.0	0.0
WhisperSystems	6.0	7.5	6.6	11.1	9.8	10.4	22.3	16.5	18.9	13.7	12.4	13.0	8.7	9.5	9.1	15.4	20.0	17.4
Wordpress	2.6	3.1	2.8	1.7	2.3	1.9	4.1	3.9	3.9	2.8	3.3	3.1	3.2	3.4	3.3	0.0	100.0	0.0
Arithmetic mean	9.3	8.8	8.8	14.0	11.5	12.6	13.6	11.0	12.1	21.2	16.1	**18.2**	19.3	14.1	16.2	10.3	18.3	12.3

- **MCG:** MCG aims to map text format entities into semantic concept categories with some probabilities. It can also overcome the limitation in traditional token-based models such as tf·idf that only compares lexical words in the document. This metric captures the semantics of words by mapping words to their concept categories. By using MCG, a word can be represented as its semantic concept categories with probabilities. For example, consider the word "Microsoft", which can be categorized into a large number of concepts such as "company", "developer", and "software". Therefore, a word can be transferred to a concept vector so that a document can then be mapped into the space by:

$$C_d = \theta^T \cdot H^M, \tag{7}$$

where θ^T is the vector of the TF-IDF weights of the words in the document computed by formula (2) and H^M is the concept matrix. A concept matrix is constructed by concatenating the concept vectors of all words in the document. Via matrix multiplication, a document is transferred to a vector of concept categories, denoted as C_d. In fact, the document is mapped to the concept space by assigning a probability to each concept category to which the document belongs. This probability is estimated by summing up the corresponding probabilities of all the words contained in the document.
After we get the concept vectors of the cluster of feedback $cluster_i$ and the issue report IR_j, we can also use the cosine similarity defined by formula (4) to compute their concept similarity.

According to these three similar metrics, our approach produces three values: $Where2Change^{Dice}$, $Where2Change^{tf \cdot idf}$, and $Where2Change^{MCG}$. We evaluate the performance of change localization in order to find the best metric. Table 6 shows the values of Top-N (N=1, 3, 5) while Table 7 shows the values of Precision, Recall, MRR, and MAP for `Where2Change` and their varietas. Note that the results are produced by evaluating the feedback clusters which can link to issue reports. We do not consider the clusters unmatched to issue reports, because in this section focuses on the performance of our approach using the four similarity metrics to build the links between feedback clusters and issue reports.

In Table 6, the arithmetic mean Top-N (N=1, 3, 5) values of $Where2Change$ are larger than that of other varietas . $Where2Change^{MCG}$ is the second-best due to their arithmetic mean Top-N values which are close to the values of $Where2Change$. In Table 7, for the arithmetic mean of recall values, $Where2Change$ is the best while $Where2Change^{MCG}$ is the second-best due to the slight difference (57.58%-54.66%=2.92%); for the arithmetic mean of MRR and MAP values, $Where2Change^{MCG}$ is the best but the differences with other metrics are not obvious. For Precision, $Where2Change^{tf \cdot idf}$ is the best but the differences with $Where2Change$ and $Where2Change^{MCG}$ are less than 9%.

We analyze the possible reasons for the evaluation results. Since $Word2Vec$ and MCG also preserve terms' semantic and syntactic relationships [10], [27], the feedback clusters can link to more relevant issue reports. More detailed descriptions about software faults and feature requests in these issue reports result in that a greater number of change requests appearing in feedback clusters are successfully located. Therefore, the Top-N values of $Where2Change$ and $Where2Change^{MCG}$ are much larger than that of $Where2Change^{Dice}$ and $Where2Change^{tfidf}$ which do not consider the terms' semantic concepts. The precision values of $Where2Change^{Dice}$ and $Where2Change^{tfidf}$ are slightly larger than that of $Where2Change^{Word2Vec}$ and $Where2Change^{MCG}$. The result reveals that our approach using $tf \cdot idf$ and $Dice$ can recommend more correct classes in top-5 results than other metrics. This fact indicates that the terms cannot always match to other terms with the same or similar semantic concepts. In other words, some terms may match to the wrong terms which have the different meaning with them. This reason results from the fact that the precision values of our approach using $Word2Vec$ and MCG is not higher than that using $tfidf$ and $Dice$. However, according to our motivation described in Section 2.2, we expect that the new change request localization approach can locate more source code classes related to change requests for more user feedback clusters. Much larger Top-N and recall values demonstrate that $Word2Vec$ and MCG are appropriate candidate metrics to implement our goal.

For each app in our data set, we find that not all issue reports are linked to the clusters of user feedback. Table 8 shows how many issue reports can be actually linked to feedback clusters when we use the different metrics. The data on last column shows the total number of issue reports in our data set shown in Table 2. In this table, we note that the numbers of issue reports linked to the feedback clusters using $Word2Vec$ and MCG are larger than that using $Dice$ and $tf \cdot idf$. Because the former metrics consider the semantic information of issue reports and user feedback, a greater number of issue reports are linked to the clusters of user feedback. In addition, we find that the number of issue reports linked to feedback clusters using $Word2Vec$ is more than that using MCG. This finding explains that why our approach using $Word2Vec$ can successfully locate a larger number of clusters of user feedback than that using MCG (See Table 6).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 6: Top-N (T1, T3, T5) values comparison using different similarity metrics for the feedback clusters linked to issue reports

Project	$Where2Change^{Dice}$			$Where2Change^{tf\cdot idf}$			$Where2Change^{MCG}$			$Where2Change$		
	T1	T3	T5	T1	T3	T5	T1	T3	T5	T1	T3	T5
AntennaPod	18	31	32	11	27	30	23	32	32	29	35	35
Automattic	3	5	7	4	7	9	10	10	10	12	12	12
Cgeo	20	20	20	20	20	20	20	20	20	20	20	20
Chrislacy	2	4	4	4	7	7	6	8	8	7	7	7
K-9 Mail	0	2	3	1	3	5	3	5	6	5	6	6
OneBusAway	32	32	32	29	29	29	31	31	31	33	33	33
Twidere	19	23	25	20	24	26	27	30	30	25	28	28
UweTrottmann	11	18	19	10	25	26	15	28	29	21	28	28
WhisperSystems	14	15	15	9	14	14	13	15	15	15	16	16
Wordpress	9	9	9	12	12	12	11	11	12	12	12	12
Arithmetic mean	16.6	20.8	21.6	14.8	21.2	22.3	19.8	24.0	24.2	**22.2**	**24.8**	**24.8**

TABLE 7: Precision (P), Recall (R), MRR (MR), and MAP (MA) values (%) comparison using different similarity metrics for the feedback clusters linked to issue reports

Project	$Where2Change^{Dice}$				$Where2Change^{tf\cdot idf}$				$Where2Change^{MCG}$				$Where2Change$			
	P	R	MR	MA	P	R	MR	MA	P	R	MR	MA	P	R	MR	MA
AntennaPod	55.91	47.28	22.82	61.09	58.73	33.64	21.94	59.12	51.43	65.46	23.20	61.14	50.99	69.99	21.45	57.50
Automattic	54.17	48.15	20.47	63.82	54.84	62.97	30.05	75.45	45.00	66.67	30.16	73.12	42.22	70.38	31.72	76.03
Cgeo	84.76	26.89	40.82	92.09	90.91	13.60	41.89	93.92	84.98	32.48	40.70	91.93	85.00	35.96	39.62	90.05
Chrislacy	68.75	13.93	29.25	75.47	67.50	34.18	28.01	71.84	81.09	75.95	30.46	75.01	75.00	83.55	31.68	76.46
K-9 Mail	22.58	9.86	4.59	18.18	33.33	9.86	10.62	35.44	37.68	36.63	7.42	25.04	36.67	46.48	10.65	34.48
OneBusAway	75.36	52.53	42.92	96.14	90.28	32.83	44.60	99.05	68.25	65.16	42.38	94.96	67.86	67.18	42.39	95.01
Twidere	34.29	18.54	25.46	66.83	41.96	18.15	23.35	61.97	34.78	27.80	29.63	74.37	37.14	30.12	29.91	74.04
UweTrottmann	52.00	39.01	20.45	55.84	47.31	44.01	18.37	51.09	48.30	71.00	18.43	51.32	46.36	70.00	16.92	48.37
WhisperSystems	75.61	33.52	33.20	81.07	73.02	24.87	33.55	82.30	63.64	56.76	31.52	77.66	62.36	60.00	32.17	78.73
Wordpress	87.64	17.07	41.73	92.95	92.77	16.85	43.79	96.93	78.32	38.74	43.06	96.17	76.33	40.92	41.83	94.13
Arithmetic mean	60.68	35.25	29.38	72.41	**65.18**	29.96	29.75	72.86	57.34	54.66	**30.18**	**73.37**	56.61	**57.58**	29.84	72.63

