Opinion Mining for App Reviews: Identifying and Prioritizing Emerging Issues for Software Maintenance and Evolution

Vitor Mesaque Alves de Lima
Faculty of Computing (FACOM), Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, MS, Brazil
vitor.lima@ufms.br

Ricardo Marcodes Marcacini
Institute of Mathematics and Computer Sciences (ICMC), University of São Paulo (USP)
São Carlos, SP, Brazil
ricardo.marcacini@usp.br

ABSTRACT

Opinion mining for app reviews aims to analyze user comments on app stores to support software engineering activities, primarily software maintenance and evolution. One of the main challenges in maintaining software quality is promptly identifying emerging issues, such as bugs. However, manually analyzing these comments is challenging due to the large amount of textual data. Methods based on machine learning have been employed to automate opinion mining and address this issue. **Gap.** While recent methods have achieved promising results in extracting and categorizing issues from users' opinions, existing studies mainly focus on assisting software engineers in exploring users' historical behavior regarding app functionalities and do not explore mechanisms for trend detection and risk classification of emerging issues. Furthermore, these studies do not cover the entire issue analysis process through an unsupervised approach. **Contribution.** This work advances state of the art in opinion mining for app reviews by proposing an entire automated issue analysis approach to identify, prioritize, and monitor the risk of emerging issues. Our proposal introduces a two-fold approach that (i) identifies possible defective software requirements and trains predictive models for anticipating requirements with a higher probability of negative evaluation and (ii) detect issues in reviews, classifies them in a risk matrix with prioritization levels, and monitors their evolution over time. Additionally, we present a risk matrix construction approach from app reviews using the recent Large Language Models (LLMs). We introduce an analytical data exploration tool that allows engineers to browse the risk matrix, time series, heat map, issue tree, alerts, and notifications. Our goal is to minimize the time between the occurrence of an issue and its correction, enabling the quick identification of problems. **Results.** We processed over 6.6 million reviews across 20 domains to evaluate our proposal, identifying and ranking the risks associated with nearly 270,000 issues. The results demonstrate the competitiveness of our unsupervised approach compared to existing supervised models. **Conclusions.** We have proven that opinions extracted from user reviews provide crucial insights into app issues and risks and can be identified early to mitigate their impact. Our opinion mining process implements an entire automated issue analysis with risk-based prioritization and temporal monitoring.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; • **Computing methodologies → Neural networks**; • **Information systems → Reputation systems**.

KEYWORDS

opinion mining, app reviews, issue detection, issue prioritization

ACM Reference Format:
Vitor Mesaque Alves de Lima and Ricardo Marcodes Marcacini. 2023. Opinion Mining for App Reviews: Identifying and Prioritizing Emerging Issues for Software Maintenance and Evolution. In *XXIII Brazilian Software Quality Symposium (SBQS 2024), November 5–8, 2023, Salvador, Brazil.* Campo Grande, MS, Brazil, 10 pages. https://doi.org/10.1145/3613372.3613417

1 PROBLEM OUTLINE

Opinions extracted from informative end-user reviews provide a wide range of user feedback to support software engineering activities, such as bug report classification, new feature requests, usage experience, or enhancements (i.e., suggestions for improvements) [33, 2, 9, 3, 32]. However, mobile application (app) developers spend exhaustive manual efforts identifying and prioritizing informative end-user reviews. Manually analyzing a reviews dataset to extract helpful knowledge from the opinions is challenging because of the large amount of data and the high frequency of new reviews published by users [33]. Therefore, to deal with these challenges, opinion mining has been increasingly used for computational analysis of people's opinions from textual data [29]. Furthermore, in the context of app reviews, opinion mining allows extracting excerpts from comments and mapping them to emerging issues according to the users' experience [25].

In the context of mobile apps, [34] has conducted research demonstrating that the most commonly occurring update in app stores is bug fixing, accounting for a substantial 63% of updates. As a result, approaches that automate the analysis of concerns from app reviews are critical for strategic updates and the prioritization and planning of new releases [24]. Furthermore, app stores provide a more dynamic method of directly distributing software to customers, with shorter release times than traditional software systems, i.e., continuous update releases are performed every few weeks or even days [38]. In this context, app reviews provide immediate crowd feedback about software misbehavior or bad user experience that may not be replicated during routine development/testing processes,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SBQS 2024, September 25–29, 2023, Salvador, Bahia
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0787-2/23/09…$15.00
https://doi.org/10.1145/3613372.3613417

such as device combinations, screen sizes, operating systems, and network conditions [42].

