What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes

Andrea Di Sorbo[1], Sebastiano Panichella[2], Carol V. Alexandru[2],
Junji Shimagaki[3], Corrado A. Visaggio[1], Gerardo Canfora[1], Harald C. Gall[2]
[1]University of Sannio, Department of Engineering, Italy
[2]University of Zurich, Department of Informatics, Switzerland
[3]Sony Mobile Communications, Japan
disorbo@unisannio.it, {panichella, alexandru}@ifi.uzh.ch,
Junji.Shimagaki@sonymobile.com,{visaggio,canfora}@unisannio.it, gall@ifi.uzh.ch

ABSTRACT

Mobile app developers constantly monitor feedback in user reviews with the goal of improving their mobile apps and better meeting user expectations. Thus, automated approaches have been proposed in literature with the aim of reducing the effort required for analyzing feedback contained in user reviews via automatic classification/prioritization according to specific topics. In this paper, we introduce ***SURF*** (**S**ummarizer of **U**ser **R**eviews **F**eedback), a novel approach to condense the enormous amount of information that developers of popular apps have to manage due to user feedback received on a daily basis. SURF relies on a conceptual model for capturing user needs useful for developers performing maintenance and evolution tasks. Then it uses sophisticated summarisation techniques for summarizing thousands of reviews and generating an interactive, structured and condensed agenda of recommended software changes. We performed an end-to-end evaluation of SURF on user reviews of 17 mobile apps (5 of them developed by Sony Mobile), involving 23 developers and researchers in total. Results demonstrate high accuracy of SURF in summarizing reviews and the usefulness of the recommended changes. In evaluating our approach we found that SURF helps developers in better understanding user needs, substantially reducing the time required by developers compared to manually analyzing user (change) requests and planning future software changes.

CCS Concepts

•**Information systems** → *Summarization;* •**Software and its engineering** → *Software maintenance tools;*

Keywords

Mobile Application; User Feedback; Text Summarization

1. INTRODUCTION

User feedback plays a paramount role in the development and maintenance of mobile applications. The experience an end-user has with the app is a key concern when creating and maintaining a successful product. Consequently, developer teams are interested in exploiting opinions and feedback of end-users during the evolution of their software [21, 45].

On distribution platforms such as the Apple Store and Google Play, users can leave plain text reviews for the apps they install and use. These reviews may not only contain simple sentiments (*e.g.*, "Great app!"), but they can provide valuable information regarding several topics that are highly relevant to the development and maintenance of the app [19, 29]. In particular, user reviews may include (i) bugs or issues that need to be fixed [29] (ii) summaries of the user experience with certain features [12] (iii) requests for enhancements [17], and (iv) ideas for new features [10, 29]. However, existing app distribution platforms provide limited support for developers to systematically filter, aggregate and classify user feedback to derive requirements [1]. Moreover, manually reading each user review to gather useful feedback is not feasible considering that popular apps receive hundreds of reviews every day [22, 29].

For this reason automated approaches have been proposed in literature with the aim of reducing the effort required to analyze feedback contained in user reviews [5, 9–13, 17, 18, 23, 28, 30, 46]. However, most of them only perform a classification (or prioritization) of user reviews according to specific topics (*e.g.*, bugs, enhancements, etc.) [10, 12, 13, 17, 18, 23, 28, 33], without reducing the amount of reviews and information developers have to deal with, which is very large for popular apps. To the best of our knowledge, no approach is able to, at the same time, (i) determine for a large number of reviews the specific topic discussed in the review (*e.g.*, UI improvements, security/licensing issues, etc.), (ii) identify the maintenance task to perform for addressing the request stated in the review (*e.g.*, bug fixing, feature enhancement, etc.), and (iii) present such information in the form of a *condensed, interactive and structured agenda of recommended software changes*, which is actionable for developers.

We argue that combining topic extraction, intention classification and the ability to synthesize well defined maintenance tasks regarding specific aspects of an app will concretely help developers in planning further software changes and meeting market requirements.

Paper contribution. The contributions of our paper are summarized as follows:

- we first define ***URM*** (**U**ser **R**eviews **M**odel), a two-level classification model which takes into account both the *users' intentions* (when giving feedback to developers) and the *review topic*, which is the specific aspect of the app covered by the review;

FSE'16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2950299

- we introduce **SURF** (**S**ummarizer of **U**ser **R**eview **F**eedback), a novel approach, built on top of *URM*, to automatically generate summaries of users feedback with the aim of helping developers better understanding user needs. *SURF* exploits summarization techniques to summarize thousands of user reviews and generate an *interactive*, *structured* and *condensed agenda of recommended software changes*.
- we conduct an empirical study involving 12 developers/engineers from companies in Switzerland, Italy and the Netherlands and 11 developers/engineers from the development team of Sony Mobile in Japan, to investigate the practical usefulness of summaries generated by *SURF* in the developers' "working context";
- we make publicly available in a replication package[1][2] with (i) material and working data sets of our study, (ii) complete results of the survey, (iii) raw data (for replication purposes and to support future studies), and (iv) the prototypical implementation of *SURF*.

2. APPROACH

In this section, we detail **URM** and **SURF**, the model and approach we use for synthetizing app reviews.

2.1 The User Reviews Model

URM aims to model *informative paragraphs* [33] contained in app reviews from a software maintenance and evolution perspective along two orthogonal dimensions:

1. **User intention**: modeling the user's goals when writing a review (*e.g.*, *Is the reviewer requesting a new feature or reporting a bug?*).
2. **Review topic**: capturing the specific topic covered by the review (*e.g.*, *Is the writer discussing the user interface or a pricing issue?*).

Previous app review content classification attempts [13, 20, 29] have seen limited adoption by researchers because they consider just one of these *two* complimentary dimensions. In recent work [33] we introduced a taxonomy that classifies app review content into categories relevant for software maintenance and evolution, namely *feature request, problem discovery, information seeking, information giving* and *other*. However, this one dimensional classification results in an insufficient leverage of the available review information, because, for example, having a huge amount of reviews classified as *feature requests* is of limited use to developers trying to distill actionable tasks.

For this reason, *URM* aims to enrich and complement the preliminarily classification described in Table 1 by identifying the specific topic of each review. In the rest of the paper we refer to the aspects of an app targeted by a review as the *review topics*. To determine a complete set of review topics, we analyzed 1390 reviews present in the dataset of our previous work [33]. Thus, we split the dataset in two parts: (i) a training set $T_{training}$ containing 438 randomly selected reviews and (ii) a test set T_{test} containing the remaining 952 reviews. Then, starting from an empty list of topics L_{topics}, two authors of the paper performed a manual labeling of each sentence contained in $T_{training}$ (we work at sentence level since different sentences in the same review can contain different kinds of feedback). During the labeling process, when a sentence did not match any of the defined topics, a new topic was added to the topic list L_{topics}, and the two validators went through all the previously labeled sentences to see whether it is more appropriate to tag some of them with the newly defined topic. Vice versa, if a review sentence matched an already defined topic in L_{topics}, the sentence was simply tagged with that topic. Since a review sentence can refer to more than one topic (a user can ask to improve the UI and in the same sentence request a bug fix related to stability of the app), some sentences were tagged with multiple labels. At the end of this process, a preliminary set of *33 topics* was defined[3]. Then, we merged (or clustered) together *review topics* that are semantically related, and that can generate redundancies and overlaps in the recommended software changes. After this process, we obtained a set of *12 topic clusters*, detailed in Table 2. However, a more accurate process for investigating/removing possible overlaps in the resulting clusters is part of our future agenda.

Table 1: Intention Categories Definition

Information Giving	Sentences that inform other users or developers about some aspect of the app.
Information Seeking	Sentences describing attempts to obtain information or help from other users or developers.
Feature Request	Sentences expressing ideas, suggestions or needs for enhancing the app.
Problem Discovery	Sentences reporting unexpected behavior or issues.
Other	Sentences not belonging to any of the previous categories.