TABLE 8: Number of issue reports linked to feedback clusters when using different metrics

Project	# **RP** linked to feedback clusters				# **RP**
	$Dice$	$tf\cdot idf$	$Word2Vec$	MCG	
AntennnaPod	55	31	102	91	114
Automattic	7	20	70	45	95
Cgeo	541	89	1307	1115	1488
Chrislacy	5	24	127	44	153
K-9 Mail	7	7	52	41	58
OneBusAway	95	30	258	222	271
Twidere	34	33	108	84	117
UweTrottmann	30	26	107	89	114
WhisperSystems	56	21	199	171	209
Wordpress	37	37	506	396	653

TABLE 9: Number of feedback clusters (ratio) unmatched to issue reports when using different metrics

Project	# clusters (ratio) unmatched to issue reports				# clusters
	$Dice$	$tf\cdot idf$	$Word2Vec$	MCG	
AntennnaPod	4(11.1%)	6(16.7%)	1(2.8%)	4(11.1%)	36
Automattic	5(41.7%)	3(25.0%)	0(0.0%)	2(16.7%)	12
Cgeo	0(0.0%)	0(0.0%)	0(0.0%)	0(0.0%)	20
Chrislacy	4(50.0%)	1(12.5%)	0(0.0%)	1(12.5%)	8
K-9 Mail	4(50.0%)	3(37.5%)	2(25.0%)	2(25.0%)	8
OneBusAway	2(5.9%)	5(14.7%)	3(8.8%)	1(2.9%)	34
Twidere	3(10.0%)	2(6.7%)	0(0.0%)	2(6.7%)	30
UweTrottmann	11(36.7%)	2(6.7%)	1(3.3%)	1(3.3%)	30
WhisperSystems	1(68.8%)	2(12.5%)	1(6.3%)	0(0.0%)	16
Wordpress	2(16.7%)	0(0.0%)	0(0.0%)	0(0.0%)	12

In addition, we note that not all clusters can link to issue reports. Table 9 shows the number of feedback clusters and the ratio which cannot link to issue reports when using difference metrics. When using $Word2Vec$, the number of feedback clusters unmatched to issue reports is the smallest while the number is the largest when using $Dice$. Overall, the number of clusters not linked to issue reports is much smaller than that linked to issue reports. Therefore, the effect of their results to all evaluation results is limited. Table 10 shows the Precision, MRR, and MAP values of our approach using difference metrics for the feedback clusters unmatched to issue reports. Our approach using $Dice$ shows the best performance than other metrics. The reason is that there are largest number of feedback clusters not linked to issue reports when using $Dice$. When using $Word2Vec$, our approach performs the worst because there are 5 projects that have no clusters unlinked to issue reports and other 5 projects that have less than or equal to 3 clusters unmatched to issue reports. This fact also demonstrates that $Word2Vec$ can help user feedback link more relevant issue reports so that more user feedback clusters are linked to correct source code classes that should be changed. Therefore, we select $Word2Vec$ to find the relevant issue reports for enriching user feedback clusters so that it can improve the performance of change request localization.

In our approach, we propose weight selection-based cosine similarity metric to compute textual similarity between enriched user feedback clusters and classes to be changed instead of classic cosine similarity metric. Therefore, we should verify whether our approach using weight selection-based cosine similarity metric (*i.e.*, $Where2Change$) performs better than that using classic cosine similarity metric (*i.e.*, $Where2Change_{cosine}$). We re-implement our approach using classic cosine similarity metric with four similar metrics introduced in Section 3. The result of performance for each project in our data set is shown in Table 11.

In Table 11, we find that the Top-N values of our approach using cosine similarity are slightly less than that using weight selection-based cosine similarity (See Table 6). This result indicates that the two similarity measures can successfully locate the Top-N classes for the similar number of user feedback clusters. However, we note that the Precision, Recall, MRR, and MAP values of our approach using weight selection-based cosine simi-

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 10: Precision (P), MRR (MR), and MAP (MA) values (%) comparison using different similarity metrics for the feedback clusters unmatched to issue reports

Project	$Where2Change^{Dice}$			$Where2Change^{tf \cdot idf}$			$Where2Change^{Word2Vec}$			$Where2Change^{MCG}$		
	P	MR	MA	P	MR	MA	P	MR	MA	P	MR	MA
AntennaPod	80.00	37.83	91.67	33.33	5.27	23.61	0.00	0.00	0.00	80.00	22.83	20.10
Automattic	100.00	20.00	80.00	0.00	0.00	0.00	0.00	0.00	0.00	80.00	20.83	33.96
Cgeo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Chrislacy	90.00	22.83	66.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
K-9 Mail	100.00	11.42	25.00	20.00	2.23	11.11	10.00	5.01	5.00	20.00	3.34	10.00
OneBusAway	100.00	10.00	50.00	44.44	29.46	88.44	0.00	0.00	0.00	20.00	13.56	8.33
Twidere	25.00	5.56	19.44	25.00	10.00	50.00	15.00	4.99	5.00	0.00	0.00	0.00
UweTrottmann	65.00	20.79	67.77	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
WhisperSystems	0.00	0.00	0.00	40.00	5.33	18.33	0.00	0.00	0.00	0.00	0.00	0.00
Wordpress	100.00	22.83	50.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Arithmetic mean	**76.94**	**19.43**	**59.40**	**25.51**	9.01	31.41	6.25	2.50	2.50	44.62	13.87	14.87

TABLE 11: Performance of our approach using cosine similarity measure

Project	$Where2Change_{cosine}$						
	T1	T3	T5	P	R	MR	MA
AntennaPod	30	32	34	22.86	36.37	4.29	19.60
Automattic	12	12	12	30.65	70.38	18.38	52.14
Cgeo	20	20	20	53.17	39.28	18.22	56.58
Chrislacy	7	7	7	48.65	68.36	13.52	45.13
K-9 Mail	2	5	6	10.26	16.91	2.50	10.25
OneBusAway	30	33	33	47.27	47.98	11.06	37.99
Twidere	21	26	27	13.20	10.04	5.95	27.21
UweTrottmann	21	26	28	21.14	41.00	4.37	18.48
WhisperSystems	13	16	15	25.18	37.84	7.63	27.12
Wordpress	12	12	12	38.71	26.26	17.01	60.65
Arithmetic mean	21.0	23.7	24.4	34.02	30.24	37.40	9.12

larity are much better than that using cosine similarity. This fact indicates that weight selection-based cosine similarity measure can help our approach recommend more accurate classes in top-5 ranking results than cosine similarity measure. The major reason is that weight selection-based cosine similarity adopts the terms' best weights to implement change request localization so that it enhances the effect of important terms to the performance, therefore, our approach using this metric can get the higher accuracy (Precision, MRR, and MAP) and Recall values than that using classic cosine similarity metric.

By comparing the evaluation results of our approach using weight selection-based cosine similarity and classic cosine similarity, we get a conclusion that using the proposed weigh selection-based cosine similarity function can help to recommend more accurate classes in top-5 ranking results than using classic cosine similarity measure.

According to overall evaluation and analysis results for each step of our approach, we answer **RQ1** as follows:

> **Answer to RQ1**: Our approach selects HDP, Word2Vec, and the weight selection-based cosine similarity measure as user feedback clustering, feedback cluster enrichment, and class ranking algorithms, respectively due to their preferable performance on change request localization.

