ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/303515399

# Approaches for prioritizing feature improvements extracted from app reviews

**Conference Paper** · June 2016

DOI: 10.1145/2915970.2916003

CITATIONS
61

READS
628

**3 authors:**

Swetha Keertipati
University of Otago
**4** PUBLICATIONS **109** CITATIONS
SEE PROFILE

Bastin Tony Roy Savarimuthu
University of Otago
**170** PUBLICATIONS **1,731** CITATIONS
SEE PROFILE

Sherlock A. Licorish
University of Otago
**130** PUBLICATIONS **1,843** CITATIONS
SEE PROFILE

All content following this page was uploaded by Sherlock A. Licorish on 31 December 2017.

The user has requested enhancement of the downloaded file.

**Full citation**: Keertipati, S., Savarimuthu, B. T. R. and Licorish, S. A. 2016. Approaches for prioritizing feature improvements extracted from app reviews, in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE 2016) (Limerick, Ireland, June 1-3, 2016). ACM, 1-6. 10.1145/2915970.2916003.

# Approaches for Prioritizing Feature Improvements Extracted from App Reviews

Swetha Keertipati
Department of Information Science
University of Otago
Dunedin, New Zealand
swetha.keertipati@otago.ac.nz

Bastin Tony Roy Savarimuthu
Department of Information Science
University of Otago
Dunedin, New Zealand
tony.savarimuthu@otago.ac.nz

Sherlock A. Licorish
Department of Information Science
University of Otago
Dunedin, New Zealand
sherlock.licorish@otago.ac.nz

## ABSTRACT
App reviews contain valuable feedback about what features should be fixed and improved. This feedback could be 'mined' to facilitate app maintenance and evolution. While requirements are routinely extracted from post-release users' feedback in traditional projects, app reviews are often generated by a much larger client-base with competing needs and priorities and ad hoc structure. Although there has been interest aimed at exploring the nature of issues reported in app reviews (e.g., bugs and enhancement requests), prioritizing these outcomes for improving and evolving apps hasn't received much attention. In this preliminary study we aim to bridge this gap by proposing three prioritization approaches. Driven by literature in other domains, we identify four attributes (frequency, rating, negative emotions and deontics) that serve as the base constructs for prioritization. Thereafter, using these four constructs, we develop three approaches (individual attribute-based approach, weighted approach and regression-based approach) that may help developers to prioritize features for improvements. We evaluate our approaches in constructing multiple prioritized lists of features using reviews from the MyTracks app. It is anticipated that these prioritized lists could allow developers to better focus their efforts in deciding which aspects of their apps to improve.

## Keywords
App reviews; crowdsourcing; prioritization; requirements elicitation.

## 1. INTRODUCTION AND BACKGROUND
Determining the software features that users require is dealt with in the Requirements Engineering (RE) stage of the Software Development Life Cycle (SDLC). This activity is established to be quite challenging, as Brooks noted "the hardest single part of building a software system is deciding precisely what to build" [1]. While traditional and agile requirements elicitation processes involve users specifying the software requirements upfront or during the SDLC, that are then developed, apps are generally released with a feature cohort that is subsequently expanded based on feedback available in the form of app reviews [2]. Also, unlike traditional and agile software developments where the feedback about post-release changes required come from the same client(s), requirements[1] come from numerous clients in the app domain. These clients may have (conflicting) preferences about features that need to be fixed or enhanced. Furthermore, app reviews are usually voluminous and ambiguous [3] compared to users' feedback provided in traditional and agile software projects. Thus, developers often need approaches to identify and extract these requirements from reviews, before prioritizing in which order they should be fixed.

While there have been works studying the nature of complaints [4, 5] and identifying features of apps that need to be improved [6], prioritization of features that need to be fixed hasn't received much attention. To address limitations of well-known prioritization techniques such as analytic hierarchy process AHP that suffer from scalability problems [7], researchers have proposed new techniques such as clustering [8], case-based ranking [9], and the B-Tree method [10]. However, these methods are not suitable for prioritizing requirements from app reviews because of the volume of reviews that are logged by users, and the ensuing concerns (e.g., identifying and grouping issues from natural language text, and determining the severity of a problem). Also, these methods assume that clients have clear priorities and they participate in the prioritization process to rank-order feature preferences. For example, the study of Laurent et al. [8] assumes that the identified requirements 'clusters' will be rank-ordered by clients, and the work of Avesani et al. [9] requires clients to be involved in the preference elicitation step. In contrast, in the app domain, there is no single client. Thus, app developers must face the challenging task of extracting a unique set of features from unstructured reviews that require action, and then subsequently employ suitable mechanisms for assigning priorities.