Table 2: Topic Clusters

Cluster	Description
App	sentences related to the entire app, *e.g.*, generic crash reports, ratings, or general feedback
GUI	sentences related to the Graphical User Interface or the look and feel of the app
Contents	sentences related to the content of the app
Pricing	sentences related to app pricing
Feature or Functionality	sentences related to specific features or functionality of the app
Improvement	sentences related to explicit enhancement requests
Updates/ Versions	sentences related to specific versions or the update process of the app
Resources	sentences dealing with device resources such as battery consumption, storage, etc.
Security	sentences related to the security of the app or to personal data privacy
Download	sentences containing feedback about the app download
Model	sentences reporting feedback about specific devices or OS versions
Company	sentences containing feedback related to the company/team which develops the app

URM assigns to each review sentence one of the intention categories (detailed in Table 1) and one or more app topic clusters (defined in Table 2). For instance, the example sentence *"I love this app but it crashes my whole iPad and it has to restart itself."* will be classified as a *Problem Discovery* involving the *App* and *Model* topics, since it reveals that the user is experiencing an app crash while she is using a specific device (*i.e.*, iPad).

2.2 The SURF Approach

`SURF` automatically (i) extracts the topics treated in reviews, (ii) classifies the intention of the writers, to suggest the specific kinds of maintenance tasks developers have to accomplish, and (iii) groups together sentences covering the same topic. Figure 1 depicts an overview of the SURF process activities, which we detail in the following sections.

[1] http://www.ifi.uzh.ch/seal/people/panichella/tools/SURF.html
[2] http://dx.doi.org/10.5281/zenodo.47323
[3] http://www.ifi.uzh.ch/seal/people/panichella/tools/SURF/APPENDIX.pdf

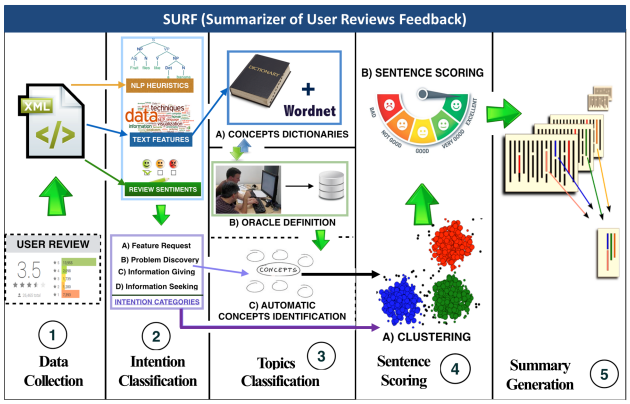

Figure 1: The SURF process

2.2.1 Data Collection

The most common app distribution platforms are heterogeneous with regard to how reviews are posted, and they collect data in different kinds of data structures. For this reason we designed a *data exchange format* for collecting information that is potentially useful to developers and software analysts for maintaining and evolving apps. Specifically, since XML is particularly well suited for data where field lengths are unknown and unpredictable and where field contents are mainly text, we developed an XML *data exchange schema* for representing review data from multiple sources. For each user review, it stores: (i) the date on which the review has been posted, (ii) the (star) rating given by the reviewer, (iii) the handle of the user who posted the review, (iv) the version of the app, (v) the title of the review, and (vi) the review text itself. The *output of the Data Collection* phase of SURF is a homogenized collection of reviews for each app in a well defined XML format.

2.2.2 Intention Classification

`SURF` employs an approach we previously defined to automatically mine reviewer *intentions* [33]. Such an `Intent Classifier` combines Natural Language Parsing (NLP), Sentiment Analysis (SA) and Text Analysis (TA) techniques through a Machine Learning (ML) algorithm for detecting sentences in user reviews that are important from a maintenance perspective (see Table 1). To help other researchers replicate the analysis performed in this step of SURF we make available online[4] the Java version of our `Intent Classifier`, already used by some developers in Germany[5].

2.2.3 Topics Classification

As discussed in Section 2.2, one of the aims of `SURF` is to group sentences involving similar *review topics* (see Table 2). In this step, `SURF` automatically associates one or more concepts illustrated in Table 2 to each review sentence.

Definition of concept dictionaries. We argue that discovering relevant *keywords* associated with the concepts reported in Table 2 represents a crucial information for better understanding relationships between *concepts* and *informative user reviews*. Thus, two authors of the paper (separately) labeled each of the sentences in $T_{training}$ (see Section 2.1) with keywords that, in their opinion, led to the assignment of a *review topic* to the sentence. *E.g.*, the keywords "orientation" and "button" can indicate that the sentence deals with the application's GUI. We conjecture that on top of such a labeled dataset it is possible to define an *NLP classifier* able to determine the affiliation of a user review with specific *topics*. Afterwards, the two raters discussed their keyword assignments as well as possible discrepancies and finally created, for each concept, a final collection of keywords (and n-grams).

To make the *dictionaries* more exhaustive we used WordNet [25] to generate synonyms for all the keywords. However, to obtain dictionaries having a good balance between exhaustiveness and coherency, we added the synonyms that are, according to the `Wu & Palmer semantic relatedness` (WUP) [47], at least 50% equal to the original set of keywords. Among all the semantic relatedness metrics, we selected this particular measure because it presents an upper limit (*i.e.*, it is defined between 0 and 1) [47]. More formally, given an ontology O (*i.e.*, WordNet) made by a set of nodes C and a root node R, establishing a threshold of 0.5 to the Wu & Palmer metric means that the least common subsumer (*i.e.*, the nearest common node) between C_1 and C_2 (*i.e.*, the two nodes of the ontology for which we would compute the semantic relatedness) must have a distance N from the root node at least equal to a quarter of the sum of the distances N_1 and N_2 (*i.e.*, the distances between the root node R and the nodes C_1 and C_2, respectively):

$$WUP = \frac{2 * N}{N_1 + N_2} \geq 0.5 \Longrightarrow N \geq \frac{N_1 + N_2}{4} \quad (1)$$

Oracle definition. Once the dictionaries were enriched with WordNet synonyms, one of the authors and an external validator (a software engineer with more than 6 years of industrial experience) separately assigned each sentence in T_{test} (described in Section 2.1) to one or more of the 12 defined concepts. For 851 of these sentences, the two raters assigned the same labels (*i.e.*, inter-rater agreement 89.39%). For the remaining 101 sentences the two raters assigned different labels: 70% of them were sentences belonging to *multiple concepts* and the raters decided to consider both their respective suggested labels as correct; for 30% of these sentences the raters were in disagreement. However, after a separate discussion, they reached an agreement on the final labeling. The manual labelling of each of the 952 sentences in T_{test} constitutes our oracle for evaluating the NLP classifier described in the next paragraph.

Automatic concept identification. We built an *NLP classifier* to automatically assign a sentence in a review to one or more concepts (see Table 2). As first step we stemmed each sentence using the Snowball Stemmer Algorithm [35] to reduce words to their root form. Thus, we built the classifier on top of the *concept-related dictionaries* (*i.e.*, *the concepts dictionaries* defined in the previous sub-steps) to assign to each sentence in a review the probability to belong to one or more concepts. More formally, let S and C be a sentence and a concept respectively; let W_C be the number of tokens in S that also appear in the list of C-related keywords (*i.e.*, *the concepts dictionary* of C defined in one of the previous sub-steps); let W_S be the total number of token in S, the NLP classifier computes the probability that S falls in the concept C as:

$$P_{(S,C)} = \frac{W_C}{W_S} \quad (2)$$

To avoid cases in which sentences are erroneously assigned to a concept C because they contain just one word also appearing in the C-related keyword list, the topic classifier assigns to S all the concepts for which $P_{(S,C)}$ is greater

[4] http://www.ifi.uzh.ch/seal/people/panichella/tools/ARdoc.html
[5] http://bsautermeister.blogspot.ch/2015/12/app-reviews-zu-powernapp-dienen-als.html

than 0.05 (*i.e.*, 5%). We evaluated the effectiveness of the NLP classifier comparing the labels assigned by the classifier with the labels assigned in the human oracle defined in the previous sub-step (*i.e.*, *oracle definition*). The effectiveness of the classifier was evaluated relying on widely adopted metrics in the information retrieval field [2]: precision, recall and F-measure. The NLP classifier performed very well the sentences assignment to topics achieving a global recall of 0.79, a global precision of 0.73, and a global F-measure of 0.76 (seeTable 3). Hence, we used such an NLP classifier in this *Topic Classification* step of SURF.

