Integrating User Reviews and Issue Reports of Mobile Apps for Change Requests Detection

Laila Al-Safoury, Akram Salah, Soha Makady
Department of Computer Science-Faculty of Computers and Artificial Intelligence, Cairo University, Cairo, Egypt

***Abstract*—There is abundance of mobile Apps released continuously on the App store, where developers are required to maintain these Apps to attain user satisfaction. Developers should consider all user feedback, as they are important resources for planning of next App's release. In order to consider user feedback, many platforms host mobile Apps and allow users to submit their opinions, such as: Google Play App store and Github Open-Source Development platform. The automated consolidation of user feedback from such platforms, and transforming it into a list of change requests would result in satisfying users across different platforms, and their analysis helps developer to reduce cost of time and effort to plan for the new release of the mobile App. In this paper, a framework is proposed which integrates user feedback from different sources and analyzes them using a state-of-art user reviews analysis tool to obtain a list of change requests, such list is further examined for similarity to remove duplicates and prioritize the identified change requests. A prototype is designed to implement the proposed framework and applied to AntennaPod. Consequently, the framework experimentation results show that reviews and issue reports can be analyzed almost equally despite the difference of text's nature.**

***Keywords*—User review; feedback analysis; mobile app maintenance; text similarity**

I. Introduction

The mobile App stores play an important role in distributing software products from different domains. In 2022 according to Statista website[1], Google Play store offers 3.3 million Apps for Android, while App store roughly includes 2.11 million Apps for iOS. The number of Apps increases over the years as they are widely discovered, purchased, and updated through the mobile App stores (*e.g.,* Google Play store and App store). Recently, researchers have studied the effect of App stores on software engineering practices [1, 2], while others have analyzed the benefit of using user reviews for software engineering [3, 4].

One of the most essential resources for the requirements elicitation activity is user reviews [5], which is offered by App stores allowing users to evaluate the downloaded Apps and to express their opinions [6]. App reviews are textual feedback associated with a star rating that indicates user satisfaction from one to five, where one is the lowest rate and five is the highest rate. However, the analysis of user reviews manually for extracting user needs is a challenging and time-consuming task [7]. As stated by Pagano et al. [8] 23 reviews per day are submitted in non-popular mobile Apps approximately and on average 4,275 reviews per day are received in popular Apps, such as Facebook. Besides, the feedback is usually written as unstructured text which cannot be parsed and analyzed easily.

Automated approaches are required to handle the consolidation of large amount of reviews and to perform review analysis tasks, such as: classifying feedback into maintenance tasks [9] or classifying feedback based on predefined topics [10] or based on user intention [11]. For example, reviews had been classified by Sorbo et al. into four intention categories based on user's intention while expressing their opinion such as information giving (opinion), information seeking (question), feature request (improvement or new feature), and problem discovery (bug report) [11].

In a recent review analysis survey [4], it had been raised that App store reviews can be integrated with other feedbacks available for developers to attain users' needs from more than one source, such as: Github [12, 13] and tweets [14, 15]. Since a mobile App can be hosted across platforms, such integration would demand a lot of manual effort from the App developer a lot of time and effort from App developer. The manual effort entails filtering manually such a large number of users' feedbacks from more than one platform to produce a list of change request which might still include duplicate change requests from different platforms.

In this paper, Google Play App store[2] reviews and Github[3] issue reports are integrated. GitHub is a leading open-source software development platform worldwide, it has more than one million android open-source repositories, four million issue reports, and 69K contributors in 2022[4]. Accordingly, Github is a good candidate as an additional source for integration since there are a reasonable number of mobile Apps exists on both platforms. The proposed framework aims at combining Google Play reviews and Github issue reports for a certain App's version release to obtain a list of change requests which includes bugs and feature requests submitted by users from both platforms and to group similar change requests for producing a list of unique change request. This list guides App's developers in planning for next release to meet further users' satisfaction, while eliminating the manual needed developer effort to analyze each platform's feedbacks separately, and the possible error proneness of such process.