5.3 Answer to RQ2: performance comparison

In order to fairly compare the performance of change request localization using CHANGEADVISOR and our approach, we adopt the user feedback selected by the classification approach introduced in Section 3.2 to re-implement CHANGEADVISOR. In the literature [33], Palomba et al. set the threshold value to 0.5 for all projects. To demonstrate whether our approach performs better than CHANGEADVISOR, we adjust the threshold value from 0.1 to 1 to obtain all evaluation results of CHANGEADVISOR. The following table (*i.e.*, Table 12) show the best evaluation results of CHANGEADVISOR. In addition, for our approach Where2Change, not all clusters can link to issue reports used to enrich them. We also consider the feedback clusters which cannot link to issue reports to compare the performance of change request localization between CHANGEADVISOR and our approach. Note that the weight-selection cosine similarity measure is not suitable for computing the similarity between these clusters and classes to be changed. Because this algorithm depends on the linked issue reports to get the best weights of terms. Therefore, we still use the similarity metric-$Word2Vec$ to compute the similarity scores between these feedback clusters not linked to issue reports and source code classes. Table 12 shows the Top-N values, Precision & Recall values, and MRR & MAP values of Where2Change and CHANGEADVISOR for all clusters in our data set.

Expect for the arithmetic mean values of MAP, we note that Where2Change performs better than CHANGEADVISOR. Especially for Top-N and Recall values, we find that the differences reach up to 17 for Top-1, 18.1 for Top-3, 17.9 for Top-5, and 50.08% for Recall. For the arithmetic mean MAP values, Where2Change is not better than but very close to CHANGEADVISOR. The difference is only 3.39% (75.89%-72.50%=3.39%).

In order to verify whether our approach can successfully locate more source code classes that should be changed for more user feedback clusters, we use Wilcoxon test [34] in the R environment [35] to further compare the performance between CHANGEADVISOR and our approach. If a p-value is more than the significance level, we accept the null hypothesis; otherwise, we reject it. We adopt the default value (*i.e.*, 0.05) as the significance level. We define the null hypothesis as follows:

- Our approach shows no noteworthy difference against the previous study CHANGEADVISOR.

Next, we introduce the Top-N, Recall, Precision, MRR, MAP values of 10 projects as the input data and perform Wilcoxon test. As a result, we get the corresponding p-values. We list them at Table 13.

We note that the p-values of Top-N and Recall are less than 0.05. In this situation, we reject the null hypothesis. Therefore, our approach can *significantly* improve the performance of change request localization for user feedback by comparing with the previous work CHANGEADVISOR. In other words, our approach can successfully locate more source code classes to be changed for more user feedback clusters than CHANGEADVISOR. In addition,

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 12: Performance comparison between Where2Change and CHANGEADVISOR

Project	*Where2Change*							*CHANGEADVISOR*						
	T1	T3	T5	P	R	MR	MA	T1	T3	T5	P	R	MR	MA
AntennaPod	29	35	35	50.99	69.99	21.45	57.50	7	8	9	56.52	11.82	30.37	87.14
Automattic	12	12	12	42.22	70.38	31.72	76.03	3	4	4	50.00	14.81	25.17	85.73
Cgeo	20	20	20	85.00	35.96	39.62	90.05	4	4	4	84.62	1.66	33.08	99.99
Chrislacy	7	7	7	75.00	83.55	31.68	76.46	1	5	6	46.67	8.86	19.00	60.00
K-9 Mail	5	8	8	31.19	47.89	10.31	33.02	5	9	9	28.57	11.27	19.30	64.28
OneBusAway	33	33	33	67.86	67.17	42.40	95.01	6	7	7	56.51	6.57	24.85	70.05
Twidere	27	31	31	35.59	30.50	29.71	73.47	5	8	8	36.00	3.47	20.14	69.18
UweTrottmann	21	28	28	46.36	70.00	16.92	48.37	8	11	11	54.55	12.00	26.63	78.89
WhisperSystems	15	16	16	62.36	60.00	32.17	78.73	4	4	4	21.05	2.16	11.30	44.44
Wordpress	12	12	12	76.33	40.92	41.83	94.13	5	7	7	85.71	3.93	31.88	78.28
Arithmetic mean	**22.5**	**25.3**	**25.3**	**56.17**	**57.69**	**29.80**	72.50	5.5	7.2	7.4	53.07	7.61	25.12	**75.89**

TABLE 13: The result of Wilcoxon test

Evaluation approach	p-value	Result
Top-1	0.00088	Reject
Top-3	0.00361	Reject
Top-5	0.00407	Reject
Recall	0.00016	Reject
Precision	0.65015	Accept
MRR	0.19876	Accept
MAP	1.00000	Accept

the p-values of Precision, MRR, and MAP are more than 0.05. Thus, we accept the null hypothesis. This result indicates that the accuracy in top-5 ranking results of our approach has no significantly difference with CHANGEADVISOR.

According to the above experimental result, we get a conclusion that our approach can successfully locate more source code classes that should be changed for more user feedback clusters than the previous study CHANGEADVISOR and keep the similar accuracy in top-5 ranking results. The major reason is that issue reports can enrich user feedback clusters due to their detailed descriptions for the software faults and feature requests so that the performance of change request localization is improved. In addition, we propose the weight selection cosine similarity measure which adopts the best weights of terms to compute the similarity between user feedback clusters and source code, which results in that more source code classes related to change requests are successfully located for more user feedback clusters.

The previous IR-based fault localization technologies utilize issue reports as queries to search where should the bugs be fixed. In this section, we utilize BLUiR [5] and BLIA [6] to resolve the problem in our work. In the literature [8], Palomba et al. compared the performance of CHANGEADVISOR and BLUiR. Therefore, we also select is as one of our baselines. Moreover, we also select BLIA as another baseline due to the two reasons. On the one hand, Youm et al. indicate that BLIA outperforms the existing tools such as BLUiR and BugLocator [4] because this approach considers the multiple data resources that include texts and stack traces in issue reports, structured information of source files, and source code change histories; On the other hand, BLIA is an open source tool, which is easily employed to implement our task.

To fairly compare the performance of our approach and the baselines, it is necessary to guarantee that the same queries (*i.e.*, user feedback clusters) are used to conduct change request localization. In our work, we utilize the weight-selection cosine similarity measure to compute the textual similarity scores between the enriched feedback clusters and source code classes. Therefore, we also use baselines to re-implement this task. Table 14 and Table 15 show the evaluation results of BLIA and BLUiR.

TABLE 14: Evaluation result of BLIA

Project	*BLIA*						
	T1	T3	T5	P	R	MR	MA
AntennaPod	21	31	32	43.21	55.80	18.76	52.11
Automattic	10	11	11	33.94	66.78	26.06	72.30
Cgeo	15	15	15	86.15	30.67	30.15	82.58
Chrislacy	6	8	8	70.19	80.22	28.79	70.13
K-9 Mail	2	4	4	27.56	40.21	16.78	31.08
OneBusAway	24	28	28	58.19	62.66	35.57	89.04
Twidere	22	22	25	33.10	25.89	25.84	61.13
UweTrottmann	20	22	22	44.34	61.26	11.13	42.15
WhisperSystems	13	13	14	59.88	55.36	30.15	77.21
Wordpress	10	11	11	83.15	33.14	37.21	89.76
Arithmetic mean	17.7	20.7	21.3	52.06	50.56	25.26	66.23

TABLE 15: Evaluation result of BLUiR

Project	*BLUiR*						
	T1	T3	T5	P	R	MR	MA
AntennaPod	21	30	31	40.21	51.36	13.16	46.68
Automattic	8	9	9	30.11	62.48	19.89	70.18
Cgeo	12	13	13	78.95	26.07	30.32	77.26
Chrislacy	4	5	7	63.20	72.19	23.57	63.19
K-9 Mail	3	4	4	22.30	31.26	14.67	28.77
OneBusAway	25	26	26	47.05	55.69	30.20	80.09
Twidere	20	23	24	30.15	19.14	22.15	62.46
UweTrottmann	17	18	20	40.18	58.88	10.32	33.68
WhisperSystems	11	11	11	52.35	49.13	22.46	68.48
Wordpress	9	10	11	79.31	27.51	30.22	82.57
Arithmetic mean	16.5	19.1	19.8	46.46	45.14	21.11	60.67

When we compare the result of our approach shown in Table 12 with that of BLIA and BLUiR, we find that our approach performs better than these baselines. This fact indicates that our approach using weigh-selection approach can locate more correct source code classes related to change requests for more user feedback clusters with the higher accuracy in top-5 results. The major reason is that selecting the best terms' weight is the most important factor than other data resources such as change history and structure of source code when we conduct the task of change request localization.