2.2.4 Sentence Scoring and Extraction

SURF uses a sentence selection and scoring mechanism to generate the summaries which is based on the following observations:

(Obs1). *Maintenance feedback first*: user feedback discussing bug reports and feature requests are more important for developers than all others review types.

(Obs2). *Review topics*: developers need reasonably useful sentences discussing a specific aspect of an app with respect to other review sentences.

(Obs3). *Review length*: longer sentences are usually more informative than shorter ones.

(Obs4). *Popular features*: reviews treating frequently discussed features may attract more attention of developers than reviews dealing with features or functionalities rarely used or discussed by users.

The heuristics defined in this section are calibrated by using reviews of the dataset described in Section 2.1 and validated by two authors of the paper and an external validator (a software engineer with more than 6 years of industrial experience).

Initial clustering and preprocessing. First, we needed to filter out useless sentences and to cluster the remaining ones according to their review topics. In particular, we discarded all sentences with less than 3 tokens (*e.g.*, *"Good app", "Great feature", etc.*), which rarely provide useful information for developers. Then, we cluster sentences belonging to the same *review topic* and we remove duplicated in the summary by applying the following process:

a. All the sentences are preprocessed applying the Snowball Stemming algorithm [35] and stop-words removal.

b. For each pair of sentences (S_i,S_j), we compute the duplicate tokens rate ($DTR_{i,j}$) through the formula:

$$DTR_{i,j} = \frac{DT_{i,j}}{TW_j} \quad (3)$$

where $DT_{i,j}$ is the number of duplicate tokens appearing both in S_i and S_j; TW_i and TW_j are the total number of tokens in S_i and S_j respectively

Table 3: Concepts Classification Results

Concepts	TP	TN	FP	FN	Recall	Precision	F-measure
GUI	82	772	67	31	0.73	0.55	0.63
App	337	451	69	95	0.78	0.83	0.80
Company	6	943	2	1	0.86	0.75	0.80
Contents	114	687	73	78	0.59	0.61	0.60
Download	8	941	3	0	1.00	0.73	0.84
Feature/Functionality	501	193	214	44	0.92	0.70	0.80
Improvement	11	922	11	8	0.58	0.50	0.54
Model	62	850	10	30	0.67	0.86	0.76
Pricing	4	947	0	1	0.80	1.00	0.89
Resources	5	945	0	2	0.71	1.00	0.83
Security	3	948	0	1	0.75	1.00	0.86
Update/Version	82	829	11	30	0.73	0.88	0.80
TOTAL	**1215**	**9428**	**460**	**321**	**0.79**	**0.73**	**0.76**

Table 4: Intention Relevance Scores

Problem Discovery	Feature Request	Information Seeking	Information Giving	Other
3.0	3.0	1.0	1.5	0.5

c. In each pair of sentences (S_i, S_j) that satisfies one of the following condition:

(a) If $TW_j > 5$ and $DTR_{i,j} > 0.75 \Longrightarrow$
  i. S_i is removed, if $TW_j \geq TW_i$
  ii. S_j is removed, otherwise

(b) If $TW_j \leq 5$ and $DTR_{i,j} > 0.85 \Longrightarrow$
  i S_i is removed, if $TW_j \geq TW_i$
  ii S_j is removed, otherwise

Scoring phase. We assign a global score $GS_{(S,C)}$ to each sentence S with respect to a topic C using the equation:

$$GS_{(S,C)} = IRS_S * P_{(S,C)} * [L_S * (1 + MFWR_{(S,C)})] \quad (4)$$

where, the partial scores IRS_S, $P_{(S,C)}$, L_S, and $MFWR_{(S,C)}$ corresponds to our four observations, i.e, Obs1-Obs4. The first partial score IRS_S takes into account our Obs1: it assigns to each sentence S an initial score on the basis of its *intention* category assigned during the classification step. Specifically, we statically provide different relevance scores to each of the *intention* categories defined in Table 1, which are reported for completeness in Table 4. Since different relevance scores could be used, a systematic study on the impact of different scores is part of our future agenda. However, we did not observed any variation of the quality of the generated summaries using different scores. The second partial score $P_{(S,C)}$ is the probability of a sentence S to be relevant for a given topic C, previously defined in section Section 2.2.3. Thus, it is used to deal with Obs2. Indeed, if a sentence S share many words with the list of C-related keywords, then, S is highly related to C resulting in a higher score. According to Obs3, longer sentences are more useful, thus, the length has to be incorporated in $GS_{(S,C)}$. To this aim, we use the third partial score L_S, which denotes the total number of characters constituting the sentence S. Finally, for each sentence S belonging to the topic C, we compute the most frequent words rate, *i.e.*, $MFWR_{(S,C)}$, to take into account our Obs4. In particular, $MFWR_{(S,C)}$ measures the ratio of frequent words appearing in the sentence ($MFR_{(S,C)}$) with respect of the number of total words in the sentence (TW_S):

$$MFWR_{(S,C)} = \frac{MFR_{(S,C)}}{TW_S} \quad (5)$$

Sentences belonging to each of the 12 topics reported in Table 2 are ranked through the scoring function $GS_{(S,C)}$ in equation 4. However, only sentences in the top positions of the ranked list are selected. More formally, let NS_C be the number of total scored sentences for the topic C, the sentences occupying the first $0.7 * NS_C$ positions in the ranked list (*i.e.*, about 2/3 of total sentences) are extracted for further processing, while the remaining sentences, occupying the last $0.3 * NS_C$ positions, are discarded.

2.2.5 Summary Generation

We needed to find a proper way to present the mined review sentences to developers such that they can (i) easily retrieve the necessary information, (ii) properly understand the maintenance tasks to accomplish, and (iii) identify which parts of the app to change. To handle this, we decided to generate summaries as structured HTML since it represents the right compromise between compactness, informativeness and usability. Indeed, using this format the summaries can

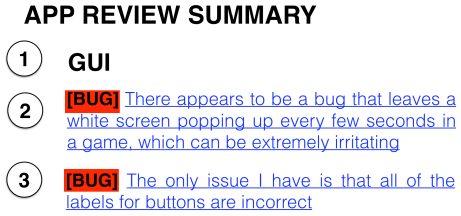

Figure 2: An extract of summary for *Stone Flood*

be easily reported in a hierarchical way, giving to developers control over paths of browsing the information that can be progressively expanded at multiple levels of detail.

In this step, SURF summarizes the sentences collected from user reviews (pre-processed and scored in the previous steps) via *clusterization*, i.e., grouping them according to their *review topics* and *intention* categories. In particular, it performs a two-level clustering: first it groups the sentences according to their topics (*e.g.*, App, GUI, etc.) leveraging on the $GS_{(S,C)}$ scores computed in the previous SURF step; then, sentences in each topic are grouped by *intention* categories, which were assigned during the intention classification step. Finally, groups in the second level are ordered as follows: we first show all the *BUGS* (*i.e.*, the sentences classified as *Problem Discovery*), then all the *REQUESTS* (*i.e.*, *Feature Request*), then all the *QUESTIONS* (*i.e.*, *Information Seeking*) and finally all the *INFO* (*i.e.*, all the sentences classified as *Information Giving* and *Other*).

Moreover, when clicking on each of the sentences reported in the summary, a popup presents to developers the original review from which the specific sentence has been extracted, including app version, user handle, date, star rating, title and the full text. SURF also uses labels to explicitly indicate the belonging semantic category of each sentence (*i.e.*, *BUG, REQUEST, QUESTION,* or *INFO*) with the aim at helping developers in planning and accomplishing specific maintenance task (*e.g.*, implementation of new functionality). Finally, in order to further increase the readability of the summaries, SURF shows a maximum of ten sentences for the INFO category, *i.e.*, only the ten sentences most frequently reported by users in the reviews (highest $GS_{(S,C)}$ score).