This paper is organized as follows: Section II presents a brief background overview and motivating example which

[1] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] https://play.google.com/

[3] https://github.com/

[4] https://github.com/search?q=Android

demonstrate the importance of the proposed framework. Section III, discusses the related work of issue reports and user reviews integration and compares between the previous work and the approach proposed. Section IV introduces the proposed framework used for integration users' feedback to obtain a unique list of informative user feedback of two intentional category types. Section V describes a case study for applying the framework's phases using a dataset and it also discusses the evaluation metrics used for evaluating the framework's results. Section VI presents the results of the case study and its discussion. Finally, Section VII concludes the study and explains the future avenues.

II. Background and Motivation Example

In this section, the Google Play store's reviews and Github's issue reports are introduced to illustrate their attributes, also the importance of users' feedbacks integration across different platforms is explained further.

Fig. 1. Example of Google app review.

The App reviews are considered as one of the main framework inputs, therefore, it is important to understand the metadata associated per user review. As illustrated in Fig. 1, an example is provided by Google Play App store where a user named "Jon Dean" submitted a review on 12 May 2022, rated the App by "4 stars", and stated a bug experienced while using Antenna pod App version 2.5.2 (i.e., a podcast manager and player App). As shown in Fig. 1, each review has a date, user name, title, body, rate, number of likes, and a reply.

Fig. 2. Example of Github issue report.

In Github platform, for each App's release there are a set of issues submitted by users to be received by App developers to inform them whether there is a bug or a feature request to be considered in the following release and sometimes users might ask a question, when the issue is executed, it became a closed issue else it remains as an open issue. In Fig. 2, a user named "PersimmomKnob" submitted issue #4210 in Github platform, the user expressed a desired feature request regard sorting subscription feature, and it was labeled as feature request (not all issues are labeled). As shown in Fig. 2, each issue has a date, an id., open or closed issue label, user name, title, body, label (type of issue), number of likes, comments, and Assignees (i.e., members who works on this issue to be solved).

TABLE I. AntennaPod Mobile App Issue Reports and Review Examples

Feedback Type	Description
Review	Downloads don't show in the downloads folder. *Podcasts dont seem to download automatically.* I like the design though...
Issue Report	When I add the Feed http://podcast.hr2.de/derTag/podcast.xml it shows all files/episodes as 0 Byte. *They are not automatically downloaded.* I can download manually and listen to every episode. Every time the feed updates the first entry vanishes (even when downloaded) until everything is gone. Recent version (0.9.8.0) from play store on Android 4.1.1.
Issue Report	When the automatic download updates the feeds and a new episode is found, that *episode is not downloaded immediately.* Instead AntennaPod will wait for your custom set update interval (2h in my case). Leaving the house with an undownloaded episode is a bit annoying, easpecially when the download would have only taken a few seconds on your wifi. I did not test if that issue appears when setting a high update interval, but if it does one would receive all new episodes only after a high delay. My personal fix is to regularly check for new podcasts and hitting the update button manully, but I would like AntennaPod to be purely push- and not pull-style.

In Table I, an example of AntennaPod mobile App's version 0.9.8.0 reviews from Google Play[5] and issue reports from Github[6] where users express the same problem regard not being able to download their podcast automatically (i.e., underlined sentences). In contrast to reviews, the issue reports are more detailed and all sentences are focused in describing one problem rather than stating more than one information. The problem is expressed using different keywords, such as "download automatically" is similar to "download immediately" and "Podcasts" is similar to "Episode" as a podcast consists a set of episodes. Accordingly, the automation of integrating reviews and issue reports prevents the challenges of manual integration, such as: time-consuming, error prone and redundancy. Besides, the automation is needed to (i) handle the different usage of phrases which is challenging automatically to group all such reviews and issue reports, which express a bug or a feature request, into one list, and (ii) consider the rate at which the same problem is reported from different platform as a possible priority/importance indicator for that problem.

III. Related Work

A few approaches address integrating user reviews and other resource and most of those researches aims at using one source of data to enrich the other for a specific purpose which fulfill their proposed approach's purpose. Such approaches are discussed below in order to explain their aims and the differences between the proposed framework and their work.

Zhang et al. [12] proposes an approach for tagging the unlabeled Github issue reports using labeled issue reports and user reviews. Github issue report have an associated label, which define the type of this issue report (i.e. bug or feature request), and this label are optionally added which leads into some unlabeled issue reports. On the other hand, user reviews are classified into bug and feature request using a tool then both labeled issue reports and classified reviews are used as input for calculating text similarity between unlabeled issue

[5] https://play.google.com/store/apps/details?id=de.danoeh.antennapod
[6] https://github.com/AntennaPod/AntennaPod/issues

reports and the input. On the contrary, the proposed framework focuses on integrating both reviews and issue reports for filling all issue reports' labels, which helps the developer to filter issue reports according to its label without dealing with redundant issue reports.