In contrast, gathering feedback and assigning priorities are well-structured in larger projects such as Android OS that receive voluminous feedback similar to the app domain. For example, the issues logged into appropriate categories (e.g., bugs and enhancements) are typically assigned priorities based on severity (e.g., low, medium and high) by a dedicated group of individuals with domain knowledge. However, in the app review domain, features need to be first extracted from unstructured reviews before undertaking prioritization. As noted above, this poses a challenge to development teams, and particularly those with limited resources. Also, the prioritization approach often needs to

[1] In this work, we consider users' requests for bug fixes, enhancements and new features logged in reviews (under the umbrella of *improvements*), as a form of crowdsourced requirements, which aids software maintenance and product evolution.

consider aspects such as the prevalence of an issue (e.g., frequency of a reported enhancement request) amongst the user population, and the seriousness of an issue (e.g., a feature not working versus a feature needing to load faster). Some app development firms employ community managers [11] who (manually) monitor and analyze app reviews, identify issues, respond to customers, and also raise issues that need to be fixed to the development team, thus contributing towards the prioritization process. Though beneficial for community engagement [11], apart from being tedious and time consuming, this may not be affordable for small groups of app developers. These issues call for structured approaches to prioritize features for improvements logged in app reviews (following a data-driven perspective).

***How can we use data-driven approaches to prioritize features for improvements extracted from user reviews?*** To answer this question we have developed three approaches which are likely to reduce the amount of manual work required in prioritizing features for improvements. Although preliminary, we anticipate that these approaches may have utility for app developers, and may also stimulate research interest.

The rest of the paper is organized as follows. In Section 2 we describe the three specific approaches for prioritizing features for improvements. In Section 3, we conduct an evaluation to assess the suitability of our approaches using reviews from the MyTracks[2] app. In Section 4 we compare the three approaches, highlight the implications, and review limitations and future work.

# 2. FEATURE PRIORITIZATION

In this section, we first discuss the steps to identify features that need to be fixed from app reviews. We then discuss how the four attributes considered for prioritization are identified. Finally, we discuss the three prioritization approaches.

## 2.1 Feature identification

We first extracted the latest 4,442 reviews for the MyTracks app from the Google Play Store using Google API. We had previously employed other text mining techniques in studying reviews from this app, so it was convenient to use these reviews to evaluate our prioritization approaches [12]. Each review contained a title, review comment (which may identify bug fixes, enhancements and new features) and the rating provided by the user. We followed earlier work [3, 12] which show that multiple issues may be reported in one review, and conducted our analysis at the sentence-level. Thus, each review was parsed into sentences, based on a set of delimiting characters (e.g., '.' and '!'). The next step was to identify the sentences that met the criteria for the study. Since our focus was to study features that need improvements, we include those sentences that were a part of reviews that had low ratings (i.e., =< 3, similar to [13]). We also anticipated that sentences that contained negative terms and emotions would highlight users' dissatisfaction with app features, and these were tagged for analysis. This pre-processing resulted in 1,271 sentences for the MyTracks app (out of a total of 8,623 sentences from 4,442 reviews), which were used for further analyses. Prior work has shown that important features discussed in the reviews can be extracted using Part-of-Speech (POS) tagging [14] and N-gram analysis [15] techniques. In this work, we employ these well-known techniques and identify nouns in sentences to represent features that need to be fixed, similar to the work in [14, 16].

[2]https://play.google.com/store/apps/details?id=com.google.android.maps.mytracks

## 2.2 Attributes considered for prioritization

The next step of the process involved the identification and extraction of attributes that can be used as the basis for prioritizing features for improvements. Three proposed approaches (discussed in the next subsection) were developed based on four different attributes of app reviews—frequency, ratings, negative emotions and deontics. These four attributes are derived from literature in three different domains. The first two are characteristics that have been used in a range of works in app-review mining in different contexts [13, 14, 17]. Guzman and Maalej's [14] study used feature count (frequency) to identify important features, and Fu et al. [13] used the rating of reviews to detect inconsistent comments (i.e., if there was discrepancy between rating provided and the sentiment expressed in review comments). The negative emotion and deontics attributes were derived from psychological [13] and sociological literature respectively [18]. We discuss these four attributes and justify their selection in this work below.