Figure 2 depicts an extract of the summary for the app *Stone Flood* (the complete summary can be found online[6]). In particular, Figure 2 reports two feedbacks (extracted from app reviews) that could be beneficial for developers interested in fixing bugs experienced by users (points 2 and 3 in Figure 2) in the UI (point 1 in Figure 2) of the app.

3. STUDY DESIGN

The *goal* of our study is to investigate to what extent the summaries generated by SURF help developers in better understanding user needs and planning future software changes. Specifically, we measure usefulness in the context of a working scenario in which 23 developers and researchers analyzed user feedback contained in user reviews relying on SURF with the goal of identifying feedback useful from a software maintenance perspective (see Section 3.2). The *quality* focus concerns the capability of developers to collect useful user feedback when supported by summarization tools. The *perspective* is that of researchers interested in evaluating the effectiveness of automatic approaches for user feedback summarization when applied in a real working context. We therefore designed our study to answer the following research

[6] http://www.ifi.uzh.ch/seal/people/panichella/tools/SURF/stoneFlood_summary.zip

Table 5: Dataset

App Name	Mined Platforms	Category	# Reviews
Picturex	Apple App Store, Windows Phone Store, Google Play	Photo & Video	120
PowernAPP	Windows Phone Store	Health & Fitness	1050
CSTP	Google Play	Transportation	35
Blinq	Google Play	Lifestyle	299
Karaoke Sing Me FreeLite	Google Play	Music & Audio	94
Karaoke Sing Me	Google Play	Music & Audio	23
Wifi File Transfer	Google Play	Tools	42
Stone Flood	Apple App Store	Games	804
Minesweeper Reloaded	Apple App Store	Games	69
Sheep-O-Block	Apple App Store	Games	35
Doodle Pairs Free	Apple App Store	Games	31
Weight Track	Apple App Store	Healh & Fitness	20
Lifelog Beta	Google+	Healh & Fitness	129
TrackID Beta	Google+	Music & Audio	131
Sketch Beta	Google+	Entertainment	130
Movie Creator Beta	Google+	Photography	84
Video Beta	Google+	Photography	343
TOTAL			3439

questions (RQs):

- **RQ1:** *Is `URM` a robust and suitable model for representing user needs in meaningful maintenance tasks for developers?* Our first goal is to verify whether developers consider URM an enough robust and suitable framework to model users' needs in terms of software maintenance tasks.
- **RQ2:** *To what extent does a summarization technique developed on top of `URM` help mobile developers better understand the users' needs?* We want to assess whether SURF facilitates the analysis of user feedback by developers. Thus, starting from the general RQ2 we derive two further sub-questions that need to be answered to qualitatively and quantitatively measure the practical usefulness of *SURF* and the impact of its generated summaries in the developers' working context:
  - **RQ2-a:** *How do app review summaries generated by `SURF` impact the time required by developers to analyze user reviews?*
  - **RQ2-b:** *How do developers perceive (or consider) the app review summaries generated by `SURF` in terms of correctness, content adequacy, conciseness, and expressiveness?*

3.1 Context

As shown in Table 5 the *context* of our study consists of 3439 reviews from 17 different apps, belonging to 9 different app categories and mined from 4 different online platforms. The reviews are extracted, for each app, in a period of between June 1st, 2015 and January 1st, 2016. In order to evaluate SURF in a real working context, Sony Mobile provided us with access to the Google+ beta test pages of 5 apps (*i.e.*, Lifelog Beta, TrackID Beta, Sketch Beta, Movie Creator Beta, Video Beta) used by developers to collect feedback by beta testers about new functionalities. Table 5 reports the main information of all the apps considered in our study: (i) the app name, (ii) the app category, (iii) the platforms from which reviews have been gathered,

and (iv) the number of reviews. For each user review involved in our study, we collected the title, comment and posting date, the author's nickname, the version of the app to which the review is referring, and the star rating. It is important to highlight that both SURF and URM have been defined relying on app reviews contained in a different dataset from the one used for answer our research questions. Specifically, URM and SURF was defined and calibrated on reviews of the dataset described in Section 2.1 while the evaluation of both URM and SURF was performed considering reviews of the dataset described in Table 5.

3.2 Analysis Method

To answer **RQ1** and **RQ2**, we performed two experiments involving developers, testers, managers and researchers from the Netherlands, Switzerland, Italy and Japan (in total 23 participants) asking them to complete a survey[7] to evaluate: (i) the *suitability and robustness* of *URM* (RQ1), (ii) the *practical usefulness* of *SURF* in a real working environment and how it can speed up the process of collecting useful review feedback (RQ2-a) and (iii) the *quality* of the summaries generated by SURF according to 4 widely known dimensions [26, 34, 43] (RQ2-b):

1. **Correctness**, which measures the accuracy of the automated classification made by `SURF`
2. **Content adequacy**, which assesses whether `SURF` generates summaries containing all important information to understand user needs
3. **Conciseness**, which assesses whether `SURF` generates summaries not containing any superfluous and unneeded information
4. **Expressiveness**, which assesses whether `SURF` produces summaries that are easy to read and whether the way they are presented facilitates the understanding of the user needs

Table 6 summarizes the questions in our survey. We designed two experiments: the first mainly aims at assessing the quality of extracted feedback and the meaningfulness thereof for developers, while the second aims to investigate the practical usefulness of SURF's summaries in a working environment when compared to the analysis of app reviews without the support of the summaries.

Experiment I. To evaluate SURF we first generated the summaries of reviews on 15 apps: Picturex, PowernAPP, CSTP, BLINQ, Doodle Pairs, Karaoke SingMe FreeLite, Karaoke SingMe, Minesweeper Reloaded, Sheep-O-Block, Stone Flood, Weight Track, Wifi File Transfer, Movie Creator Beta, Video Beta, and TrackId Beta. Then we contacted all the original developers of each of the selected apps and asked them to complete the survey and evaluate the summaries generated by SURF. Initially, all of the original developers of such apps confirmed their availability to participate in our study. However, in the end, only six of them were actually able to participate. Consequently, for the remaining apps, we asked other developers (or experts) to evaluate the usefulness of SURF's summaries. Thus, for this first experiment we involved 16 participants in total. 6 of them are the original developers of five applications considered in our dataset (*i.e.*, Picturex, PowernAPP, CSTP, Movie Creator Beta, and Video Beta) while of the others, 4 are researchers in the field of software engineering and 6 are software developers employed in Italian, Swiss, Dutch and Japanese companies. We assigned to each participant an app (except for the six original developers involved in our study) and provided the corresponding summaries generated by `SURF`. After that we explained to the participants the tasks to be performed during the experiment (*i.e.*, how to browse the summaries and validate them) and asked them to answer the questions of our survey.

[7] http://www.ifi.uzh.ch/seal/people/panichella/tools/SURF/Survey.pdf

Table 6: Survey Questions

STEP 1 – EVALUATION OF THE SUMMARY	
Q1	Survey participant has to write down a report that should contain the list of reviews incorrectly (only the incorrect) classified in the automatically generated summary
Q2	Please, report the time required to perform the validation task and report it in minutes and seconds (e.g. "25 minutes and 45 seconds") before to go in the next step
STEP 2 – POST QUESTIONNAIRE	
Q3	How do you judge the usefulness and comprehensibility of the provided summary?
Q4	How difficult is to analyze user feedback contained in reviews WITH the summary?
Q5	How difficult is to analyze user feedback contained in reviews WITHOUT the summary?
Q6	Proportionally, how much time you can save by analyzing feedback contained in user reviews WITH the proposed summary (if compared with the time required WITHOUT the summary)?
Q7	Please rank the categories of reviews reported in the summary in order of importance (from 1 to 12 where 1 is most important to you and 12 is least important to you) from a development point of view
Q7.1	Are missed potential useful categories in the taxonomy above?
Q8	WHITOUT the proposed summary, evaluating the user feedback contained in reviews is prohibitively difficult and/or tedious
Q9	Considering only the content of the summary of user feedback and not the way it is presented, do you think that the report?
Q10	Considering only the content of the summary and not the way it is presented, do you think that the report?
Q11	Considering only the way the summary of user feedback is presented and not its content, do you think that the report?
Q12	Do you have any suggestions to improve the summary and make it more understandable?
Q13	Are the summaries useful for understanding user reviews feedback of mobile apps?