Zhang et al. [13] introduces an approach for grouping user reviews as a cluster which addresses bugs and feature requests of a certain mobile App, along with relevant issue reports for enriching the cluster of user feedback. This approach utilizes user feedback clusters for linking each cluster to set of relevant code classes (i.e., affected classes if this group of bugs or feature requests is implemented). The classes are obtained through calculating the text similarity between classes' names and text of reviews and issue reports within each cluster producing a ranked list of classes recommended for each cluster. The paper integrated reviews and issue reports for enriching the reviews' cluster for recommending more accurate classes, which guides the developer on having a list of classes recommended per each cluster, unlike the proposed framework, if an issue report is not similar to any review, then it will be excluded, also if there are a number of similar issue reports that are not similar to any review, they will be excluded.

Yadav et al. [15] proposes a framework to analyze the users' feedback, from Google Play store and Twitter, by embedding their semantics. The framework classifies the feedback into two classes of bug reports and feature requests then it investigates whether the approach can identify the similar feedbacks. This paper is an example for integration using Twitter, where user can express their opinions regard mobile Apps through posting a tweet with a limited 140 character. On the contrary, the proposed framework uses different type of information which Github issue report where users are able to write in more details as shown in Table I and they are not limited to express their opinions briefly using limited short text.

IV. Proposed Framework

In this framework, as shown in Fig. 3, the Google Play App store is used as a source of reviews while Github is used as a source of issue reports where both reviews and issue reports form feedback inputs. There are three main phases which process the input to produce a unique change request list, which are explained as follows:

1) Feedback pre-processing: Each feedback has a text which is divided into sentences and pre-processed by applying Snowball Stemming [16].

2) Feedback classification: Each sentence is parsed using Stanford Typed Dependencies (STD) [17] then mapped to a set of 246 natural language processing (NLP) heuristics[7] to obtain the sentence's structural patterns associated with one of user's categories. Additionally, each sentence is analyzed to get sentiment annotator using Stanford CoreNLP [18] for improving the accuracy of intent classification. Both structural patterns and sentiment features are used as an input for a pre-trained Machine Learning [19, 20]. The output of this phase is a list of change requests classified as bug or feature request.

3) Text similarity calculation: According to previous work [12], cosine similarity measure and BM2F model were used for calculating textual similarity between issue reports, while Jaccard similarity is used for calculating textual similarity between issue reports and reviews because it performs better when data is sparse. The output of this phase is a unique change requests list where similar change requests are grouped.

For applying phase 1 and phase 2, there are a set **of** publicly-available App review mining tools according to the recently published survey [4]**.** SURF (Summarizer of User Reviews Feedback) was used in other research papers and they showed promising results [12, 13]**.** For input processing, SURF is used as a tool which summaries the feedback written by users to assist developers in figuring out user needs and dealing with an abundance number of reviews. The tool works as follows: (i) classifies user's intention determining the type maintenance task required to fulfill user's needs, (ii) gathers sentences of the same topic, (iii) receives user feedback in XML format and also generates the output in XML format which allows integrating them in third party frameworks, and (iv) produces a visualized report for browsing the summaries easily.

Moreover, SURF is constructed based on User Reviews Model (URM) [21, 19] which categorizes each sentence contained in App reviews into two dimensions: (i) the user intention: It is user's goals when writing a review (such as: Information giving, Information seeking, Feature request, Problem discovery, or other), and (ii) the review topics: It finds the most relevant topic(s) belong to this review (such as: App, GUI, Contents, Pricing, Feature or Functionality, Improvement, Updates/Versions, Resources, Security, Download, Model, or Company).

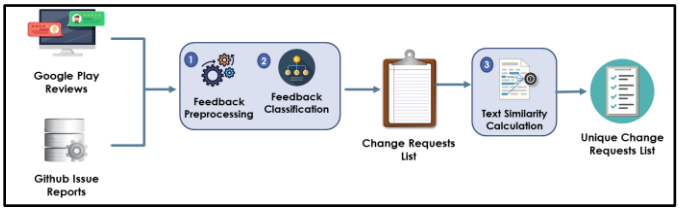

Fig. 3. Proposed framework phases.