App developers can use this ongoing community feedback in developing preventive maintenance processes. Given this, we argue that software engineers would employ an opinion-mining approach to investigate bugs, misbehavior, and bad user experience early when an app receives negative reviews. Opinion mining techniques can organize reviews based on the software features and their associated user's sentiment [9, 4, 25]. Consequently, developers can investigate issues to comprehend the user's concerns about a faulty feature or compromised user experience and potentially fix or improve it more quickly, i.e., before impacting many users and negatively affecting the app's ratings [25]. Given this dynamic environment and a large amount of data, the problem of detecting and prioritizing issues from reviews is crucial and remains for practitioners and researchers [24, 32].

Different strategies have been recently proposed to support issue detection and classification [51], such as issue categorization [21, 15, 41, 35, 23, 8, 17, 18, 30, 48, 39, 40, 4, 20, 36, 1, 16] and prioritization [24, 32]. Although previous studies are promising, they do not cover the entire process of issue analysis from start to finish, from the automated collection of reviews to the tasks of detection, prioritization, and analysis using an unsupervised approach.

This work introduces an innovative two-fold approach aimed at investigating emerging issues derived from user feedback. Firstly, it entails identifying potential defective software requirements and developing predictive models to anticipate issues that may lead to negative app evaluations. Secondly, it involves detecting issues within reviews and categorizing them using a risk matrix that assigns prioritization levels, thereby enabling the monitoring of their evolution over time. We also present the construction of a risk matrix from app reviews using recent Large Language Models (LLMs). By utilizing Open Pre-trained Transformers (OPT), our approach facilitates the use of LLMs in scenarios with limited computational resources and data privacy constraints. Our approach enables us to effectively address reviews on time, mitigate negative impacts on the overall app rating, and maintain the app's competitiveness, ensuring timely maintenance and facilitating software evolution. We introduce an analytical data exploration tool with an interactive dashboard and a real-time issue monitor.

1.1 Objectives and Research Questions

Our main objective is to address the challenges faced by developers in efficiently analyzing informative end-user reviews, applying opinion mining to automate the process of identifying and prioritizing emerging app issues, and enabling proactive software quality maintenance through timely issue fixing, user-centric improvements, and minimizing the time between issue occurrence and correction. We aim to ensure prompt issue identification and resolution by achieving these goals, facilitating timely software maintenance and evolution.

In this sense, we raise the followings research questions:

- ***RQ1*** *How do we predict initial trends on defective requirements from users' opinions before negatively impacting the overall app's evaluation?*
- ***RQ2*** *How do we prioritize and address app issues from reviews in time so that the app is competitive and guarantees the timely maintenance and evolution of the software?*

RQ1 was mainly addressed by the investigations and solutions reported in Section 2, while RQ2 was mainly addressed in Sections 3 and 4 of this work.

To enhance understanding, we've organized the research timeline to ensure a logical progression of proposals throughout the work. The proposed methods and approaches build upon each other, with each section presenting advancements and improvements over the previous ones. This sequential organization strengthens the coherence and comprehensiveness of our research. As illustrated in Figure 1, the timeline provides an overview of the sections' distribution and their interconnected progression.

Figure 1: Research timeline

- **MApp-Reviews (Section 2):** Introduces the temporal dynamics of requirements engineering using mobile app reviews. Here, we describe the MApp-Reviews method, present its architecture, and discuss the key findings and results. Section 2 describes the initial direction of our research.
- **MApp-IDEA (Section 3):** Introduces the MApp-IDEA method and experimentally evaluates it to derive outcomes and findings. In this point, the status of the research is a 2-fold approach, in which the stages of both methods can be explored to combine a third instantiation of the opinion mining approach. However, the approach presented in Section 3 is superior and brings more technical advancements and results. In practice, MApp-IDEA represents an evolution of MApp-Reviews.
- **Risk Matrix using LLM (Section 4):** Investigates how recent Large Language Models such as GPT and OPT can be leveraged to facilitate the automatic construction of risk matrices from app reviews. This investigation encompasses various stages, from extracting review features to classifying them into priority levels.

2 TEMPORAL DYNAMICS OF REQUIREMENTS ENGINEERING FROM MOBILE APP REVIEWS

To explore the temporal dynamics of software requirements extracted from app reviews, we present the MAPP-Reviews (Monitoring App Reviews) method. First, we collect, pre-process, and extract software requirements from large review datasets. Then, the software requirements associated with negative reviews are organized

into groups according to their content similarity by using a clustering technique. The temporal dynamics of each requirement group are modeled using a time series, which indicates the time frequency of a software requirement from negative reviews. Finally, we train predictive models on historical time series to forecast future points. Forecasting is interpreted as signals to identify which requirements may negatively impact the app in the future, e.g., identify signs of app misbehavior before impacting many users and prevent low app ratings. Figure 2 presents the MApp-Reviews method architecture.