Experiment II. The second is a controlled experiment involving 7 participants, all employed at Sony Mobile. The survey participants had the following profiles: (i) test engineers who do not program but deal with bug db (*e.g.*, Bugzilla), (ii) device driver engineers who have never worked with applications, (iii) product project managers who care about software requirements, deadlines, and people workload, and (iv) team managers who evaluate technical performance. The summaries enrolled in this experiment are regarding the user reviews of two Sony Mobile' apps (Lifelog Beta and Sketch Beta). The data of the reviews was collected from specific Google+ pages of Sony Mobile.

We first separated the participants in two groups: Group 1 (3 subjects) and Group 2 (4 subjects). Then we performed two sub-experiments: Experiment II-A and Experiment II-B. During the Experiment II-A, Group 1 analyzed the original reviews contained in the Google+ page of Lifelog Beta trying to manually extract useful feedback and to classify it according to Tables 1 and 2. Meanwhile, Group 2 analyzed the original reviews contained in the Google+ page of Sketch Beta in the same fashion. Likewise, during Experiment II-B, Group 1 analyzed the summary of Sketch Beta's reviews generated by SURF, with the purpose of validating its content, while Group 2 validated the Lifelog Beta's summary. The time for the entire Experiment II was of 30 minutes (depending on the availability of Sony Mobile' participants): (i) 5 minutes for explaining the purpose of the study, sharing the materials and grouping participants in Group 1 and Group 2, (ii) 10 minutes for experiment II-A and experiment II-B, and (iii) 15 minutes for answering the questionnaire. The short time was necessary to remove the influence of confounding

Table 7: Classification Accuracy

APPLICATION NAME	GUI		APP		COMPANY		CONTENTS		DOWNLOAD		FEATURE/FUNC		IMPROVEMENT		MODEL		PRICING		RESOURCES		SECURITY		UPDATE/VERSION		TOTAL	
	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total	miss-classified	total
Picturex	5	26	0	37	0	0	3	14	0	1	0	45	0	0	4	9	0	1	0	0	0	1	1	6	13	140
PowernAPP	7	50	1	134	3	4	4	28	0	3	7	277	2	11	1	10	0	16	1	1	1	1	10	21	37	556
CSTP	1	2	4	10	0	0	0	1	0	0	2	8	0	1	0	0	0	0	0	0	0	0	0	3	7	25
BUNQ	2	6	2	17	2	2	0	10	0	1	1	24	0	1	0	2	0	0	0	0	0	10	1	5	8	78
Doodle Pairs	0	1	0	10	0	0	0	3	0	1	0	8	0	0	0	1	0	1	0	2	0	0	0	1	0	28
Karaoke SingMe Free/Lite	1	2	1	18	0	0	0	10	0	4	1	22	0	0	0	0	0	1	0	1	0	0	0	1	3	59
Karaoke SingMe	1	8	0	11	0	0	0	3	0	0	3	20	0	1	0	4	0	0	0	0	0	1	0	1	4	49
Minesweeper Reloaded	0	13	1	11	0	0	0	1	0	1	6	23	1	3	0	1	0	1	0	0	0	0	0	5	8	59
Sheep-O-block	0	1	0	7	0	0	0	0	0	1	0	8	0	0	0	1	0	1	0	0	0	0	0	0	0	19
Stone Flood	3	19	4	27	0	0	0	11	0	5	4	58	0	11	0	12	0	10	0	0	0	0	0	15	11	168
Weight Track	0	8	0	12	0	0	0	1	0	0	0	15	0	1	0	0	0	0	0	0	0	0	0	0	0	37
WifiFileTransfer	1	1	2	13	0	0	3	5	0	1	5	16	0	0	4	4	0	0	0	1	0	0	1	1	16	42
Video Beta	12	36	3	24	0	2	2	8	1	12	7	84	0	4	0	11	0	0	0	0	0	0	0	18	25	199
Movie Creator Beta	8	29	3	16	0	0	4	7	0	0	7	46	0	6	0	3	0	0	0	1	0	0	2	9	24	117
TrackID Beta	0	30	0	17	0	0	0	22	0	3	0	56	0	4	0	7	0	1	0	0	0	3	0	18	0	161
TOTAL	41	232	21	364	5	8	16	124	1	33	43	710	3	43	9	65	0	32	1	6	1	16	15	104	156	1737
Correctness rate	**0.82**		**0.94**		**0.38**		**0.87**		**0.97**		**0.94**		**0.93**		**0.86**		**1.00**		**0.83**		**0.94**		**0.86**		**0.91**	

factors (e.g., collaboration with colleagues, access to the web) and have high confidence in the (exclusive) impact of our method on the task's outcome. We plan to replicate our experiments involving developers of further apps, performing longer evaluation tasks.

For each task, the participants reported the number of reviews they analyzed and classified during Experiments II-A and II-B and answered the questions of our survey.

3.3 Research Method

In order to assess the suitability and the robustness of *URM* and answer **RQ1** we asked the survey participants to rank the *categories* of reviews reported in the summaries (Q7 in Table 6) and to also suggest possibly missing categories (Q7.1 in Table 6).

To answer **RQ2-a** we asked the survey participants to report the time required for performing the validation of summaries we provided (Q2 in Table 6) and to express their opinion on the speed-up introduced by the summary in analyzing the user feedback (Q6 in Table 6). To qualitatively complement this quantitative information we also asked respondents to judge the usefulness and the comprehensibility of the provided summaries (Q3 in Table 6), and to provide their opinion on any difficulties when analyzing the user feedback in the reviews WITH or WITHOUT the provided summary (Q4 and Q5 in Table 6).

Finally, to answer **RQ2-b** we asked survey participants to manually validate the classification correctness of data contained in the summaries (Q1 in Table 6) and provide their opinion on (i) the content adequacy (Q9 of Table 6), (ii) the conciseness (Q10 of Table 6), and (iii) the expressiveness (Q12 of Table 6). We complement this data by asking survey respondents for their opinion on the statement in Q8 of Table 6 and for a general judgment on the usefulness of the summarization approach in a real working context (Q13 in Table 6).

4. RESULTS

In this section we summarize the results obtained in our experiments.

4.1 RQ1 Results

To answer RQ1, we analyzed replies collected from survey participants of both experiments.

78.26% (18 out of 23) of participants declared that URM is not missing any relevant information and that the topics considered in URM are exhaustive, or maybe even too detailed (*i.e.*, just one participant complained about the number of topics in the model:*"... don't need many categories, 3 - 5 categories are enough..."*). The remaining feedback comprised proposals to (i) discriminate change requests between *adaptive* and *corrective* maintenance, (ii) add a *raging customers* category, (iii) introduce an *advertisement* category, (iv) add a topic dealing with *"the connection to external devices"*, and (v) insert a topic treating the *"comparison with other apps"*.

Since the aim of our work is to facilitate the extraction of useful feedback from a maintenance perspective, we believe that the *raging customers* and *advertisement* categories would not add valuable information for developers, while the other suggestions can be considered for future improvements of the model.