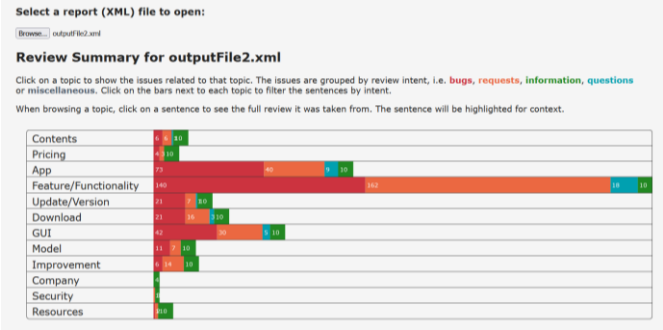

Fig. 4. Example of output report.

[7] https://www.ifi.uzh.ch/seal/people/panichella/Appendix.pdf

Not all of the change request sentences will appear on the output report in case it did not match an intention type or one of the predefined topics. Moreover, a sentence can be dismissed from the output report due to the sentence scoring which is calculated to select 2/3 of total sentences having the highest scores will be included on the report. The rewarding factors of sentence scoring are: (i) sentences classified as feature requests or bug reports, (ii) sentences related to specific topics, (iii) longer sentences, and (iv) sentences containing frequently discussed features. Afterwards, the output report is obtained in XML format which can be easily browsed through visualized utility supported in the SURF replication package[8]. As illustrated in Fig. 4, the topics are listed in rows where each topic has four types of intention categories colored differently, each type has number of sentences extracted from xml input file, and the sentences appear when you click on the number.

The SURF tool produces XML file report which includes each feedback's sentence and its user intention category. In this paper, the sentences are considered as change request, if it is classified as bug or feature request, extracted to form a list of change requests, then the text similarity is applied for reducing the list size by grouping similar sentences into one change request.

V. Framework Casestudy and Evaluation

A case study is applied on the proposed framework using an open-source dataset for implementing and evaluating the phases of the framework. In this section, the details of such case study (section 5.1) and evaluation (section 5.2) are presented to address two research questions: (RQ1) 'Can the framework classify the users' reviews and issues to obtain a list of change requests?'; (RQ2) 'To what extend the framework is capable of classifying all the change requests into bug or feature request accurately?'; (RQ3) 'Does the framework similarity percentage is reasonable and worth to be a part of the proposed framework?'.

A. Experimental Case Study

In phase 1, the dataset used for the experimentation had been publicly shared by [12], it includes top 17 popular android open-source mobile Apps which are available in both Google Play store and Github. The user reviews and the issue reports were collected for each App during a specific period which is different from App to another, where the dataset includes 20,135 issue reports and 43,649 reviews. The framework is applied on "AntennaPod" App which includes 1108 Github issue reports and 2082 Google Play reviews, such data was collected from 3 August 2012 till 9 January 2018. Particularly, App version 0.9.8.0 was selected as input which is gathered from 19 October 2013 to 26 February 2014 (i.e. before version 0.9.8.1 publishing date on 27 Feb 2014) and it has 43 issue reports and 50 reviews.

Additionally, the date attribute in issue xml file had been added to the dataset for obtaining the version of App in which the issue was submitted.

```
<review id="1901">
    <app_version/>
    <user>Drew Miller</user>
    <date>October 20, 2013</date>
    <star_rating>5</star_rating>
    <review_title/>
    <review_text>Best android podcast client</review_text>
</review>
```

```
<Manager Number="211">
    <labels>
        <label>wontfix</label>
    </labels>
    <bugID>Issue #302</bugID>
    <url>
    https://github.com/AntennaPod/AntennaPod/issues/302
    </url>
    <description>
    Downloading of audio file failed if audio URL contains
    spaces. Streaming is fine.
    </description>
    <title>Failed Download</title>
    <date>Oct 26, 2013</date>
</Manager>
```

Fig. 5. Example of dataset review and issue XML files.

As shown in Fig. 5, the dataset stores reviews and issue in form of XML format, where a set of attributes are defined for each review and issue report. In reviews, the xml file contains user name, date, user rating, review title, and review text (body), on the other hand, the issue reports xml file contains label, issue id., issue URL, description (body), title, and date.