### *2.2.1 Frequency*

Frequency (or count) of a feature can reveal the magnitude of the problem associated with that feature [14]. In this work the frequency is computed by checking whether a given feature (e.g., battery) is present in each of the sentences. If the feature is present, its count is incremented. Each feature is searched for only once in the sentences. The features are then rank-ordered using the count measure.

### *2.2.2 Rating*

The rating attribute is based on the rating score (one to five) provided by app users when rating the app. For each feature that appears in the sentences, a running sum of the rating that sentence received is calculated, and then the average is found based on the number of times it appears. The feature with the lower rating would receive a higher priority.

### *2.2.3 Emotions*

Psychologists have analyzed emotion expressed in text using categories identified by Parrott [19], who identified over 100 emotions and classified them into the primary emotion categories—anger, fear, joy, love, sadness, and surprise. Research has revealed that negative emotions such as anger, sadness and fear could signal discontent of users [13]. Thus, we consider words that belong to the three negative categories (sadness, anger, and fear) in our analyses. The words for our list of emotions were obtained from the LIWC dictionary [20], and from word lists used in other emotions-based research [21]. Words from the LIWC categories for sadness, anger, and anxiety (i.e., fear) were used. Features mentioned in sentences were queried for their association with the words in each of the emotion words in the lists. If a feature was present along with a word from a category, the count was incremented for that category. The feature with the higher count of negative emotions increased in priority.

### *2.2.4 Deontics*

Sociologists investigating social-media data to obtain insights have noted that deontic terms such as 'must', and 'must not' are used to indicate the persuasiveness of a message [22], signaling obligations and prohibitions respectively [18]. For example, the sentence "*this exercise tracking app must not ask for my contact details"* indicates that the user is strongly against the app requiring contact details. These deontic terms may be useful in determining the urgency with which features should be fixed. We constructed a dictionary containing the modal verb phrases for prohibitions and obligations respectively. The number of times

features were mentioned along with a deontic term (prohibitions, obligations) in a review sentence was computed. The feature with the higher count of deontic terms would obtain a higher priority.

## 2.3 Three prioritization approaches

We review our three prioritization approaches in this subsection.

### *2.3.1 Individual attribute-based approach*

In this approach, each attribute is examined individually. For example, when ranking features based on frequency count, the ratings are not considered. This approach may be useful where developers are interested in only particular attributes, such as the ratings. This silo-approach would enable developers to obtain a quick overview of the priorities based on how users perceive their various app features based on a specific criteria (e.g., rating), without needing to look at the other attributes. The usefulness of such an approach has been demonstrated using the frequency approach in app-review mining and other relevant domains [13, 14]. However, the limitation of this approach is that it forgoes potential insights that could be obtained through an integrated approach involving specific combination of attributes. This is the main focus of the weighted approach, introduced next.

### *2.3.2 Weighted approach*

This approach enables the consideration of two or more attributes in the prioritization process by allowing them to be combined. The weighted approach that we propose here is based on prior research in Multi Criteria Analysis (MCA) [23] that is used to find the optimal choice between different alternatives. A method of MCA is the Weighted Sum Model. In this approach, the values for each criterion (e.g., rating and frequency) must first be normalized so they become comparable. Weights are then assigned to each criterion to show their relative importance. These weights are multiplied by the values of the corresponding attribute, and then added together to determine the weighted model score. These scores are then used to rank the different alternatives considered.

All four attributes (frequency, ratings, negative emotions and deontics) can be combined, to produce the priority for feature *i* using the formula

$$\mathbf{P(i) = C1 * F(i) + C2 * R(i) + C3 * E(i) + C4 * D(i)} \qquad (1)$$

where, F(i) is the normalized frequency of the chosen feature, R(i) is the normalized average rating for the feature, E(i) is the normalized emotion count for the feature, and D(i) is the normalized deontics count for the feature. C1 to C4 are the weights associated with the attributes where the sum of the weights is 1. The attribute considered to be of higher importance would receive a higher weighting. This enables the priority value for all features to be computed, based on which the features may be ranked.