From a software development perspective participants considered the *most important topics* highlighted by *URM* to be: (i) the *App* (82.61% of participants ranked it in the first three positions), (ii) the *GUI* (60.87% of participants ranked it in the first three positions), and (iii) the *Feature or Functionality* (34.78% of participants ranked it in the first three positions). Vice versa, the topics considered least important are (i) the *Company* (69.57% of participants ranked it in the last three positions), (ii) the *Model* (65.22% of participants ranked it in the last three positions), and (iii) *Download* (52.17% of participants ranked it in the last three positions). A possible explanation for these ranking results may be related to the fact that app developers are more concerned with collecting functional requirements rather than non-functional ones. Survey respondents also considered the *intention* classification of sentences that *URM* provides to be very interesting. For instance, some participants say that *"... I found the classification GUI-BUG, APP-BUG, etc very useful..."*, *"... in case I'm searching for BUGs, I can just look for the category, instead of reading everything over and over again..."*, *"... categorization of reviews with a summary is very helpful to me. Especially the categories/topics related to bugs, crashes and security issues..."*. In summary, we can conclude that:

> **RQ1** *According to the developers' judgment URM has shown to be robust and suitable enough for representing user needs in meaningful maintenance tasks and to be usable in practical contexts. The most important topics modeled in URM are the App, GUI and Feature or Functionality categories.*

4.2 RQ2 Results

4.2.1 RQ2-a Results

Experiment I results. Survey participants spent, on average, 28 minutes and 12 seconds for browsing and validating the summaries. 93.75% (15 out of 16) of participants considered our approach time-saving: 75% (12 out of 16) of survey respondents replied that SURF allows them to save at least one third of the time that they would otherwise have spent

on manually collecting and analyzing user reviews. Among these 12 participants, 9 subjects (75%) claim that, with the summaries, the saved time ranges between 50% and 95%, while 18.75% (3 out of 16) of participants affirmed that our approach allows to save 10%-20% of time. Only one subject stated that summaries do not allow to save any time. We noticed that among the apps having less than 100 reviews, 67% (6 out of 9) of subjects declared that our approach allows to save more than 50% of time, but among the apps having more than 100 reviews we observe a degradation of this result. Indeed, for such apps the subjects who claim 50% as the saved time are just 3 out of 7 (47%). A possible explanation for this degradation could reside in the fact that, for apps with more reviews, SURF generates longer summaries which need more time to be read and analyzed and this could influence the perception of the real gain of time enabled through the approach.

These results are related to the time saving capability of SURF *perceived by developers*. However, this measure is fairly subjective and a more quantitative evaluation is required. Thus, in order to avoid a biased estimation of these results, we discuss the *validation times* declared by the subjects in the survey (Q2 in Table 6) and compare them with the execution times required by SURF to generate the summaries. The execution times have been measured through a 32-bit Intel Celeron Dual-Core CPU T3500 2.10 GHz machine with 2 GB DDR3 RAM running Windows 7. On average the tool spent about 1.05 seconds per review. Indeed, for analyzing each app, that in average has 220 reviews, SURF requires an average execution time of 3 minutes and 51 seconds. The 15 summaries involved in Experiment I contain, in total, 1737 sentences and each summary contains, on average, around 116 sentences. Considering that the average number of sentences is 116, survey participants spent an average validation time of 14.59 seconds per sentence (since the average validation time in the experiment is 28 minutes and 12 seconds). This means that, when supported by SURF, a developer spent, for extracting, analyzing and validating each user feedback an average total time of 15.64 seconds (*i.e.*, 14.59 seconds + 1.05 seconds). Guzman *et al.* [12] demonstrated that, for a fine-grained manual analysis of 900 user reviews, in the best case, developers spent 8 hours. We can infer that validators in [12] spent an average time of 32 seconds for analyzing each user review feedback. Thus, when supported by SURF, the *time required by developers for analyzing each user feedback decreases by at least 51.12%* (from 32 to 15.64 seconds).

From a qualitatively point of view, 87.50% of subjects considered the provided summaries highly/enough useful and comprehensible (see Q3 in Table 8) and 62.50% of participants affirmed that analyzing user feedback with the provided summaries is easy (see Q4 in Table 8).

Experiment II results. The results of Experiment II empirically confirm most of the results obtained in Experiment I. However, contrary to Experiment I, only 3 out 7 participants (42.86%) declared that our approach allows to save 30%-50% of time. In this case the time saving capability of SURF *perceived by developers* is, again, subjective and a more quantitative evaluation is needed. We notice that in Experiment II-A, when analyzing user feedback using Google+ pages during a time slot of 5 minutes, each of the subjects extracted, on average, 5.86 feedbacks useful for software maintenance. Vice versa, in Experiment II-B, when analyzing user feedback using SURF's summaries, in the same

Table 8: Aggregated answers from participants

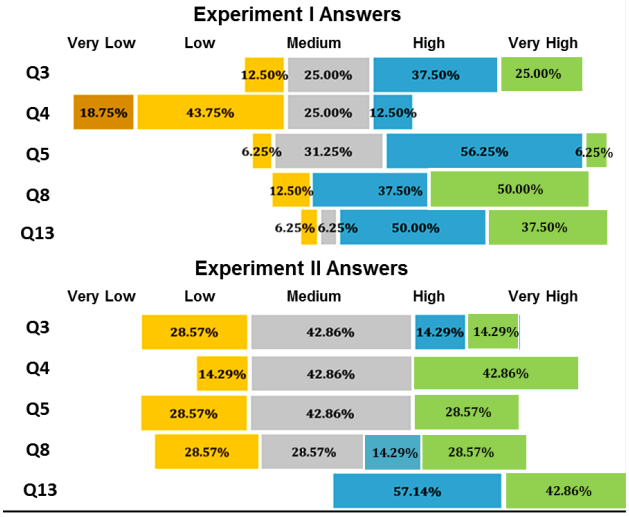

Table 9: Raw data of the questionnaire concerning the evaluation of SURF summaries.

Content adequacy

Response category	% of Ratings	
	Exp I	Exp II
A) Is not missing any information.	43.75%	14.28%
B) Is missing some information but the missing information are not necessary to have an overview of users' needs	25%	28.57%
C) Is missing some very important information	25%	42.86%
D) Not sure	6.25%	14.29%

Conciseness

Response category	% of Ratings	
	Exp I	Exp II
A) Has no unnecessary information.	87.50%	28.57%
B) Has some unnecessary information.	12.50%	57.14%
C) Has a lot of unnecessary information.	0%	0%
D) Not sure	0%	14.29%

Expressiveness

Response category	% of Ratings	
	Exp I	Exp II
A) Is easy to read and understand.	68.75%	28.57%
B) Is somewhat readable and understandable.	18.75%	57.14%
C) Is hard to read and understand.	12.50%	14.29%

time, subjects extracted in average many more feedbacks, *i.e.*, 16.57 instead of 5.86. This means that for extracting a single useful feedback without the summaries, the participants spent, on average, 51.19 seconds, while for analyzing (and validating) a single feedback extracted through SURF, the subjects spent, on average, 18.10 seconds. For the two apps involved in the Experiment II (*i.e.*, Lifelog Beta and Sketch Beta) there were, in total, 259 user reviews and the tool had a total execution time of 10 minutes and 58 seconds for generating the two summaries. Thus, considering that, in this experiment, the average execution time required by the tool for each review is 2.54 seconds, we can conclude that when supported by SURF the time required by participants for extracting (and analyzing) each user feedback decreases from 51.19 to 20.64 (18.10 + 2.54) seconds, *i.e.*, 59.68%.

It's important to highlight that 27 out of 41 (*i.e.*, 66%) of feedbacks manually extracted by subjects also appear in the summaries automatically generated by SURF. In summary, we can conclude that:

RQ2-a *Developers consider the summaries generated by SURF useful (manually extracted feedback appears also in the automatic generated summaries) and comprehensible. SURF helps to prevent more than half of the time required by developers for analyzing users feedback and planning software changes.*

4.2.2 RQ2-b Results

Experiment I results. Table 7 shows for each app (each row) and each app topic (each column) the amount of sentences misclassified by SURF according to the judgment of respondents as well as the total sentences appearing in the summary. The validation task performed by the survey participants highlights the very *high classification accuracy of SURF*, which is 91%.