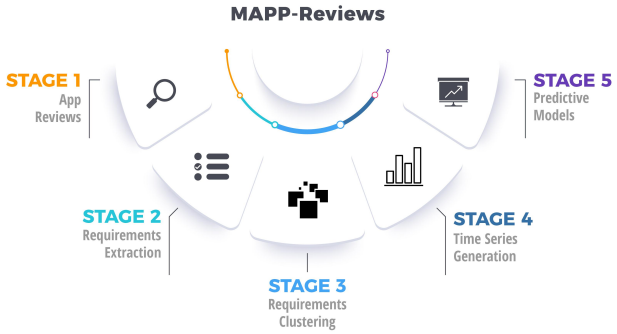

Figure 2: Overview of the proposed method for analyzing temporal dynamics of requirements engineering from mobile app reviews

2.1 Requirements Extraction

MAPP-Reviews uses the pre-trained RE-BERT [3] model to extract software requirements from app reviews. The RE-BERT model was trained using a labeled reviews dataset generated with a manual annotation process, as described by [9]. RE-BERT uses a cross-domain training strategy, where the model was trained in 7 apps and tested in one unknown app for the test step. RE-BERT software requirements extraction performance was compared to SAFE [22], ReUS [14] and GuMa [19]. Since RE-BERT uses pre-trained models for semantic representation of texts, the extraction performance is significantly superior to the rule-based methods. Given this scenario, we selected RE-BERT for the requirement extraction stage. Figure 3 shows an example of review and extracted software requirements. In the raw review *"I am ordering with delivery but it is automatically placing order with pick-up"*, four software requirements were extracted ("ordering", "delivery", "placing order", and "pick-up"). Note that "placing order" and "ordering" are the same requirement in practice. In the clustering step of the MAPP-Reviews method, these requirements are grouped in the same cluster, as they refer to the same feature.

2.2 Requirements Clustering

After mapping the software requirements into word embeddings, MAPP-Reviews uses a technique to obtain a clustering model of semantically similar software requirements. For this task, we use the K-means clustering algorithm [31]. At this point in the MAPP-Reviews method, we have software requirements pre-processed and represented through contextual word embeddings, as well as an organization of software requirements into k clusters.

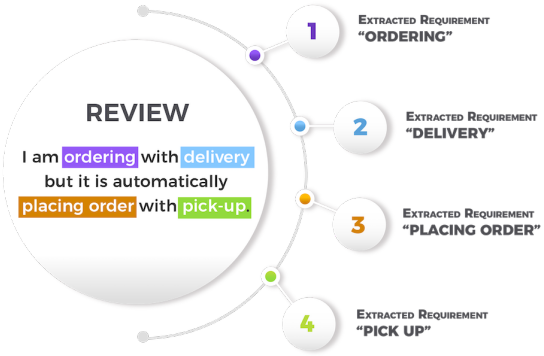

Figure 3: Example of a review and extracted requirements

2.3 Time Series Generation

Time series can be described as an ordered sequence of observations [7]. A time series of size s is defined as $X = (x_1, x_2, ..., x_s)$ in which $x_t \in \mathbb{R}$ represents an observation at time t.

MAPP-Reviews generates time series for each software requirements cluster, where the observations represent how many times each requirement occurred in a period. Consequently, we know how many times a specific requirement was mentioned in the app reviews for each period. Each series models the temporal dynamics of a software requirement, i.e., the temporal evolution considering occurrences in negative reviews.

2.4 Predictive Models

Predictive models for time series are very useful to support an organization in its planning and decision-making. Given a confidence interval, such models explore past observations to estimate observations in future horizons. In our MAPP-Reviews method, we aim to detect the negative reviews of software requirements that are starting to happen and make a forecast to see if they will become serious in the subsequent periods, i.e., a high frequency of negative reviews. The general idea is to use p points from the time series to estimate the next $p + h$ points, where h is the prediction horizon.

MAPP-Reviews uses the Prophet Forecasting Model [47]. Prophet is a model from Facebook researchers for forecasting time series data considering non-linear trends at different time intervals, such as yearly, weekly, and daily seasonality. We chose the Prophet model for the MAPP-Reviews method due to the ability to incorporate domain knowledge into the predictive model.