The values of each of the four attributes are normalized to a scale of 0 to 1 by applying the well-known formula given in Equation 2, before they are used in Equation 1. The normalized score of a feature *i* is given by Norm (i), where $x_i$ represents the attribute score of feature *i* (i.e., the feature under consideration), $x_{max}$ and $x_{min}$ correspond to the maximum and minimum scores among all features considered (for a given attribute).

$$\mathbf{Norm\ (i) = (x_i - x_{min}) / (x_{max} - x_{min})} \qquad (2)$$

It should be noted that the approach can be customized by practitioners depending on their experience. For example, if they consider that frequency and rating are the two important attributes to consider then C3 and C4 can be assigned a value of zero. Also, it should be noted that the formula in Equation 1 provides the ability for a human-user to specify weights for each attribute considered based on which attribute may be more important to a developer (or a development team). These weights could come from domain knowledge, and/or the experience of the team. The inclusion of this human aspect may be beneficial in the rapidly changing app domain as humans may have insights derived from a range of apps which can be used to assign weights (instead of considering just one app). Thus, this approach is a blend of data-driven bottom up approach (i.e., use of values obtained for attributes from the reviews) and a top-down approach (i.e., specification of weights for individual attributes based on prior knowledge).

### *2.3.3 Regression-based approach*

The third approach is a pure data-driven approach where the investigation aimed at examining influential variables for determining the severity of reviews, and thus, urgency with which a feature should be fixed. We employed multiple regressions where the regression equation used for the prediction of a dependent variable (Y) is given by Equation 3. Here, b0 is a constant, b1 is the coefficient of the first predictor variable (x1) and bn is the coefficient of the last predicator variable (xn).

$$\mathbf{Y = b0 + (b1 * x1) + (b2 * x2) + \ldots + (bn * xn)} \qquad (3)$$

Using the regression approach, we pursued two small studies. In both the studies, there were a total of 25 independent variables. These were: (a) top-20 feature counts (i.e., top-20 features based on frequency attribute), (b) emotion category counts (three forms of emotions – sadness, anger and fear) and (c) deontic category counts (two such categories – obligations and prohibitions).

In the first study, the dependent variable was the rating. The objective here was to investigate whether a model that can predict the rating can be constructed based on the 25 independent variables. If such a model can be constructed, then the correlation coefficients of the statistically significant variables (e.g., b1 and b2 in Equation 3) can be used as the basis for prioritization. The higher the coefficient for a particular statistically significant variable (i.e., a feature), the higher is its influence on the dependent variable. Hence, such a feature would have a higher priority for fixing. In the second study we changed the dependent variable. Instead of rating, we used a tag that was assigned by independent raters, where the tag represented the severity (or urgency) of fixing a problematic feature from a scale of 0 to 3. A tag of '0' was given to sentences where valuable or actionable information (i.e., features with issues) was not identified. A tag of '1' was given to sentences that contained suggestions for enhancement, a tag of '2' for problems with the functioning and performance of the core features of the app, and a tag of '3' for severe problems where an app feature was unusable. Based on these guidelines, independent coders (PhD students in computing) familiar with app development were asked to examine each of the negative sentences, and rate the severity of the problem, from the perspective of an app developer.

Note that the first study does not include any additional information from the humans (i.e., all the variables are inferred from the app review dataset). In contrast, the second study brings human into the loop to assign tags indicating the severity of a feature that needs to be fixed (but not for prioritizing features). The two analyses were undertaken to scrutinize if there is a difference in the regression analyses of the human-coded evaluation (study 2) and the data-centric evaluation (study 1). The result of this analysis can inform the suitability of these two types of regression approaches for the prioritization process. We examine the suitability of this and the other two approaches in our evaluation presented next.

# 3. EVALUATION AND ASSESSMENT

This section presents the results obtained from the three prioritization approaches for the MyTracks app. MyTracks is an exercise tracking app available from the Google Play Store, which records the user's path, speed, distance, and elevation while they walk, run, and bike. While recording, users can view their data live, annotate their paths, and hear voice announcements of their progress. The app also allows users to sync and share their tracks with friends via Google Drive, and social media sites, such as Facebook. It also integrates data from heart rate monitors developed by certain external vendors. The results of the three prioritization approaches for 1,271 negative sentences (obtained using the process discussed in Section 2) are presented in the following subsections.