According to above results of evaluation and analysis, we can answer the research question **RQ2** as follows:

> **Answer to RQ2**: Our approach can recommend more source code classes related to change requests for more user feedback clusters than the previous change request localization approach-CHANGEADVISOR. In addition, by re-implementing the previous fault localization approaches-BLIA and BLUiR on user feedback clusters, our approach also performs better than them.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

6 Discussion

6.1 Answer to RQ3: performance analysis

In Section 5.3, we demonstrate that our approach can locate more source code classes that should be changed for more user feedback clusters than the previous study CHANGEADVISOR. We analyze the reason in this section.

In this work, we utilize issue reports to enrich user feedback clusters for improving the performance of change request localization. This is the difference with CHANGEADVISOR. In order to demonstrate the importance of issue reports in our approach, we compare the performance of change request localization using our approach that utilizes issue reports and that removes issue reports. We name the latter as "$Where2Change_{-IR}$". For $Where2Change_{-IR}$, we only compute the similarity between a cluster of user feedback and source code by using the similarity metrics that include $tf \cdot idf$, $Word2Vec$, and MCG. We do not use $Dice$ because CHANGEADVISOR adopts it to conduct the same task and the result is shown in Table 12. The weight-selection cosine similarity measure cannot be used to conduct this task because there are no issue reports that are used to training for finding the terms' best weights. Table 16 and Table 17 show the result.

When we remove issue reports, using $Word2Vec$ can produce the highest Top-N values while using MCG can get the highest Precision, Recall, MRR, and MAP values. We note that our approach using issue reports (See Table 12) performs better than that removing issue reports due to the higher Top-N, Precision, Recall, MRR, and MAP values. Specially for Top-1 and Recall values, using issue reports improve 2.3 times (22.5/9.7=2.3) for Top-1 while it improves 2.6 times (57.69/22.51=2.6) for Recall values than our approach removing issue reports. In addition, when we compare the performance of our approach removing issue reports and CHANGEADVISOR, we find that Precision, MRR, and MAP values of the latter one is higher than that of the former one. Therefore, removing issue reports cut down the performance of change request localization for user feedback. This result indicates that issue reports can help our approach successfully locate more classes to be changed for more user feedback clusters.

Cases: In Table 9, we show the ratio of feedback clusters unmatched to issue reports when using different metrics. Note that for K-9 Mail, the ratio is higher than others. Except for Recall, we find that Top-N, MAP, and MRR values of CHANGEADVISOR are also higher than that of our approach. Specially for MAP value which is twice as much as that of our approach. For Precision, there is no significant difference between two approaches. Otherwise, for Cgeo, the unmatched ratio is 0%. We find that Top-N, Precision, Recall, and MRR values of our approach are also higher than that of CHANGEADVISOR, specially for Top-N and Recall values. For MAP value, the difference between two approaches is less than 10%.

The above-mentioned two cases demonstrated that our approach performs much better than CHANGEADVISOR when a lot of user feedback clusters can link to historical issue reports. Otherwise, if there are no enough historical issue reports which can enrich user feedback clusters, our approach cannot locate more accurate source code classes for more user feedback clusters by comparing with CHANGEADVISOR.

We explain why issue reports can help to improve the performance of change request localization. There are two reasons: 1) According to our previous investigation for issue reports in mobile apps [1], we find that issue reports contain the detailed information such as stack traces, code examples, and patches. The important information describes the clues why a bug appears or a feature request is proposed. By enriching user feedback clusters, they can assist our approach to locate more correct classes to be changed for more user feedback; 2) Issue reports can help us get the best terms' weights used to compute the similarity scores between user feedback clusters and source code classes so that our approach can successfully locate more source code classes related to change requests for more user feedback clusters.

According to above-mentioned experimental results and analysis, we can answer RQ3 as follows:

> **Answer to RQ3**: Issue reports can help our approach locate more source code classes related to change requests for more user feedback due to their detailed information and the contribution on finding the best terms' weights used to improve the performance of the cosine similarity measure.

6.2 Answer to RQ4: query verification

In this work, we adopt user feedback extracted from user reviews rather than issue reports as queries to conduct change request localization for mobile apps. In Section 6.1, we demonstrate that our approach using issue reports can locate more source code classes to be changed for more user feedback clusters. Thus, the question "*why do not adopt issue reports as queries to conduct change request localization if they can provide detailed information?*" is thrown out. We investigate the reasons in this subsection.

Different from issue reports, user reviews are posted by users who may have no or less experience on software development and debugging. However, user reviews reflect users' requirements for the next update of mobile apps. Especially for the user reviews related to bugs and feature requests, the users expect that developers can fix these bugs and add the appropriate features in the next version. Comparing with traditional desktop software, the update rate of mobile apps is more frequent. Users can choose high-quality apps which have good user experience in the short term. Therefore, satisfying users' requirements is a lifeline of mobile apps [36].

In the literature [7], Mcilroy et al. point out that mobile apps' developers usually depend on user reviews to resolve the issues and update the apps. This finding brings inspiration to us so that we want to understand developers' real behavior for resolving issues in mobile apps. Thus we send a brief questionnaire to top 200 most active developers who have the maximum number of times to resolve issues and to give the comments[12] in top 100 popular mobile apps via public mail addresses. They are invited to answer the question Q1 shown in Table 18.

As a result, we receive 98 responses (49% response rate). Among them, 71 (72.4%) developers select **Answer A** while 13 (13.3%) developers choose **Answer B**. For **Answer C**, 14 (14.3%) developers choose it. By analyzing the investigation result, we have two findings. On the one hand, we note that most of developers (71+13=84) worked for mobile apps depend on user reviews for finding and resolving issues. This finding is same as the report at the literature [7]. On the other hand, we note that 72.4% developers still depend on historical issue reports to help them locate source code classes related to change requests. We send

12. Some active developers may act both of issue assignees and commentators

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 16: Top-N (T1, T3, T5) values of our approach when removing issue reports

Project	$Where2Change_{-IR}^{tf\cdot idf}$			$Where2Change_{-IR}^{Word2Vec}$			$Where2Change_{-IR}^{MCG}$		
	T1	T3	T5	T1	T3	T5	T1	T3	T5
AntennaPod	12	17	18	16	32	36	10	24	32
Automattic	0	4	4	3	9	12	5	11	11
Cgeo	10	11	12	13	18	19	10	19	19
Chrislacy	0	2	2	6	8	8	4	5	7
K-9 Mail	1	2	2	2	3	4	3	3	7
OneBusAway	29	32	32	18	33	33	21	31	32
Twidere	3	7	10	5	14	20	6	13	20
UweTrottmann	3	4	7	6	23	28	10	20	25
WhisperSystems	0	6	8	4	12	12	3	11	12
WordPress	7	11	11	5	8	8	4	9	11
Arithmetic mean	9.2	12.4	13.7	**9.7**	**20.5**	**23.1**	9.5	18.3	22.0

TABLE 17: Precision (P), Recall (R), MRR (MR), and MAP (MA) values (%) of our approach when removing issue reports

Project	$Where2Change_{-IR}^{tf\cdot idf}$				$Where2Change_{-IR}^{Word2Vec}$				$Where2Change_{-IR}^{MCG}$			
	P	R	MR	MA	P	R	MR	MA	P	R	MR	MA
AntennaPod	31.71	11.82	2.93	13.00	43.01	36.36	16.48	49.15	40.45	32.73	21.40	62.70
Automattic	35.29	22.22	4.63	18.15	37.04	36.03	23.19	62.09	37.04	37.03	19.47	55.31
Cgeo	60.98	3.78	17.10	51.44	58.44	6.80	26.68	70.72	63.75	7.70	30.45	75.81
Chrislacy	50.00	7.59	4.54	13.07	48.15	16.46	18.75	52.43	73.33	27.85	34.25	84.58
K-9 Mail	11.11	2.82	3.33	16.67	23.53	11.27	12.46	46.32	16.22	8.45	8.21	31.32
OneBusAway	42.86	4.55	30.60	82.97	59.42	20.71	29.39	75.84	66.67	32.32	28.11	70.51
Twidere	18.00	3.47	4.49	18.56	22.52	9.65	8.31	33.31	17.58	6.18	9.19	32.15
UweTrottmann	20.59	7.00	2.93	13.00	40.51	32.00	15.42	48.39	36.08	35.00	15.96	48.48
WhisperSystems	16.13	2.70	4.25	19.17	27.27	9.73	12.17	40.35	28.17	10.81	13.40	43.19
WordPress	50.00	1.97	21.31	75.28	51.92	5.91	21.31	57.58	42.86	5.25	18.06	48.03
Arithmetic mean	32.75	6.73	10.45	33.64	42.26	20.77	18.53	53.81	**42.40**	**22.51**	**19.87**	**55.58**