In phase 2, the SURF tool requires the input in form of XML file, where attributes should be written in a specific format (as shown in Fig. 6). Accordingly, the attributes of the dataset xml files has to be converted to fit SURF tool input prerequisite specification in attributes naming (e.g. the review title attribute is named as <title> in issue report xml file, it has to be changed into <review_title> for all stored issue reports to fit SURF input file). For all reviews and Github issue reports of a version App, the naming of attributes are changed to match SURF input file, all non-English feedback are removed because the tool only process English text, then both feedbacks are gathered in one file for to be processed by the tool.

```
<reviews>
  <review>
      <date>2014-12-04</date>
      <star_rating>1</star_rating>
      <user>Jmbrowne</user>
      <app_version>2.0.1</app_version>
      <review_title>Painfully slow</review_title>
      <review_text>
          This puzzle would be enjoyable if it was not
          so painfully slow.
      </review_text>
  </review>
  . . .
</reviews>
```

Fig. 6. SURF tool input XML file format [11].

TABLE II. SURF Classified Sentences for Dataset

Source of Classified Sentence	Number of Sentences
Total Sentences (S)	70
Feature Requests in Reviews (R_F)	9 (7)
Bugs in Reviews (R_B)	4 (4)
Feature Requests in Issues (I_F)	19 (14)
Bugs in Issues (I_B)	10 (9)

[8] https://github.com/panichella/SURF/tree/SURF-v.1.0

In Table II, when SURF tool is used to process the input only 70 sentences were classified while the rest did not appear in the output XML report. The SURF tool does not provide a sentence in output report unless it is able to classify its intention type (i.e., according to "predefined patterns" matching) and also if it is able to classify its topic (i.e., based on "topic words dictionary" matching). Additionally, sentence scoring results is reduced by 30% of total sentences according to four factors which are previously mentioned. As shown in Table II, the number of sentences per each type is mentioned and the number of correctly classified types is written between brackets. For example, the tool classified nine review's sentences as feature requests and compared to the manually labeled 19 sentences mentioned in Table IV only 7 sentences were classified correctly. According to the results mentioned in Table II, RQ1 is answered:

Answer to RQ1: the framework is able to classify both reviews and issues using the same approach explained previously in Section IV.

TABLE III. CHANGE REQUESTS SIMILARITY FEATURES

Change Request	Issue	Review
Login Dark Theme	2	0
Auto Delete	1	3
Auto Download	2	1
Pause Frequently	0	2
Stop Playing	1	2
fix widget	0	3
Download Scheduler	0	2
Less Cache	0	2
skip back	1	1
Total	7	16

In phase 3, the list of change requests is studied manually to analyze the repeated requests, the results are depicted on Table III which lists the name of the addressed feature, and the number of issues or reviews mentioned this feature as a bug or a feature request having similar change request. There are nine change requests which had been expressed by more than one user whether in Github or Google Play store platforms.

B. Case Study Evaluation and Results

After executing the framework, the dataset's sentences are manually classified and labeled to be used in evaluating the accuracy of the framework's output. Thus, the input is divided into a set of sentences to be manually classified for obtaining the user intention type for each sentence to be used as a ground truth. For splitting the input into sentences, a natural language toolkit (NLTK) has been utilized in python program which offers NLP methodology implementation.

As shown in Table IV, the total number of sentences in the dataset are 252, part of these sentences belongs to issue report extracted from Github platform (135 sentences) and the other sentences belong to reviews extracted from Google Play store (117 sentences). The manually labeled sentences show that 16% of reviews' sentences are feature requests while 23% of reviews' sentences are bugs, on the other hand, 17% of issue report's sentences are feature requests while 39% of issue report's sentences. Those percentages show that, in such dataset, only half of the total sentences are useful for the developers as they state the changes requested by the App's users.