## 3.1 Individual attribute-based ranking

Given space restrictions, the top-10 features based on the frequency-based ranking and emotion-based ranking approaches are shown in Table 1 (out of 57 features identified for the app). From the frequency based ranks (columns 1-2), it can be observed that the feature *gps* was found to be the most mentioned feature within the negative review sentences, with a count of 169. The feature *track* was a close second, with 168. This is intuitive as it is an exercise app that uses GPS to track users' fitness. Users who have problems with the main features that the app claims to provide would voice their concerns when their expectations are violated. This can be seen in reviews such as "*the application keeps hanging so the track is lost*". *Map*, which has a count of 109, was the feature ranked next. The obvious issues with this are problems with the mapping of their tracks, as shown by "*maps usually not appear on my track*". However, these instances may have also arisen due to issues related to the integration with Google Maps, which is a functionality that the app advertises. The next two features correspond to the tracking of the *time* taken to complete a workout and the nature of the *data* recorded. The common problems included the inaccuracy of the time being recorded and the data being lost or inaccurate.

**Table 1. Prioritization using frequency and emotion attributes**

| rank | Frequency | | Emotion | |
|---|---|---|---|---|
| | feature | count | feature | count |
| 1 | gps | 169 | gps | 240 |
| 2 | track | 168 | track | 228 |
| 3 | map | 109 | map | 162 |
| 4 | time | 79 | time | 103 |
| 5 | data | 58 | data | 76 |
| 6 | battery | 51 | signal | 70 |
| 7 | file | 50 | file | 64 |
| 8 | signal | 49 | package | 63 |
| 9 | package | 48 | battery | 61 |
| 10 | distance | 41 | speed | 52 |

Emotion-based ranking tells a slightly different story (columns 4 and 5 of Table 1). While the top-five features retain the same ranking as frequency, the rank-orders of the other features are different. Also, the feature *distance* does not appear in the emotion-based ranking which is replaced by *speed* in this ranking. An implication of using this silo-approach is that, if prioritization is based on a different attribute, the same team will prioritize features differently. We posit that this approach will be beneficial for a team to compare the ranks from different attributes to obtain an understanding of the top-problematic features that are consistent across all attribute-based ranking schemes. For example, if the top-5 features appear to be consistent in all these ranking mechanisms, the team can confidently go ahead with improving those features for the next iteration.

## 3.2 Weighted ranking

To demonstrate the operationalization of the weighted approach, we consider the use of frequency and rating attributes (instead of all four attributes shown in Equation 1). Table 2 shows the results involving the two attributes with three different weight combinations. Columns 3, 5 and 7 show the priority values for three different combinations of weights for C1 and C2 [0.5 & 0.5, 0.7 & 0.3, 0.3 & 0.7]. When both frequency and rating have the same weights, *track*, *gps*, and *map* appear to be the top three ranked features. This is also the case when frequency has a higher weight than rating. However, when rating is given more importance, *design* and *sharing* are among the top three ranked features, while *map* ranks 6th.

**Table 2. Ranks for weighted approach (frequency and rating)**

| rank | P= (0.5*F)+(0.5*R) | | P=(0.7*F)+(0.3*R) | | P=(0.3*F)+(0.7*R) | |
|---|---|---|---|---|---|---|
| | feature | priority | feature | priority | feature | priority |
| 1 | track | 0.72 | track | 0.83 | design | 0.70 |
| 2 | gps | 0.70 | gps | 0.82 | sharing | 0.62 |
| 3 | map | 0.58 | map | 0.60 | track | 0.61 |
| 4 | design | 0.50 | time | 0.46 | gps | 0.58 |
| 5 | sharing | 0.46 | data | 0.37 | export | 0.56 |
| 6 | time | 0.45 | speed | 0.34 | map | 0.55 |
| 7 | export | 0.41 | battery | 0.33 | calorie | 0.51 |
| 8 | speed | 0.40 | distance | 0.32 | kml | 0.50 |
| 9 | route | 0.39 | file | 0.32 | route | 0.50 |
| 10 | calorie | 0.38 | signal | 0.31 | file | 0.50 |

There were a total of 16 unique features that appeared in the top-10 lists (in Table 2). Out of the 16, 3 appeared in all three (*track*, *gps* and *map*), 8 appeared in two lists and 5 appeared in only one of the three lists. It is interesting to note that the results for weighting scheme 2 (C1=0.7 and C2=0.3), shown in column 4 of Table 2, is similar to the results shown in column 2 of Table 1. When comparing the two columns, the ranks for the first two features are reversed, but otherwise the other features in the top-5 are consistent. Only one of the 10 features differs (*speed* appears in Table 2 vs. *package* in Table 1). However, the ranks for the features in the bottom half of the lists are different. These results show that the weights chosen will significantly impact the features that are selected for prioritization (i.e., financial gains or market share could be foregone by not choosing the right weights).