TABLE 18: Questions and answers in the questionnaire

Q1: Which one is the real issue (i.e., software faults and feature requests) resolution process among the following three options in mobile apps that you worked?
A. I find the issues from user reviews, but I need to depend on historical issue reports to locate source code classes related to change requests so that I can resolve them.
B. I find the issues from user reviews, then I can directly locate source code classes related to change requests and resolve them without historical issue reports.
C. I do not refer to user reviews. I already wrote the issue reproducing test cases in issue reports and other developers are responsible for fixing the issues.
Q2: What is the process when you create an issue report?
A. I fix the bug and just create an issue report to record the maintenance task.
B. I already wrote the fault reproducing test cases, locate the source code classes related to change requests, and wrote the issue report.
C. I wrote the issue report just based on the user reviews and my own knowledge about the system.
D. I just fix the bug and cannot write the issue report.

the emails for these developers to ask the reasons. One developer worked at WordPress team told us: "*I can find some relevant source code files by utilizing some historical issue reports which describe the similar faults with new user reviews.*" Therefore, we think that selecting user reviews as queries can help to develop a full-automatic change request localization technology for saving the developers' time of reading and understanding thousands of user reviews in mobile apps. Moreover, historical issue reports can facilitate change request localization as the auxiliary role. We also have interests on the produce of historical issue reports. Thus, in the questionnaire, we require that the developers who select A or B to answer the question Q2 shown in Table 18.

We note that 52 developers (61.9%=52/84) select answer A while 30 developers (35.7%=30/84) select answer B. Only 2 developers select answer C and no one select answer D. In summary, these developers can also write the issue reports in order to help other developers use or refer to them for fixing the coming issues. Therefore, historical issue reports can continually produce, which can be used to improve the performance of our approach.

We also investigate the generation frequency (*i.e.*, how much time a new one is generated) of issue reports and user reviews. Table. 19 shows a comparison result of the generation frequency between issue reports and user reviews. We note that the generation frequency of user reviews is much faster than that of issue reports. The average generation frequency of user reviews is less than 1 day whereas that of issue reports reaches up to about 4 days. Thus, in the real debugging process for mobile apps, user reviews, which have high generation frequency, can facilitate developers to localize the change requests quickly in order to improve apps' performance timely.

TABLE 19: Comparison of generation frequency between issue reports and user reviews

Project	Generation frequency (days)	
	Issue report	User Review
AntennaPod	2.8	0.79
Automattic	7.41	0.88
Cgeo	0.88	0.43
Chrislacy	2.41	1.23
K-9 Mail	3.47	0.26
OneBusAway	5.31	1.25
Twidere	3.86	0.86
UweTrottmann	9.72	0.47
WhisperSystems	2.98	0.02
WordPress	0.81	0.16
Average	3.97	0.64

In Table 19, we note that the generation frequency of user reviews is much higher than that of issue reports. Therefore, a new coming user review may describe a new bug or a new feature request which is not described in historical issue reports even

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

though the similar issue is reported. We should verify whether historical issue reports can help to locate classes to be changed for new user reviews. We perform a *real experiment* introduced at Section 5.1 in order to achieve the goal. Before we start the experiment, the issue reports and user reviews are divided into two groups as the time frame. In detail, we collect the issue reports submitted before the corresponding timeline into **G1** and gather the user reviews posted after this timeline into **G2**. Each project has its own timeline. We decide each timeline as the following rules:

1) We choose the submission time of the last submitted issue report in each project as the reference point for deciding the corresponding timeline. In other words, we want to utilize as more historical issue reports as possible to locate changes for newly posted user reviews.
2) We keep a certain number (minimum is 2) of user feedback clusters to implement our approach. The major reasons is that few number of feedback clusters can lead to a meaningless results.

Table 20 shows the data scale of issue reports and user reviews in G1 and G2. Actually, verifying the timeline for each project as the above-mentioned rules is a trade-off problem. Utilizing more number of historical issue reports can reduce the number of user reviews used in our experiment so that the number of user feedback clusters is decreased as well. For example, in Automattic, when we put off the timeline from 01/2015 to 02/2015 to include more number of historical issue reports as option 1), we can only get one feedback cluster so that the evaluation result is meaningless. For this case, we violate option 2). On the contrary, if we bring forward the timeline from 01/2015 to 12/2014, the number of clusters still is 4. But we waste quite a few number historical issue reports so that we violate option 1). Therefore, we select 01/2015 as the timeline for Automattic. By the same token, we do not violate option 1) and option 2) when deciding the timeline for other projects. There are three special cases that include K-9 Mail, UweTrottmann, and WordPress. In these apps, when we put off the timelines shown in Table 20 by one month or more than one months, the number of clusters is decreased to below 50% but it is great than or equal to 2 while less than 10% number of historical issue reports are added. Therefore, even though these apps do not violate option 1) and option 2), we also do not put off the timelines because we should consider the cost performance. In other words, we must prevent the case that using more than or equal to 50% number of feedback clusters to trade less than 10% number of historical issue reports. According to the options and the reason to explain the three special cases, we verify the corresponding timeline for each project shown in Table 20.

When the experiment starts, we first extract the user feedback entries and cluster them in G2 to generate the feedback clusters, then use the issue reports in G1 to enrich the feedback clusters. Next, we re-implement our approach described in Section 3. For each project, we call our approach in this experiment as $Where2Change_T$ which utilizes the issue reports generated before the corresponding timeline to enrich the clusters of user feedback produced after this timeline for locating the changes appearing in the source code. T stands for the time frame. We also utilize `CHANGEADVISOR` to conduct change localization for the new coming user reviews in G2. The comparison results are shown in Table 21.

Overall, for new coming user reviews, our approach still can recommend more correct classes to be changed due to much higher recall values, the difference between $Where2Change_T$ and $CHANGEADVISOR_T$ reaches to 28.48%. From Precision, MRR, and MAP values, we find that there is no significant difference between two approaches. This finding is same as the comparison result shown in Section 5.3. For Top-N values, the differences are less than 1 between two approaches. The major reason is that few number of clusters can affect the performance of our approach. The result also demonstrated that our approach should be employed to more than 1400 user reviews which can generate more than 60 feedback clusters. We also explain it in Section 5.1.

TABLE 20: Data scale of issue reports and user reviews in the specific time frame

Project	# reports in G1	# reviews in G2	# clusters in G2	Timeline
AntennaPod	102	117	4	01/2016
Automattic	78	113	4	01/2015
Cgeo	460	367	2	01/2013
Chrislacy	45	79	2	07/2013
K-9 Mail	33	882	6	06/2016
OneBusAway	46	160	4	06/2015
Twidere	65	86	4	03/2016
UweTrottmann	87	324	6	01/2015
WhisperSystems	147	340	4	01/2016
WordPress	556	1,159	10	06/2015

Based on the evaluation result for new coming user reviews, historical issue reports can help to locate more correct classes. However, in the task of change localization for mobile apps, issue reports are treated as the secondary role which cannot replace user reviews as queries. Expect for the above-mentioned reasons, there is also a reason: enriched issue reports describe the similar but not same software faults or feature requests with user reviews. Thus they can only be used to enrich the corresponding user feedback clusters but cannot replace them. We list some examples shown in Table 22 to explain this fact.

The three examples are real cases in WordPress. For Example 1, both the user review and the issue report describe the problem of uploading. However, the former concerns the image while the latter focuses on the posts; For Example 2, both the user review and the issue report also describe the problem of uploading. However, the former concerns that the user cannot upload the data while the latter indicates that users may not see the uploaded data; For Example 3, both the user review and the issue report describe the problem of login. However, the former concerns that the user cannot log in the app while the latter presents that some parts of the app cannot work after login.