2.5 Results and Discussion

The experimental evaluation was carried out to verify whether it is possible to detect emerging issues days or weeks in advance to mitigate the impact of negative ratings on the overall evaluation of the app. In this sense, our experiment models the temporal dynamics of software requirements associated with negative user reviews to predict upward complaints trends. Furthermore, we demonstrate that it is possible to use a predictive model with satisfactory accuracy to predict the temporal dynamics of a software requirement.

The method starts by gathering negative app reviews and extracting software requirements using the RE-BERT tool. This yields

a time series depicting the evolution of each requirement over time. Due to the difficulty in predicting future bug reports accurately, predictions are made on a weekly basis instead of monthly. Models are trained with previous points for each series and used to forecast the next four weeks. Prediction errors are evaluated using the MAPE metric, and significant increases in the time series are identified. The Prophet model is employed for forecasting, incorporating both automatic and custom changepoints. Custom changepoints are determined based on training set predictions exceeding the mean plus standard deviation of all values, aiding in capturing significant deviations and trends.

Conducted an experimental evaluation involving approximately 86,000 reviews over 2.5 years for three food delivery apps. The experimental results show that it is possible to find significant points in the time series that can provide information about the future behavior of the requirement through app reviews, indicating the potential impact of incorporating changepoints into the predictive model using the information of app developers. Furthermore, our method can provide important information to software engineers regarding software development and maintenance, enabling the model to predict the peaks of negative reviews for the software requirement one week in advance. Therefore, software engineers can act proactively through the proposed MAPP-Reviews approach, reducing the impacts of a defective requirement. More details are in [25].

Our main contributions of the MApp-Reviews method are summarized below:

(1) **Clustering Software Requirement Variations:** Although there are promising methods for extracting candidate software requirements from application reviews, such methods do not consider that users describe the same software requirement in different ways with non-technical and informal language. Our MAPP-Reviews method introduces software requirements clustering to standardize different software requirement writing variations. In this case, we explore contextual word embeddings for software requirements representation, which have recently been proposed to support natural language processing. When considering the clustering structure, we can more accurately quantify the number of negative user mentions of a software requirement over time.
(2) **Temporal Dynamics of Software Requirements:** Using time series, we present an approach to generate the temporal dynamics of negative ratings of a software requirements cluster. Our method uses equal-interval segmentation to calculate the frequency of software requirements mentioned in each time interval. Thus, a time series is obtained and used to analyze and visualize the temporal dynamics of the cluster, where we are especially interested in intervals where sudden changes happen.
(3) **Incorporating Domain-specific Data into Forecasting:** Time series forecasting is useful to identify in advance an upward trend of negative reviews for a given software requirement. However, most existing forecasting models do not consider domain-specific information that affects user behavior, such as holidays, new app releases and updates, marketing campaigns, and other external events. In the MAPP-Reviews method, we investigate the incorporation of software domain-specific information through trend changepoints. We explore both automatic and manual changepoint estimation.

3 ISSUE DETECTION AND PRIORITIZATION BASED ON APP REVIEWS

To detect and prioritize emerging app issues, we introduce MApp-IDEA (Monitoring App for Issue Detection and Prioritization) method, which explores word embedding techniques to construct acyclic graphs performing as representations of app-related issues. This novel method is designed to promote the timely identification and prioritization of emergent issues and potential risks involving app features, environment, and user experience. Furthermore, the MApp-IDEA performs in real-time, enhancing its practical utility and responsiveness. We have trained a multilingual BERT-based model with more than 100.000 nodes. Our proposal is an unsupervised approach, while promising methods for detecting issues from app reviews use supervised approaches.

3.1 MApp-IDEA Method Architecture

Our method is divided into five stages, as shown in Figure 10. First, we collect mobile app reviews from app stores via a web crawler. Second, these reviews are processed on a multilingual network model with over 100,000 nodes. We found the network node most associated with each review's snippets and searched the network for the best label to display. Third, we prioritize reviews in a risk matrix divided into three priority levels. Fourth, we model the risk matrix in time series to detect issue peaks over time and trigger alerts. Finally, we present the output of the previous stages in a user-friendly real-time interface through an interactive dashboard. An overview can be seen in Figure 10.