## 3.3 Regression-based ranking

While the weighted approach involves experts knowledgeable in the domain of investigation, which in many cases may be available within an organization, and has also been demonstrated in other examples in software engineering (e.g., customizing and fine-tuning COCOMO models [24] based on organizational data and experience), we now investigate if important features can be inferred purely based on a data-driven approach. We examine the outcomes of the two types of regression analyses conducted next.

The first study used the rating provided by the app-reviewers as the dependent variable. The multiple regression conducted produced a regression model with six significant variables out of 25 ($p<0.05$): *route*, *widget*, *package*, *speed*, *map* and *anger*. Despite returning a significant model however, the six variables only accounted for 3% of the variation in rating.

In the second study, tags were assigned by independent coders who examined the review sentences from the viewpoint of

developers (as discussed above). For the MyTracks app, there were four coders who coded the 1,271 sentences (a four-way split). The interrater reliability (based on 100 sentences coded) for the study using the intraclass correlation coefficient (ICC) [22] metric was 0.84 (indicating 84% agreement). From the regression analysis with tags as the dependent variable, a significant model emerged ($F_{22.48}$ = 20.04 with $p < 0.01$). The Adjusted R squared value accounted for 20% of the variance in the urgency to fix certain features (compared to 3% for the first study). The 10 statistically significant features (out of 25) from this analysis are shown in Table 3.

**Table 3. Significant features from regression analysis**

| Rank | Feature | Coefficient | Rank | Feature | Coefficient |
|---|---|---|---|---|---|
| 1 | package | 1.17* | 6 | gps | 0.61* |
| 2 | battery | 0.91* | 7 | accuracy | 0.46** |
| 3 | download | 0.69* | 8 | file | 0.46* |
| 4 | elevation | 0.67* | 9 | speed | 0.42* |
| 5 | distance | 0.62* | 10 | track | 0.39** |
| Note: ** = p<0.01, *=p<0.05 | | | | | |

The magnitudes of the coefficients of the statistically significant variables indicate the priority for fixing that feature. The higher the coefficient, the higher is its impact in influencing the dependent variable (i.e., tag). The result suggests that the feature *package* should be given the top-priority. This is because one unit increase in the package variable increases 1.17 units of the tag variable. This suggests those sentences that had high tag values (say 3, which indicates high severity) had more instances of *package* appearing in them as compared to those that were tagged lower (say 0 or 1). Hence, *package* is ranked the highest. Our manual verification revealed that the majority of reviews that contained the word 'package' were complaining that the app had indeed become unusable. In the frequency-based ranking of features (Table 1), *package* was ranked 9th, with a count of 48. This suggests that *package* may be more influential in improving the outcome of the review than can be observed just by looking at the frequency. So, the regression analysis has revealed insights that may not be obvious. This pattern may also be observed when looking at the features *gps*, *track*, and *map*. These appeared to be the most problematic features from the frequency-based ranking. However, the regression results suggest there are other more problematic features such as *package*, *battery* and *download*.

Looking at the features that exist in both the top-ranked features from frequency-based ranking and this regression outcome (Table 1 and Table 3), *package*, *battery*, *distance*, *gps*, *file*, and *track* (a total of 5 features) exist in both. This suggests that the individual attribute-based ranking that considers the frequency attribute is closely aligned with the regression results using tag as the dependent variable. Additionally the results using tag as the dependent variable is closer to the weighted approach (with 7 matches out of 10 – but with different rank-orders) where the values of C1 and C2 for the frequency and rating attributes are 0.7 and 0.3 respectively, than the other two weight-combinations considered (on comparing Table 3 and Table 2). This shows that the weighted approach is closer to the regression-based approach than just the frequency based approach.

# 4. DISCUSSION

The research question addressed in this work is: ***How can we use data-driven approaches to prioritize features for improvements extracted from user reviews?*** Based on inspirations from the literature on app-review mining, psychology and sociology this paper identified a set of four attributes – frequency, rating, emotions and deontics. These were used to construct three prioritization approaches (individual attribute-based approach, weighted approach and regression approach). Using MyTracks as a case study, it was shown that the three approaches can be used to prioritize feature improvements. However, there are relative strengths and weaknesses of these approaches and these are discussed below. The subsequent subsections highlight the implications of our outcomes, and the limitations and future work.