For each example, we note that the problem described in the user review and the corresponding issue report is similar. They are also related to the same source code class(es). We take an example, for issue report-# 862, we find that the class `WordPress` was changed for fixing the software fault reported in -# 862 by checking the commit. By investigating the ground truth, we find that there is a link between the feedback cluster that includes the user review-A and `WordPress`. Therefore, the historical issue reports indeed help to enrich the user reviews so that more correct classes are recommended to more user feedback clusters. However, issue reports cannot replace user reviews as queries because they may describe the different problems with reviews even though these problems are similar.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

TABLE 21: Performance comparison between `Where2Change` and `CHANGEADVISOR` for new coming user reviews

Project	$Where2Change_T$							$CHANGEADVISOR_T$						
	T1	T3	T5	P	R	MR	MA	T1	T3	T5	P	R	MR	MA
AntennaPod	4	4	4	65.63	38.19	27.18	68.99	4	4	4	66.67	7.27	33.08	93.02
Automattic	3	4	4	70.00	51.86	37.19	86.57	2	2	2	96.99	22.22	18.92	50.00
Cgeo	2	2	2	80.01	4.23	43.07	95.81	1	1	1	99.01	0.76	22.83	49.98
Chrislacy	2	2	2	77.09	46.84	37.09	85.64	2	2	2	88.89	10.13	40.67	90.21
K-9 Mail	1	5	5	31.95	32.40	5.03	19.84	3	4	4	64.71	15.49	32.00	87.01
OneBusAway	3	3	3	85.25	52.53	43.01	96.01	2	2	2	87.50	3.54	25.33	70.10
Twidere	4	4	4	67.02	25.10	40.86	93.26	3	3	3	57.14	1.54	22.92	72.92
UweTrottmann	2	5	5	63.16	36.00	13.49	40.94	4	5	5	56.25	9.00	28.50	70.07
WhisperSystems	1	4	4	90.01	34.06	25.73	67.88	1	1	1	60.00	1.62	7.67	18.89
WordPress	10	10	10	80.30	35.67	44.36	98.02	10	10	10	96.00	5.25	41.01	96.78
Arithmetic mean	4.0	5.3	5.3	69.55	*36.44*	30.67	73.01	4.3	4.5	4.5	76.84	*7.96*	28.95	74.14

TABLE 22: Examples to show the similar but not same contents of user reviews and issue reports

No.	User review	Issue report	Relevant class(es)
1	A: I've been trying for too long to upload an image but it just won't!	# 862: I've had support threads with two users so far that can't upload posts in 2.6...	`WordPress`

2	B: It shown that could not update data at this time.	# 3607: If you update your site's title and then return to the main menu, you'll still see the old title at the top of the page until you either tap Switch Site and re-select...	`MySiteFragment`
			`SiteSettingsFragment`
			`BlogUtils`

3	C: I love the site but hate this app because I can't log in to the app...	# 4798: ...When I log in with my email and a magic link, some parts of the app do not work...	`ThemeWebActivity`
			`StatsViewHolder`
			`ReaderActivityLauncher`

According to above results of evaluation and analysis, we can answer the research question **RQ6** as follows:

> **Answer to RQ4**: We select user reviews as queries rather than issue reports to conduct change localization due to the three reasons: 1) They can help to develop full-automatic change request localization technology; 2) They can help to successfully locate source code classes related to new change requests proposed by users, which are not found by developers; 3) The problems described in user reviews and issue reports may be similar but not same.

7 THREATS TO VALIDITY

In this section, we discuss some threats of our work from two aspects: external validity and validity.

7.1 External validity

We have only collected the data from ten mobile apps managed by GitHub to perform our experiments. These apps are selected according to stars' ranking in GitHub. In other words, we only consider the popular projects which have more stars provided by users. Thus our approach may not be generalizable to other projects. Even though we think that these popular projects are representative, we would like to further explore more projects in our future work. In addition, we just choose the mobile apps in GitHub as our experimental objects. Other mobile apps management systems such as Bitbucket also have the project hosting services to manage issues of mobile apps. Therefore we are not sure whether the proposed change localization approach can still keep the effectiveness for these apps. However, we think that this threat is reduced because Bitbucket mainly supports private (or business) projects which are different from open source projects.

7.2 Internal validity

7.2.1 Topic modelling

In this study, we utilize topic model to cluster user feedback. Topic modelling depends on data distributions in the data sets, therefore different data sets may affect the performance of topic modelling. However, the negative impact is small for our work. In Table 2, we find that the number of user feedback is more than 1000 for all projects (3 projects have more than 4000 user feedback). Therefore, plenty of user reviews reduce this threat.

7.2.2 User feedback

In this work, we adopt a cluster of user feedback as a query to search the corresponding classes that should be changed. Ideally, a user feedback entry should be treated as a query. However, we find that the accuracy is not acceptable (*i.e.*, very low accuracy). The major reason is that the information containing in single feedback is not enough. Moreover, some feedback entries describe a same or similar issue. Thus, considering clusters of user feedback as queries also can get the correct classes to be changed for developers so that they can reduce their workload.

7.2.3 Ground truth

We invite four developers who have more than 10 years software programming and testing experience to build the ground truth that includes the links between user feedback clusters and source code classes. Obviously, this is a difficult and challenge task. In order to ensure the credibility of the ground truth, we also invite the senior software test specialist who has more than 15 years software testing experience from Alibaba Company to verify whether these source code classes are linked to the given user feedback clusters accurately. Thus we believe that the threat for the credibility of the ground truth is reduced. Moreover, in order to further assess the quality of the ground truth, we invite the top-10 active developers to verify what the hitting rate and the missing rate are. One developer from Automattic helps to check the links in the app. We find that the hitting rate is 97.06% while the missing rate is 5.71%. However, we are not sure what the hitting rate and the missing rate are in other apps. It may be a threat, but we believe that the threat is not large since Automattic has characteristics similar to many other mobile apps [1].

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

We open our ground truth for all developers and researchers so that they can help to perfect it. In addition, we also expect that more developers and researchers can join us to build more links between user feedback clusters and source code classes for a greater number of apps.

8 Related work

In this section, we introduce some previous studies related to our work. These studies concern spectrum-based fault localization, IR-based fault localization, spectrum and IR-based fault localization, reviews-based fault localization, and software maintenance for mobile apps.

8.1 Fault localization techniques

Fault localization is an important foregoing task of bug fixing. Wong et al. [37] provide a systematic survey of such fault localization techniques and discuss some key issues and future directions. In this survey, they created a publication repository that includes 331 papers and 54 Ph.D. and Masters theses on software fault localization techniques, and analyzed the different evaluation metrics for these techniques. Moreover, they found that the factors such as overhead for computing the suspiciousness of each program component, time and space for data collection, human effort, and tool support should also be included in the contributions of automated fault localization techniques.

8.1.1 Spectrum-based fault localization

Spectrum-based fault localization techniques can help developers locate the software faults by testing a small portion of code. These methods analyze a program spectra that includes a series of program elements, and rank these elements so that they can achieve the purpose of fault localization. Tarantula [38] and Ochiai [39] are two early-stage automatic localization techniques, which utilized the different suspiciousness formulas. Abreu et al. [39] demonstrated that Ochiai performed better than Tarantula [38]. Xie et al. [40] analyzed the different suspiciousness score formulas and proposed a new method which did not require the existence of testing oracles to enhance the performance of spectrum-based fault localization. Lucia et al. [41] incorporate data fusion methods to design Fusion Localizer to normalize the suspiciousness scores of different fault localization techniques, selects fault localization techniques to be fused, and combines the selected techniques using a data fusion method.This approach requires no training data but improves the effectiveness of fault localization. Laghari et al. [42] propose a variant of spectrum-based fault localization, *i.e.*, patterned spectrum analysis. In detail, they utilize patterns of method calls by means of frequent itemset mining. The experimental results show that the proposed method is effective in localising the faults. Perez et al. [43] proposed a metric named DDU to increase the value generated by creating thorough test-suites so that the proposed method can help widely-used spectrum-based fault localization techniques to accurately pinpoint the location of software faults.

In a word, the spectrum-based fault localization approaches require program execution traces so that these methods increase the computational cost and require more data resources.

8.1.2 IR-based fault localization

In recent years, IR-based fault localization techniques attract more attention due to their low cost and easy-to-access data resources (*e.g.*, requiring only issue reports and source code files).