68.75% (11 out of 16) of subjects considered the summaries' content to be adequate: 7 out of 11 participants affirmed that summaries we provided have no loss of information, while 4 of them declared that, even if there were a possible information loss, our summaries still provide a complete overview of user needs (see Table 9). On the contrary, 4 subjects out of 16 (25%) believed that some important information does not appear in the summaries (*e.g.*, pictures or videos of the original reviews that could explain the context for reproducing bugs).

About conciseness, 87.50% of subjects (14 out of 16) declared that *summaries do not contain unnecessary information* (see Table 9), while 2 subjects out of 16 (12.50%) said that they contain just *some* unnecessary information (*e.g.*, sentences that do not provide any useful information for developers).

As shown in Table 9, 87.5% (14 out of 16) of *subjects consider the provided summaries readable and easy to understand* (11 out of 14) or somewhat readable and understandable (3 out of 14). For example, participants say about the summaries generated by SURF: *"...it's simple & useful...", "...everything is very compact and we have a good overview...", "...The summaries are very concise and easy to read...", "...The information is more on less reorganized and expressed in form of change requests which I find very useful...", "...It is very clear and hierarchically organized...", "...The summary is self explanatory...", "...the use of pop-ups allows a developer to trace the entire user comment..."*). 2 subjects (12.50%) believed that the summaries are hard to read and understand (*e.g.*, *"...It is presented in a very old-fashioned HTML style"*).

Moreover, 87.5% of subjects considered SURF useful in a working context (see Q13 in Table 8): *"...If we have more huge feedbacks, I think this system is useful. I think this can be used for not only reviewing details of each feedback but also understanding statistical bug/request trends...", "...The tool you propose is very useful to highlight the most useful reviews. Without your tool reviews are just reviews, not requests..."*).

Experiment II results. 42.86% of participants (3 out of 7) declared that the provided summaries are lacking some important information (see Table 9), like screenshots posted by beta-tester useful for reproducing certain bugs (*i.e.*, 4 out of 7 participants complain about this issue). However, screenshots are not allowed in the user reviews of the most popular mobile distribution platforms and our approach was originally conceived to collect data from these platforms, and not from pages hosted on a general purpose social network (*i.e.*, Google+). As shown in Table 9, 28.57% of the subjects (2 out of 7) affirmed that summaries do not contain unnecessary information, while 4 participants (*i.e.*, 57.14%) said that they contain some unnecessary information (*e.g.*, sentences not providing any useful information for developers or duplicate data). Regarding the expressiveness, 85.71% of the participants (6 out of 7) claimed the summaries are readable and understandable (see Table 9). Furthermore, *all the subjects generally considered the summaries highly useful for better understanding user needs* (see Q13 in Table 8). In summary, we can conclude that:

> **RQ2-b** *According to the survey participants, the summaries generated by SURF are reasonably correct, adequate, concise, and expressive. Moreover, they are also considered useful in a real "**working context**".*

Feedback of survey participants. Some of the participants suggested to extend `SURF`'s functionality to make them even more readable For instance: *"...I would improve the report design with a more readable interface and by including extra information...", "...To make the summary more immediate would need to enter a statistical graph for all topics...", "...a better graphic: a graphic report of the amounts of bugs, requests, info etc, a navigation bar or menu, hierarchically expandable categories/folders, some categories filters and a more efficient visual classification of the distributions and quality of the feedbacks..."*).

5. THREATS TO VALIDITY

This section outlines potential threats to the validity of our study.

Internal validity. These threats concern confounding factors that could influence our results. The main threat to internal validity in our study is that the assessment is performed on data provided by human subjects, hence it could be biased. To counteract this issue, we selected 23 different subjects in our study who (i) meet different profiles in the field of software development, (ii) have different cultural backgrounds (*i.e.*, they come from 4 different nations), and (iii) 6 of them were the original developers of 5 apps in our dataset (see Section 3.1). Moreover, in the Experiment I, the validation times are manually reported by the subjects: they could entail imprecisions. For alleviating this issue we provided precise timing instructions to participants. Another issue consists in the preventive assignment of some scores. We statically assign some of the values on the basis of certain observations. Further investigation is needed to establish whether different distributions of these values enable better results.

External validity. These kinds of threats relate to the generalizability of our findings. Our experiments are small in scale (*i.e.*, 17 apps out of millions) and the chosen apps may not be representative. To mitigate this issue we investigated user reviews of apps belonging to 9 different app categories and mined 4 different online platforms (see Section 3.1). Specifically, to assess the robustness of SURF in classifying and summarizing user feedback, we considered reviews that contain different vocabularies and were written by different user audiences: users of these apps (i) interact with devices and technologies in different ways, (ii) belong to different age groups, and (iii) have different expectations. In Experiment II, participants validated the summaries for a total time of 10 minutes. Specifically, the availability of Sony Mobile' participants in performing Experiment II was only 30 minutes, of which (i) 5 minutes were spent explaining the purpose of the study, sharing the materials and grouping participants in Group 1 and Group 2, (ii) 10 minutes were used for conducting Experiment II-A and Experiment II-B, and (iii) 15 minutes were available for answering the questionnaire. Manually analyzing the user feedback for 10 minutes is not sufficient to generalize our findings. However, in Experiment I, participants had all the time they needed to validate recommended feedback and analyze reviews. It is

important to highlight that the gain times (RQ-a) obtained when using SURF are almost identical in Experiment I and Experiment II. In future work we are interested in replicating the study involving additional developers of other companies to consolidate our findings. Finally, only one of the apps in our dataset is not free (Karaoke SingMe). Commercial apps may have different review patterns. In the future, for alleviating this threat, we plan to extend the investigation to the user reviews of more commercial apps.

6. RELATED WORK

Several researchers have focused on mining and analyzing app data with the goal of deriving important information to help developers evolve their apps [9, 15, 29, 30, 33, 40, 41]. First of all, Harman *et al.* introduced the concept of app store mining and identified important correlations between the customer rating and the download rank of apps [15]. Pagano *et al.* investigated the correlation between reviews and ratings [29] while Bavota *et al.* presented a study revealing a direct relationship between the rating of an app and the fault-proneness of the underlying API used by the developers [3].

Recently, approaches have been proposed for automatically mining requirements from app reviews [4, 10, 12, 17, 23, 31, 33, 39]. For instance, Chandy *et al.* used a Latent Class graphical model, clustering reviews to identify spam [4]. Gu *et al.* on the other hand focused on identifying positive reviews using a pattern-based parsing approach to extract opinions and summarize reviews [11]. Guzman *et al.* performed sentiment analysis to extract coherent features from reviews that relate to requirements evolution tasks [12].

Numerous researchers have also applied natural language processing and text retrieval techniques towards automating the extraction of useful content from app reviews. Specifically, Iacob *et al.* used Latent Dirichlet Allocation to extract feature requests from user reviews [17] while Carreno *et al.* use topic modeling to extract common topics in a corpus of user reviews and summarize each topic with a few sentences [10]. Chen *et al.* presented a computational framework which automatically groups, prioritizes and visualizes informative reviews [5]. Likewise, Maalej *et al.* classified app reviews into bug reports, feature requests, user experience and ratings [23]. Panichella *et al.* [33] proposed an approach that combines natural language parsing, sentiment analysis and text analysis techniques, through a Machine Learning (ML) algorithm in order to detect sentences in user reviews that are important from a maintenance perspective. We rely on this approach for performing the intention classification described in Section 2.2.2. Most of the automated approaches discussed above perform an automatic classification (or clustering/prioritization) of user reviews according to specific topics (*e.g.*, bugs, features etc.) and, as pointed out by Gu *et al.*, they *"are based on a bag-of-word assumption without considering sentence structures and semantics"* [11].