## 4.1 Comparison of the three approaches

The individual attribute-based approach is a silo prioritization approach (or subset of the weighted approach) where a single attribute is considered for prioritization. The approach can be used in two ways. First, it enables developers to prioritize features based on what they consider to be the most important attribute (similar to what has been adopted elsewhere [13, 14]). Second, by enabling the possibility of investigating all the four silo-rankings for the same features it facilitates the comparison of rankings. This could enable developers to make informed decisions about which features to prioritize (e.g., choosing features that appear in all the four ranking schemes). So, we propose the use of this approach for exploratory purposes, since it may miss potential insights that can be revealed by combining different attributes.

The weighted approach allows different attributes to be considered (in various combinations) that are deemed significant by development teams. As certain attributes of reviews may be more important to developers than others when prioritizing features for improvements, practitioners need an approach that can serve as a 'what-if' scenario analyzer. This is achieved by considering only those attributes that are considered to be relevant (in Equation 1) and also adjusting the weights of attributes. It should be noted that the approach values human expertise and experience by soliciting weights of different attributes that are combined. As development teams become more experienced (e.g., through domain knowledge gained in building several apps in the same genre, say games), we posit that the assignment of weights should become less cumbersome. For example, the success of choosing a particular weight in the prioritization of feature improvements for an early iteration may be used to determine the weights for subsequent iteration(s).

Though the weighted approach can be beneficial for development firms with expertise in the app domain, startups and small app development firms may lack such knowledge, and a pure data-driven approach might be useful for them. We suggest the use of the regression based prediction approach as a bootstrap mechanism to compensate for the lack of expertise. The use of this approach over time can be used to identify the features that tend to routinely run into problems and mitigation strategies could be planned. Another advantage of the data-driven approach is that it might provide specific 'micro-insights'. Often, knowledge within the development firm is based on common trends observed over time (i.e., macro-insights). However, emerging micro-trends (i.e., new problematic features) may not have yet become the common knowledge, and hence, the weighted or attribute-based approaches may not be suitable to infer those. For example, certain features in a released version of an app can quickly run into problems causing discontent among users. These should thus be swiftly identified and prioritized for improvements. We indeed observed this issue for MyTracks latest release. Some users complained about the app not working after upgrading to Android 5.1 Lollipop. Also, some users complained about a specific brand of heart rate monitor (Polar H7 monitor) not working with the latest version of the app despite the support indicated. These users promptly provided

poor ratings (rating =< 3). Since these two cases may have affected only a small proportion of users, they do not appear in the top-10 frequency-based list. However, these would be identified more easily by the data-driven approach when compared to the other two approaches, since the regression approach identifies priorities based on correlations. Thus, the regression approach appears to possess the most promise of the three. Having said that, we have shown the result using the weighted approach is in fact closer to that of the regression approach (refer to Sections 3.2 and 3.3). However, there is a requirement to select the 'correct' weights based on expertise.

## 4.2 Implications

This work contributes to research efforts aimed at enabling requirements prioritization in the app development domain by proposing three approaches. These approaches provide a layer of abstraction that hides low-level details from the developers, where they do not need to look at app reviews directly to identify app features for improvements. Instead, they may use the proposed approaches to determine which aspects of their app to enhance or fix next. More broadly, our approaches can complement those that identify features from reviews (e.g., Android OS reviews) [6, 16], in going one step further to help with prioritization.

These approaches could be of utility to practitioners. The three approaches presented above are complimentary in nature. Developers may use our approaches as part of a two-stage prioritization process. The individual attribute-based approach may be used for exploratory investigations of troublesome features, followed by either the weighted approach (contingent to prior experience) or a data-driven regression approach (subject to lack of experience). The latter approach is also useful for bootstrapping feature prioritization, and identifying micro-trends of features that need urgent attention.

## 4.3 Limitations and future work

Our study considered reviews which had a rating of =< 3, in line with previous studies [13]. However, there could be some sentences that contain problematic features in reviews with ratings of 4 or 5, which may have been missed, a threat to the validity of our approach. Our evaluation results have been presented for one case study, which affects the work's generalizability. We intend to conduct further evaluations in different app domains (e.g., games and health apps) to examine the domain-specific insights about features that need to prioritized for improvements.