Lukins et al. [12] proposed a LDA-based approach to locate the buggy files for over 300 bugs in 25 versions of Eclipse and Rhino. Nguyen et al. proposed BugScout [3], a topic model-based automatic localization technique to help developers reduce the workload by narrowing the search space of buggy files. Rao and Kak [13] compared the performance of different IR-models, including Unigram Model (UM), VSM, Latent Semantic Analysis Model (LSA), LDA, and Cluster Based Document Model (CBDM) when performing the task of bug localization. Zhou et al. [4] proposed BugLocator to rank all source code files based on the textual similarity between a new issue report and the source code using a revised VSM. They also consider to combine the similarity between the given issue report and the similar bugs for improving the accuracy of fault localization. Kim et al. [14] proposed a two-phase recommendation model. In this model, they adopted Naive Bayes to filter out the uninformative issue reports and predict the buggy file for each submitted issue report. Thung et al. [15] develop an online tool to support fault localization for helping developers find where the bug is from the project's source code repository. Kochhar et al. [16] analyze the potential biases in fault localization. Authors mainly focus on what content (*e.g.*, bug, feature request, or code update) an issue report describes. Lam et al. [17] proposed an information retrieval approach which combines deep learning to locate the buggy files for issue reports.

Except for issue reports and source code, some studies also consider the more data sources and the structure of source code. Saha et al. develop BLUiR [5] which build AST to extract the program constructs of each source code file, and utilize Okapi BM25 to calculate the similarity between the given issue report and the constructs of each candidate buggy file. Ye et al. [18] leverages project knowledge through functional decomposition of source code, API description of library components, the bug-fixing history, the code change history, and the file dependency graph to generate a ranking list of source files scored by a weighted combination of an array of features. Wang et al. [19] utilize version history, similar reports, and code's structure to locate the buggy files. Chaparro et al. [20] adopted observed behavior to reformulate queries for improving the performance of information retrieval-based fault localization. Youm et al. proposed BLIA [6] which utilized texts and stack traces in issue reports, structured information of source files, and source code histories to conduct fault localization.

Our approach also utilizes IR-based localization technique to locate changes. However, there are some differences with the previous studies. First, our approach adopts user feedback clusters as queries rather than issue reports to locate source code classes that should be changed. Second, we focus on mobile apps rather than desktop software. Finally, we propose a weight-selection cosine similarity to compute the similarity scores between queries and source code.

8.1.3 Spectrum and IR-based fault localization

Le et al. [21] build a multi-model technique that consider both issue reports and program spectra to localize bugs. This work addresses the limitation of existing studies by analyzing both issue reports and execution traces. Because this method adopts more data resources (*i.e.*, issue reports and program spectra) to execute

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

fault localization, the results show that it performs better than the IR-based localization technique [18] and the spectrum-based localization technique [44].

Even if the hybrid model can improve the performance of fault localization. However, program spectrum should take as input a faulty program and two sets of test cases (*i.e.*, failed test cases and passed test cases). Thus, this approach needs more computing cost. Our approach is a lightweight method. In the one hand, the data resources can be easily collected from Google Play Store and GitHub; in the other hand, the algorithm contains fewer parameters but gets good performance.

8.2 Reviews-based change localization

Palomba et al. [8] proposed a new approach that analyzes the structure, semantics, and sentiments of sentences including in user reviews to extract the useful information for localizing code changes in mobile apps. The experimental results show that this method presents higher accuracy than BLUiR [5].

To our best of knowledge, this is the only work that is similar to our study. Both two studies also consider user reviews to locate the code changes in mobile apps. However, there are some differences presented as follows:

- We adopt issue reports to enrich user feedback extracted from user reviews in order to improve the performance of change localization. The experimental results shown that our approach can successfully locate more changes for more user feedback clusters than `CHANGEADVISOR` [8] due to higher Top-N and Recall values.
- We propose an accurate similarity metric (*i.e.*, weight selection-based similarity function) to compute the similarity between the queries and source code. It performs better than classic cosine similarity metric due to the use of the best terms' weights.

8.3 Software maintenance for mobile apps

Software maintenance for mobile apps become an important task in industry since an increasing number of mobile apps have been developed, however, only few works in academia focus on this topic. Syer et al. [45] analyzed 15 most popular open source Android apps, and they found that the "best practices" of existing desktop software development cannot apply to mobile apps due to the different features. Bhattacharya et al. [46] conducted an empirical analysis of bug reports and bug fixing in open source Android apps. They investigated the quality of bug reports and analyze the bug-fixing process. In addition, they showed how differences in bug life-cycles of Bugzilla applications and Android apps of Google Code can affect the bug-fixing process. Zhou et al. [47] conducted a cross-platform analysis of bugs and bug-fixing process in open source projects of different platforms, including desktop, Android, and IOS. They analyzed the different attributes (*e.g.*, fixing time and severity) of bug-fixing process in these different platforms.

These studies on empirical analysis of bug reports and bug-fixing process of mobile apps provide the inspiration for starting our work. In [46], [47], the researchers mainly studied the mobile apps in Google Code. However, since Google Code is shutting down at 2016 [48], a growing number of mobile apps select GitHub as their management and issue tracking tool. Therefore, we select the mobile apps of Github as our study objects. Under the circumstances, the analysis results (*e.g.*, the length of description) in [46], [47] cannot be appropriate for our data set. In [1], we only performed an empirical analysis for the features of issue reports in two different platforms. In our work, we mainly focus on how to locate source code that should be changed for clusters of user feedback in mobile apps.

8.4 Review analysis

With a great number of mobile apps appear in online app stores such as Google Play Store, Apple App Store, or Windows Phone App Store, users can rate the apps using stars rating and text reviews. These reviews describe users' impressions, comparisons, and attitudes towards the apps. Thus, app store reviews can be used by developers as a feedback to facilitate the development process. Some studies tend to analyze users' reviews for extracting the useful information.

Chen et al. present `AR-Miner` [24], a novel computational framework, to perform comrehensive analysis for user reviews. They group the most of informative reviews by filtering noisy and irrelevant ones. Panichella et al. [23] propose an intent classifier to group the user reviews into the different categories: *Feature Request*, *Problem Discovery*, *Information Seeking*, *Information Giving* and *Other*. In their follow-up work, SURF [9] is proposed to summary reviews and recommend the useful changes. Scalabrino et al. develop CLAP [49], a web application, to categorize user reviews and cluster them. Moreover, this tool can also prioritize the clusters of review. Grano et al. [50] analyze the available information in user reviews and indicate what type of user feedback can be actually adopted for testing apps. Palomba et al. develop `CRISTAL` [33] to trace informative crowd reviews onto source code changes, and use this relations to analyze the impact of crowd reviews on the development process. Ciurumelea et al. [51] analyze the reviews and classify them according to the taxonomy and recommend for a particular review what are the source code files that need to be modified to hander the issue. Genc-Nayebi and Abran [52] introduce the proposed solutions for mining online opinions in app store user reviews. In this systematic literature review, they also describe the challenges, unsolved problems, and new contributions to software requirements evolution. Sun et al. developed a novel system-`PADetective` [53] to detect promotional attacks from a large number of user reviewers. Xie et al. [54] proposed an effective approach to detect abused apps and related collusive attachers by analyzing the user reviews and corresponding raters in mobile app store. Chen et al. [55] proposed an approach to identify attackers of collusive promotion groups in an app store by exploiting the unusual ranking change patterns from user reviews.

The above-mentioned previous studies focus on analyze, classify, and extract the useful information from user reviews. In our work, we utilize SURF [9] to extract user feedback to locate classes to be changed. However, we not only analyze and cluster the user feedback from user reviews, but also build a link between user feedback and issue reports. Moreover, we utilize enriched clusters of user feedback as queries to recommend classes that should be changed for satisfying users' needs. This is a first work to use issue reports for enriching user feedback clusters in order to locate classes to be changed in mobile apps.

9 Conclusion

In this paper, we develop a new approach to locate source code classes that should be changed for each cluster of user feedback in

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

order to overcome the challenge when using the previous method named CHANGEADVISOR. In the first phase of our approach, we extract the user feedback from user reviews and cluster them in two categories *Discovery* and *Feature Request*; in the second phase, we build a link between a feedback cluster and the issue reports by computing the textual similarity between them. Next we utilize the issue reports to enrich the cluster of user feedback, and treat the enriched version as a query to perform information retrieval-based change request localization. In detail, we propose a weight selection-based cosine similarity metric to compute the similarity between the enriched cluster of user feedback and the source code. Finally, our approach can return a ranked list of classes that should be changed for each cluster of user feedback.