For evaluating accuracy of the change request list's output sets (i.e. list of sentences), both ground truth's four sets (MI_B, MI_F, MR_B, and MR_F) shown in Table IV and SURF tool's produced sets (I_B, I_F, R_B, and R_F) shown in Table II are used to measure recall and precision which are a well-established metrics reported in Equations 1 and 2.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

TABLE IV. MANUALLY CLASSIFIED SENTENCES FOR DATASET

Source of Sentence	Number of Sentences
Total Sentences (MS)	252
Google Play Reviews (MR)	117
Github Issue Reports (MI)	135
Feature Requests in Reviews (MR_F)	19 (16%)
Bugs in Reviews (MR_B)	27 (23%)
Feature Requests in Issues (MI_F)	23 (17%)
Bugs in Issues (MI_B)	53 (39%)

In particular, TP represents the number of true positives reviews/issue reports classified correctly; False Positive (FP) is the number of reviews/issue reports classified incorrectly by the approach; and False Negative (FN) is the number of correct reviews/issue reports (i.e., manually classified as bug or feature request) that are not classified by the approach.

For applying accuracy measurement, three variables are required to calculate precision and recall as in equation (1) and (2), shown in Table V. Also, F-measure is used as it combines both recall and precision by calculating the harmonic mean of precision and recall to obtain the overall accuracy of the proposed approach, known as the F-measure in Equation 3:

$$\text{F-measure} = 2 * \frac{\text{Precision} * \text{recall}}{\text{Precision} + \text{recall}} \quad (3)$$

After calculating accuracy measurements, it is noticed that precision of bug is better than feature request (i.e. the ability of predicting change requests correctly), as shown in Table VI and Table VII, for both inputs issue reports or reviews. On the other hand, the recall of feature request is better than bug (i.e., the ability of to find all manually labeled relevant cases) in both issue reports and reviews. The SURF tool is able to predict the intentional type as they follow predefined textual patterns, such as: bug patterns "[something] is missing" and "[someone] has an issue", but these patterns are not sufficient for covering all sentences expressing bugs and feature requests to be matched with the dataset. According to the results mentioned in Table VI and Table VII, RQ2 is answered:

TABLE V. THE CALCULATION OF ACCURACY MEASUREMENTS' VARIABLES

Accuracy Measure	TP	FN	FP
Bug Issues	Numbers of issues sentences classified as "bug" and manually labeled as "bug".	Numbers of issues sentences manually labeled as "bug" but not classified as "bug".	Numbers of issues sentences not manually labeled as "bug" but classified as "bug".
Feature Request Issues	Numbers of issues sentences classified as "feature request" and manually labeled as "feature request".	Numbers of issues sentences manually labeled as "feature request" but not classified as "feature request".	Numbers of issues sentences not manually labeled as "feature request" but classified as "feature request".
Bug Reviews	Numbers of reviews sentences classified as "bug" and manually labeled as "bug".	Numbers of reviews sentences manually labeled as "bug" but not classified as "bug".	Numbers of reviews sentences not manually labeled as "bug" but classified as "bug".
Feature Request Reviews	Numbers of reviews sentences classified as "feature request" and manually labeled as "feature request".	Numbers of reviews sentences manually labeled as "feature request" but not classified as "feature request".	Numbers of reviews sentences not manually labeled as "feature request" but classified as "feature request".

TABLE VI. CLASSIFICATION EVALUATION OF SURF RESULTS FOR ISSUE REPORTS

Change Request Type	TP	FN	FP	Precision	Recall	F-measure
Bug	9	44	1	90%	17%	29%
Feature Request	14	9	5	74%	61%	67%

TABLE VII. CLASSIFICATION EVALUATION OF SURF RESULTS FOR REVIEWS

Change Request Type	TP	FN	FP	Precision	Recall	F-measure
Bug	4	23	0	100%	15%	26%
Feature Request	7	12	2	78%	37%	50%

Answer to RQ2: The experiment's precision was better than recall which affect negatively in F-measure, which means that the tool is able to classify most of the sentences correctly but it is not able to detect all relevant sentences in the used dataset. A few of modifications are exerted in Section 6 for enhancing the results of the tool used in phase 3.