Our research represents a cross-section of the field of issue categorization [16] and classification/summarization of natural language corpora [7, 8, 32, 36–38, 44] or source code [6, 14, 24, 26, 27, 42, 43] in the specific context of mobile app reviews. From an engineering point of view, our work extends the line of research on mining requirements from app reviews and it is novel for three main reasons. First of all we present `URM`, a conceptual model that takes into account both the users' *intentions* (when giving feedback to developers) and the *aspect* concerning the app, which is the subject of the review, to model users' needs in terms of software maintenance and evolution tasks that are important for developers. Moreover, we propose a novel approach, namely SURF, which combines sophisticated summarization techniques for generating (according to URM) summaries of thousands of user reviews in form of an *interactive*, *structured* and *condensed agenda of future change tasks*. Finally, we evaluate the impact of the generated summaries in a real scenario where developers analyzed feedback in user reviews of real mobile apps with the support of SURF's summaries.

To the best of our knowledge only the work by Gu *et al.* [11] is closely related to ours. Indeed, Gu *et al.* proposed a summarization framework, called SURMiner, which classifies reviews into five categories and extracts aspects (or opinions) in sentences. The approach by Gu *et al.* considers *aspect evaluation* sentences, from which it extracts aspect-opinion pairs, associates a sentiment value to each pair and provides developers with positive and negative opinions about each aspect. Our approach is based on URM, which models the user feedback as explicit maintenance tasks considering a broader set of review types (see Table I and II). Thus, it does not only deal with feedback reporting user opinions about aspects (that are modeled in the *Information Giving* category). SURF's summaries present to developers also feature requests and bug reports related to specific parts of an app.

7. CONCLUSION

SURF's ability to summarize thousands of app reviews in form of an *interactive*, *structured* and *condensed agenda of recommended software changes* represents a significant step forward on the cutting edge of app review mining. SURF assists developers in effectively and quickly understanding user needs and substantially reduces the time required for both the manual analysis of user feedback and the planning of software changes. Through SURF a developer can quickly become aware of requests and issues reported by the users, for instance to add a new option in the UI, to fix a stability issue or to protect the privacy of the user, all without analyzing hundreds of useless reviews. We assess the robustness and suitability of ***URM*** (RQ1) as well as the usefulness of ***SURF*** (RQ2) in a working context by conducting two experiments involving 17 real-word apps and 23 subjects having different profiles in the field of software development.

In future work, we plan on making the summarization interface more interactive, using more effective visualizations of feedback distributions. Moreover, we are interested in (i) improving the classification capabilities of SURF by adding new natural language heuristics, (ii) adding internationalization (as currently only English reviews are supported) and (iii) implementing, on top of SURF, a mechanism able to recognize which part of the source code needs to be changed in the app to perform the change tasks suggested by SURF. We finally, plan to integrate a prioritization mechanism in SURF to help developers focus on the most relevant tasks.

Acknowledgments

We thank Alessandro Di Sorbo for his help in the human oracle building of our approach. We also thank all practitioners from Italy, Switzerland, Netherlands and Japan, and employees of Sony Mobile (Tokyo), who participated to our study. Finally, Sebastiano Panichella gratefully acknowledges the Swiss National Science foundation's support for the projects *"Essentials"* (SNF Project No. 200020−153129) and *"SURF-MobileAppsData"* (SNF Project No. 200021−166275).

8. REFERENCES

[1] U. Abelein, H. Sharp, and B. Paech. Does involving users in software development really influence system success? *IEEE Software*, 30(6):17–23, 2013.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. The impact of api change- and fault-proneness on the user ratings of android apps. *Software Engineering, IEEE Transactions on*, 41(4):384–407, April 2015.

[4] R. Chandy and H. Gu. Identifying spam in the ios app store. In *Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality*, WebQuality '12, pages 56–59, New York, NY, USA, 2012. ACM.

[5] 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*, ICSE 2014, pages 767–778, New York, NY, USA, 2014. ACM.

[6] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, 19(5):1383–1420, 2014.

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

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

[9] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284, New York, NY, USA, 2013. ACM.

[10] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591, Piscataway, NJ, USA, 2013. IEEE Press.

[11] X. Gu and S. Kim. What parts of your apps are loved by users? (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 760–770, 2015.

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

[13] E. Ha and D. Wagner. Do android users write about electric sheep? examining consumer reviews in google play. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 149–157, Jan 2013.

[14] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus. On the use of automated text summarization techniques for summarizing source code. In *Proceedings of the International Working Conference on Reverse Engineering (WCRE)*, pages 35–44. IEEE, 2010.

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

[16] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 392–401, Piscataway, NJ, USA, 2013. IEEE Press.

[17] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44, Piscataway, NJ, USA, 2013. IEEE Press.

[18] C. Iacob, R. Harrison, and S. Faily. Online reviews as first class artifacts in mobile app development. In G. Memmi and U. Blanke, editors, *Mobile Computing, Applications, and Services*, volume 130 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 47–53. Springer International Publishing, 2014.

[19] V. N. Inukollu, D. D. Keshamoni, T. Kang, and M. Inukollu. Factors Influencing Quality of Mobile Apps:Role of Mobile App Development Life Cycle. *ArXiv e-prints*, Oct. 2014.

[20] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[21] S. Krusche and B. Bruegge. User feedback in mobile development. In *Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle*, MobileDeLi '14, pages 25–26, New York, NY, USA, 2014. ACM.

[22] S. A. Licorish, A. Tehir, M. F. Bosu, and S. G. MacDonell. On satisfying the android os community: User feedback still central to developers' portfolios. In *2015 24th Australasian Software Engineering Conference (ASWEC*, pages 78–87, 2015.

[23] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 116–125, Aug 2015.

[24] P. W. McBurney and C. McMillan. Automatic documentation generation via source code summarization of method context. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 279–290. ACM, 2014.

[25] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[26] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker. Automatic generation of natural language summaries for java classes. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 23–32. IEEE, May 2013.

[27] G. C. Murphy. *Lightweight Structural Summarization As an Aid to Software Evolution*. PhD thesis, 1996.

AAI9704521.

[28] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song. Facilitating developer-user interactions with mobile app review digests. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 1809–1814, New York, NY, USA, 2013. ACM.

[29] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *In Proceedings of the 21st IEEE International Requirements Engineering Conference (RE 2013)*, pages 125–134. IEEE Computer Society, 2013.

[30] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 291–300, Sept 2015.

[31] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[32] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora. Mining source code descriptions from developer communications. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 63–72, 2012.

[33] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can I improve my app? classifying user reviews for software maintenance and evolution. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 281–290, Sept 2015.

[34] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. Gall. The impact of test case summaries on bug fixing performance: An empirical investigation. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2016.

[35] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[36] O. Rambow, L. Shrestha, J. Chen, and C. Lauridsen. Summarizing email threads. In *In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL) Short Paper Section*, 2004.

[37] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: A case study of bug reports. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 505–514, New York, NY, USA, 2010. ACM.

[38] S. Rastkar, G. C. Murphy, and G. Murray. Automatic summarization of bug reports. *IEEE Trans. Softw. Eng.*, 40(4):366–380, Apr. 2014.

[39] F. Sarro, A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 76–85, 2015.

[40] N. Seyff, G. Ollmann, and M. Bortenschlager. Appecho: A user-driven, in situ feedback approach for mobile platforms and applications. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, MOBILESoft 2014, pages 99–108. ACM, 2014.

[41] A. Sharma, Y. Tian, and D. Lo. Nirmal: Automatic identification of software relevant tweets leveraging language model. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 449–458, March 2015.

[42] G. Sridhara. *Automatic Generation of Descriptive Summary Comments for Methods in Object-oriented Programs*. PhD thesis, Newark, DE, USA, 2012. AAI3499878.

[43] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 43–52. ACM, 2010.

[44] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora. Codes: Mining source code descriptions from developers discussions. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 106–109, New York, NY, USA, 2014. ACM.

[45] T. Vithani. Modeling the mobile application development lifecycle. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2014, Vol. I*, IMECS 2014, pages 596–600, 2014.

[46] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach. *CoRR*, abs/1505.04657, 2015.

[47] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ACL '94, pages 133–138, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.