In order to demonstrate whether our approach performs better than CHANGEADVISOR, we execute the proposed approach and the baseline method CHANGEADVISOR on 31,597 reviews and 3,272 issue reports of 10 open source mobile apps in GitHub. The experimental results show that our approach can locate more source code classes related to change requests for more user feedback clusters produced by user reviews than CHANGEADVISOR due to higher Top-N and Recall values. We also compare the performance of our approach with two IR-based fault localization approaches-BLUiR and BLIA. The results show that our approach performs better than them. Moreover, we also conduct the empirical study for analyzing user reviews and issue reports and the results demonstrated that issue reports can help to improve the performance of change request localization but cannot replace user reviews to conduct this task for mobile apps.

In the future, we plan to explore a better way to select the user feedback related to real faults and feature requests reported in issue reports. Moreover, we are interested in developing a new method to locate source code classes to be changed for each user feedback entry rather than a cluster of user feedback.

Replication Package. We have publicly shared our dataset at: https://github.com/ReviewBugLocalization/ReviewBugLocalization. We will also publicly released Where2Change source code at the same GitHub repository when the work is published.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their quality reviews and suggestions.

REFERENCES

[1] T. Zhang, J. Chen, X. Luo, and T. Li, "Bug reports for desktop software and mobile apps in github: What's the difference?" *IEEE Software*, vol. 36, no. 1, pp. 63–71, 2019.

[2] N. Shahmehri, M. Kamkar, and P. Fritzson, "Semi-automatic bug localization in software maintenance," in *Proceedings of the International Conference on Software Maintenance 1990*, 1990, pp. 30–36.

[3] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen, "A topic-based approach for narrowing the search space of buggy files from a bug report," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 263–272.

[4] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 14–24.

[5] R. Saha, M. Lease, S. Khurshid, and D. Perry, "Improving bug localization using structured information retrieval," in *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering*, 2013, pp. 345–355.

[6] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *Proceedings of the 22nd Asia-Pacific Software Engineering Conference*, 2015, pp. 190–197.

[7] 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.

[8] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De 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.

[9] 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 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 499–510.

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2013, pp. 3111–3119.

[11] D. Binkley and D. Lawrie, "Information retrieval applications in software development," *Encyclopedia of Software Engineering*, 2010.

[12] S. Lukins, N. Kraft, and L. Etzkorn, "Source code retrieval for bug localization using latent dirichlet allocation," in *Proceedings of 15th Working Conference on Reverse Engineering*, 2008, pp. 155–164.

[13] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 43–52.

[14] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? a two-phase recommendation model," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1597–1610, 2013.

[15] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo, "Buglocalizer: Integrated tool support for bug localization," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 767–770.

[16] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 803–814.

[17] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *Proceedings of the 25th International Conference on Program Comprehension*, 2017, pp. 218–229.

[18] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 689–699.

[19] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 53–63.

[20] O. Chaparro, J. M. Florez, and A. Marcus, "Using observed behavior to reformulate queries during text retrieval-based bug localization," in *Proceedings of the 33rd International Conference on Software Maintenance and Evolution, to appear*, 2017.

[21] T.-D. B. Le, R. J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: Better together," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 579–590.

[22] S. Park, D. Lee, J. Choi, S. Ryu, Y. Kim, S. Kown, B. Kim, and G. G. Lee, "Hierarchical dirichlet process topic modeling for large number of answer types classification in open domain question answering," in *Proceedings of the 10th Asia Information Retrieval Societies Conference*, 2014, pp. 418–428.

[23] S. Panichella, A. D. 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 31st IEEE International Conference on Software Maintenance and Evolution*, 2015, pp. 281–290.

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

[25] Y. Kong, D. Wang, L. Shi, S. C. N. Hui, and W. C. W. Chu, "Dice similarity coefficients (dsc) of segmentation results using

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2956941, IEEE Transactions on Software Engineering

our approach and three predefined metrics on noisy synthetic datasets at different noise levels (online)," 2014. [Online]. Available: https://figshare.com/articles/_Dice_similarity_coefficients_DSC_of_segmentation_results_using_our_approach_and_three_predefined_metrics_on_noisy_synthetic_datasets_at_different_noise_levels_/968403

[26] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "Interpreting tf-idf term weights as making relevance decisions," *ACM Transactions on Information Systems*, vol. 26, no. 3, pp. 1–37, 2008.

[27] L. Wei, Y. Liu, and S.-C. Cheung, "Oasis: Prioritizing static analysis warnings for android apps based on app user reviews," in *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 672–682.

[28] L. De Vine, G. Zuccon, B. Koopman, L. Sitbon, and P. Bruza, "Medical semantic similarity with a neural language model," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 1819–1822.

[29] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 410–420.

[30] F. Palomba, P. Salza, S. Panichella, A. Ciurumelea, H. Gall, F. Ferrucci, and A. D. Lucia, "Recommending and localizing code changes for mobile apps based on user reviews: Online appendix," 2016. [Online]. Available: https://sites.google.com/site/changeadvisormobile/

[31] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Proceedings of the IEEE 22nd International Requirements Engineering Conference*, 2014, pp. 153–162.

[32] S. Tata and J. M. Patel, "Estimating the selectivity of tf-idf based cosine similarity predicates," *SIGMOD Record*, vol. 36, no. 2, pp. 7–12, 2007.

[33] F. Palomba, M. Linares-Vsquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution*, 2015, pp. 291–300.

[34] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, pp. 80–83, 1945.

[35] R. Ihaka and R. Gentleman, "R: A language for data analysis and graphics," *Journal of Computational and Graphical Statistics*, vol. 5, no. 3, pp. 299–314, 2012.

[36] M. Gomez, R. Rouvoy, B. Adams, and L. Seinturier, "Mining test repositories for automatic detection of ui performance regressions in android apps," in *Proceedings of the IEEE/ACM 13th Working Conference on Mining Software Repositories*, 2016, pp. 13–24.

[37] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[38] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 273–282.

[39] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[40] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Spectrum-based fault localization: Testing oracles are no longer mandatory," in *Proceedings of the 11th International Conference on Quality Software*, 2011, pp. 1–10.

[41] Lucia, D. Lo, and X. Xia, "Fusion fault localizers," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 127–138.

[42] G. Laghari, A. Murgia, and S. Demeyer, "Fine-tuning spectrum based fault localisation with frequent method item sets," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 274–285.

[43] A. Perez, R. Abreu, and A. van Deursen, "A test-suite diagnosability metric for spectrum-based fault localization approaches," in *Proceedings of the 39th International Conference on Software Engineering*, 2017, pp. 654–664.

[44] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 191–200.

[45] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams, "Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source android apps." in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, 2013, pp. 283–297.

[46] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 133–143.

[47] B. Zhou, I. Neamtiu, and R. Gupta, "A cross-platform analysis of bugs and bug-fixing in open source projects: desktop vs. android vs. ios," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, 2015, pp. No.7: 1–10.

[48] R. McCormick, "Google code is closing down because developers aren't using it," http://www.theverge.com/2015/3/13/8206903/google-code-is-closing-down-github-bitbucket.

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

[50] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall, "Exploring the integration of user feedback in automated testing of android applications," in *Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*, 2018, pp. 72–83.

[51] A. Ciurumelea, A. Schaufelbhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*, 2017, pp. 91–102.

[52] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *Journal of Systems and Software*, vol. 125, pp. 207–219, 2017.

[53] B. Sun, X. Luo, M. Akiyama, T. Watanabe, and T. Mori, "Characterizing promotional attacks in mobile app store," in *Proceedings of the 8th International Conference on Applications and Techniques in Information Security*, 2017, pp. 113–127.

[54] Z. Xie, S. Zhu, Q. Li, and W. Wang, "You can promote, but you can't hide: Large-scale abused app detection in mobile app stores," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 374–385.

[55] H. Chen, D. He, S. Zhu, and J. Yang, "Toward detecting collusive ranking manipulation attackers in mobile app markets," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 58–70.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.