The output list is reduced using similarity calculation to reduce the total number of change requests (CR); thus, each sentence S_i text is compared to other sentences. If S_i is similar to one on more sentences, then they will be grouped as one change request, else nothing will change. The evaluation is performed by calculating the percentage of reduction using CR which is the total number of change requests' sentences before reduction (i.e., before grouping) and unique change requests (UCR) which is the total number of unique change requests' sentences, shown in Equation 4.

$$\text{Reduction Percentage} = \frac{\#UCR * 100}{\#CR} \quad (4)$$

According to Table III and Equation 4, the total number of repeated change requests is 23 change requests (i.e., 7 issues and 16 reviews), the change requests list will be reduced from 252 into 238 (i.e., reduced by 6%). According to such results mentioned, RQ3 is answered:

Answer to RQ3: The framework's output of the similarity percentage is slightly reduced in phase 3 compared to the original change request list produced from phase 4, this observation is related to the nature of selected App's version dataset which needs to be investigated for other Apps' dataset.

VI. RESULTS ENHANCEMENT AND DISCUSSION

In this section, the impact of adding the title of the change request (*e.g.,* review or issue report) is investigated to the change request body as an input to phase 2. Then, the output of phase 2 is evaluated to check whether the added topic may improve the recall values. Additionally, the patterns of SURF tool were studied to examine their sufficiency for covering all types of sentences belongs to bug or feature request.

Another experimental trial has been performed for enhancing the results, which is adding the title of the change request into the beginning of the body. Because SURF tool only considers the review/issue report's body without including its title, i.e., it is a sentence-based classifier, although the title might contain valuable general information about the bug or feature request itself while the body includes extra explanatory information. Also, including the title might increase the fourth factor of sentence scoring which increase the feature discussed. In Table VIII, the evaluation results slightly changed only in review input, and it was a minor change where an extra bug was classified by SURF correctly (i.e., increasing recall and f-measure) and a feature request where misclassified (i.e., decreasing precision and f-measure).

As shown in Table IX, when the sentences are changed to follow the predefined SURF pattern, such in all examples except the third one (i.e., begin with the pattern by splitting the sentence), the SURF tool was able to classify the sentences correctly. Additionally, the position of the pattern affects the ability of classification, since the tool is not able to detect the pattern when it is embedded in the middle of a long sentence. It is noticed that papers [12, 13], explained in Section II, used SURF tool and both mentioned obtaining high overall accuracy of 91.36% and 95.64% respectively which are stated to be acceptable results. Although, this study uses the same dataset as paper [12] but the accuracy obtained was 77% as not all of the dataset was used, only one version of an App.

TABLE VIII. CLASSIFICATION EVALUATION OF SURF RESULTS FOR REVIEWS INPUT (INCLUDING TITLE)

Change Request Type	TP	FN	FP	Precision	Recall	F-measure
Bug	5	23	0	100%	19%	31%
Feature Request	7	12	2	70%	37%	48%

TABLE IX. ALTERING SENTENCES ACCORDING TO SURF PATTERNS

SURF Pattern	Original Sentence	Altered Sentence
"[something] can be fixed by [something]"	My personal fix is to regularly check for new podcasts and hitting the update button manully, but I would like AntennaPod to be purely push- and not pull-style.	This can be fixed by regularly checking for new podcasts and hitting the update button manully, but I would like AntennaPod to be purely push- and not pull-style.
"[something] should be fixed"	The widget needs fixing since it doesnt always work.	The widget should be fixed since it doesnt always work.
[someone] could/should add/provide/offer/integrate [something]	Congrats for your job in this App, its so amazing and for me almost perfect, I don't know if its hard to do or not, but if you could add the auto delete of the podcast when we finish it will be good.	Congrats for your job in this App, its so amazing and for me almost perfect, I don't know if its hard to do or not. if you could add the auto delete of the podcast when we finish it will be good. Thanks!
	Wish i could schedule when it downloads the podcast.	Wish i could add a schedule when it downloads the podcast.

VII. CONCLUSION AND FUTURE WORK

The users' contribution in expressing their opinions is considered as an important source for evolving mobile Apps and for better release planning. Mobile App can be distributed on many platforms where each platform allows users to submit their feedback for other users and App's developers to ask questions, state information, report a bug, or suggest a modification. Accordingly, the integration of more than one feedback source assists developer to satisfy large number of users which increase App's rating. Because of the large amount of feedback provided by users, an automatic tool for feedback analysis is required. In this paper, the proposed framework analyzed Google Play reviews and Github issues to filter them and to extract change requests out of the integrated feedback. After performing two trials, the accuracy results of reviews and issue reports showed slightly different values which proves that the nature of both textual feedbacks did not affect results after using the same tool. Despite the tool was able to classify the integrated inputs into bugs and feature requests, it was not able to extract all of them. Besides, the similarity can assist on reducing the number of change requests to decrease of the time needed for developer to check the list of change requests. In the future, we are interested in including more Apps feedback from dataset to be able to evaluate whether the results' insights are repeated using other versions or other Apps. Also, the similarity will be automated and examined in larger dataset versions and Apps to evaluate further the effect of reducing the list of change request. Finally, other analysis tools can be investigated for producing better results of the integrated users' feedback classification.