We intend to experiment with machine learning algorithms (e.g., Conditional Random Fields [25]), for automatically tagging large amounts of reviews. Such approaches would require developers to tag only a small subset of reviews, which will then be used as input to automated review tagging. Furthermore, there is scope to create a toolset for extracting reviews from the Google Play Store, and parsing these to prioritize features for improvements. We will also consider issues associated with features during the prioritization process (e.g. *time* being *inaccurate*).

# 5. REFERENCES


[1] Brooks, F. No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20, 4 (1987), 10-19.

[2] Hosseini, M., Phalp, K. T., Taylor, J. and Ali, R. Towards Crowdsourcing for Requirements Engineering (2014).

[3] Chen, N., Lin, J., Hoi, S. C., Xiao, X. and Zhang, B. *AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace*. In Proc. of ICSE, 2014, 767-778.

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

[5] Iacob, C. and Harrison, R. *Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews*. In Proc. of MSR, 2013, 41-44.

[6] Maalej, W. and Nabil, H. *Bug report, feature request, or simply praise? on automatically classifying app reviews.* In RE, 2015, 116-125.

[7] Achimugu, P., Selamat, A., Ibrahim, R. and Mahrin, M. N. A Systematic Literature Review of Software Requirements Prioritization Research. *Information and Software Technology*, 56, 6 (2014), 568-585.

[8] Laurent, P., Cleland-Huang, J. and Duan, C. *Towards automated requirements triage.* In Proc. of RE, 2007, 131-140.

[9] Avesani, P., Bazzanella, C., Perini, A. and Susi, A. *Facing scalability issues in requirements prioritization with machine learning techniques.* In Proc. of RE, 2005, 297-305.

[10] Beg, R., Abbas, Q. and Verma, R. P. *An approach for requirement prioritization using b-tree*. In Proc. of ICETET, 2008, 1216-1221.

[11] Moretti, A. and Tuan, A. The social media manager as a reputation's gatekeeper: an analysis from the new institutional theory perspective. *ISSN 2045-810X* (2015), 153.

[12] Patel, P., Licorish, S., Savarimuthu, B. T. R. and MacDonell, S. Studying expectation violations in socio-technical systems - A case study of the mobile app community, To appear in Proc. of *ECIS*, 2016.

[13] Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J. and Sadeh, N. *Why people hate your app: Making sense of user feedback in a mobile app store*. In Proc. of SIGKDD, 2013, 1276-1284.

[14] Guzman, E. and Maalej, W. *How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews.* In RE, 2014, 164-172.

[15] Suen, C. Y. N-Gram Statistics for Natural Language Understanding and Text Processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions* 2(1979), 164-172.

[16] Lee, C. W., Licorish, S., Savarimuthu, B. T. R., MacDonell, S. and Patel, P. They'll Know It When They See It: Analyzing Post-Release Feedback from the Android Community (2015), In AMCIS, 2015, 1-11.

[17] Martin, W., Sarro, F., Jia, Y., Zhang, Y. and Harman, M. A Survey of App Store Analysis for Software Engineering. *RN*, 16 (2016), 02.

[18] Frantz, C., Purvis, M. K., Nowostawski, M. and Savarimuthu, B. T. R. *Modelling Institutions Using Dynamic Deontics*. In Proc. of PRIMA, 2014, 153-162.

[19] Parrott, W. G. *Emotions in Social Psychology: Essential Readings.* Psychology Press, 2001.

[20] Pennebaker, J. W., Francis, M. E. and Booth, R. J. Linguistic Inquiry and Word Count. *Mahway: Lawrence Erlbaum Associates*, 71 (2001).

[21] Wang, W., Chen, L., Thirunarayan, K. and Sheth, A. P. *Harnessing Twitter "Big Data" for Automatic Emotion Identification*. In Proc. of PASSAT, 2012, 587-592.

[22] Crawford, S. E. and Ostrom, E. A grammar of institutions. *American Political Science Review*, 89, 03 (1995), 582-600.

[23] Triantaphyllou, E. *Multi-Criteria Decision Making Methods.* Springer, 2000.

[24] Clark, B., Devnani-Chulani, S. and Boehm, B. *Calibrating the COCOMO II post-architecture model*. In Proc. of ICSE, 1998, 477-480.

[25] Lafferty, J., McCallum, A. and Pereira, F. C. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data (2001).


View publication stats