REFERENCES

[1] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra and M. Harman, "App store effects on software engineering practices," IEEE Transactions on Software Engineering, vol. 47, no. 2, pp. 300-319, 2019.

[2] A. A. Al-Subaihin, "Software Engineering in the Age of App Stores: Feature-Based Analyses to Guide Mobile Software Engineers," PhD diss., UCL (University College London), 2019.

[3] W. Martin, F. Sarro, Y. Jia, Y. Zhang and M. Harman, "A survey of app store analysis for software engineering," IEEE transactions on software engineering, vol. 43, no. 9, pp. 817-847, 2016.

[4] J. Dabrowski, E. Letier, A. Perini and A. Susi, "Analysing app reviews for software engineering: a systematic literature review.," Empirical Software Engineering, vol. 27, no. 2, pp. 1-63, 2022.

[5] E. A. AlOmar, W. Aljedaani, M. Tamjeed, M. W. Mkaouer and Y. N. El-Glaly, "Finding the needle in a haystack: On the automatic identification of accessibility user reviews.," in Proceedings of the 2021 CHI conference on human factors in computing systems, 2021.

[6] S. Lim, A. Henriksson and J. Zdravkov, "Data-driven requirements elicitation: A systematic literature review.," SN Computer Science, vol. 2, no. 1, pp. 1-35, 2021.

[7] Y. Zhang, J. Du, X. Ma, H. Wen and G. Fortino, "Aspect-based sentiment analysis for user reviews," Cognitive Computation, vol. 13, no. 5, pp. 1114-1127, 2021.

[8] P. Dennis and W. Maalej, "User feedback in the appstore: An empirical study," in 21st IEEE international requirements engineering conference, 2013.

[9] A. Al-Hawari, H. Najadat and R. Shatnaw, "Classification of application reviews into software maintenance tasks using data mining techniques," Software Quality Journal, vol. 29, no. 3, pp. 667-703, 2021.

[10] M. v. Vliet, E. C. Groen, F. Dalpiaz and S. Brinkk, "Identifying and classifying user requirements in online feedback via crowdsourcing," in International Working Conference on Requirements Engineering: Foundation for Software Quality, 2020.

[11] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio and G. Canfora, "SURF: summarizer of user reviews feedback," in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017.

[12] T. Zhang, H. Li, Z. Xu, J. Liu, R. Huang and Y. Shen, "Labelling issue reports in mobile apps," vol. 13, no. 6, pp. 528-542, 2019.

[13] T. Zhang, J. Chen, X. Zhan, X. Luo, D. Lo and H. Jiang, "Where2Change: Change Request Localization for App Reviews," IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2590-2616, 2019.

[14] P. R. Henao, J. Fischbach, D. Spies, F. Julian and A. Vogelsang, "Transfer Learning for Mining Feature Requests and Bug Reports from Tweets and App Store Reviews.," in 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW), 2021.

[15] A. Yadav, R. Sharma and F. H. Fard, "A semantic-based framework for analyzing app users' feedback," in IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020.

[16] M. Bounabi, K. El Moutaouakil and K. Satori, "A comparison of text classification methods using different stemming techniques.," International Journal of Computer Applications in Technology, vol. 60, no. 4, pp. 298-306, 2019.

[17] J. Kleenankandy and A. N. KA, "An enhanced Tree-LSTM architecture for sentence semantic modeling using typed dependencies.," Information Processing & Management, vol. 57, no. 6, p. 102362, 2020.

[18] C. D. Manning, M. Surdeanu, J. Bauer, J. . R. Finkel, S. Bethard and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, 2014.

[19] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora and H. C. Gall, "How can I improve my app?

Classifying user reviews for software," in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016.

[20] S. Panichella, A. . D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfor and H. C. Gall, "Ardoc: App reviews development oriented classifier," in Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, 2016.

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