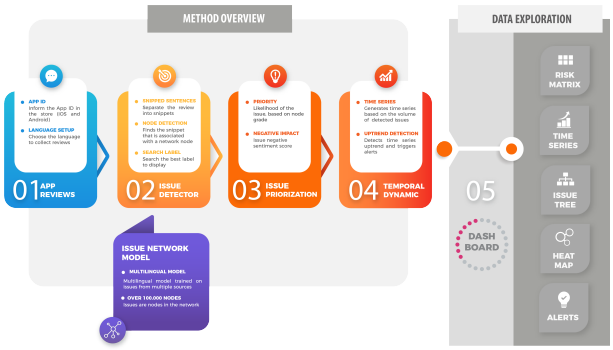

Figure 4: The architecture of the MApp-IDEA framework for detecting and prioritizing emerging issues

3.2 Issue Detector

We explore word embeddings to build acyclic graphs for representing app issues. Word embeddings have recently been proposed to

support NLP. They can calculate the semantic proximity between tokens and entire sentences, and the embeddings can be used as input to train the classifier. Additionally, graph-based methods have been widely used in several NLP tasks, such as text classification and summarization [37].

Our approach has a forest represented by the disconnected graph of the disjoint union of 3-tier trees. Each tree in the forest can have up to 3 tiers, wherein we have the best issue formations on the first and second tiers, as shown in Figure 5. The child nodes are connected to the parent nodes with better formations, but this is the same or related issue in practice. Therefore, we can group related issues and search the tree for the best node to display. We might have structurally weak issues in the tree, but we can search the tree and find the best good issue to display.

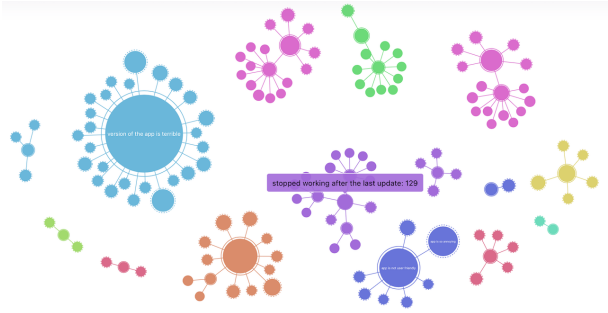

Figure 5: 3-tier tree dashboard

To generate the issues graph, we used a multilingual model based on the BERT [13], called DistilBERT (a distilled version of BERT) [46], which allows the processing of reviews in several languages for issue detection. We collected negative user opinions from different data sources and domains to train the model, e.g., repositories, app stores, and social networks. We have a trained model with more than 100.000 nodes.

BERT is a contextual NLM where we can learn a word embedding representation for a token for a given sequence of tokens. Formally, let $E = \{i_1, i_2, ..., i_n\}$ be a set of n extracted issues, where $i = (t_1, ..., t_k)$ are a sequence of k tokens of the issue i. BERT explores a masked language modeling procedure, i.e., the BERT model first generates a corrupted $\hat{x}$ version of the sequence, where approximately 15% of the words are randomly selected to be replaced by a special token called [MASK] [3]. One of the training objectives is the noisy reconstruction defined in Equation 1,

$$p(\bar{i}\|\hat{i}) = \sum_{j=1}^{k} m_j \frac{exp(\mathbf{h}_{c_j}^{\top}\mathbf{w}_{t_j})}{\sum_{t'} exp(\mathbf{h}_{c_j}^{\top}\mathbf{w}_{t'})} \quad (1)$$

where $\hat{i}$ is a corrupted token sequence of issue i, $\bar{i}$ is the masked tokens, m_t is equal to 1 when t_j is masked and 0 otherwise. The c_t represents context information for the t_j token, usually the neighboring tokens. The token embeddings are extracted from the pre-trained BERT model, where $\mathbf{h}_{c_j}$ is a context embedding, and $\mathbf{w}_{t_j}$ is a word embedding of the token t_j. The term $\sum_{t'} exp(\mathbf{h}_c^{\top}\mathbf{w}_{t'})$ is a normalization factor using all tokens t' from a context c. BERT uses the Transformer deep neural network to solve $p(\bar{i}\|\hat{i})$ of the Equation 1. For example, the vector space of embeddings preserves the proximity of similar issues but written in different ways by users such as "*problem with the payment*", "*i can't pay*", " *i can't complete the payment*" and "*payment error*".

Summarly, we found the graph node most associated with each review's snippets and searched the 3-tier tree for the best label to display. From this, we can prioritize the most critical issues through sentiment analysis and particulars of the issues graph.

3.3 Issue Prioritization

After identifying issues, app developers must prioritize and address the most critical issues and ensure timely software maintenance and evolution. We propose prioritizing issues through an automatically generated risk matrix combining sentiment analysis, clustering, and graph techniques. Therefore, given an app and its reviews, we summarize reviews with one or more issues into a risk matrix, as shown in Figure 6.

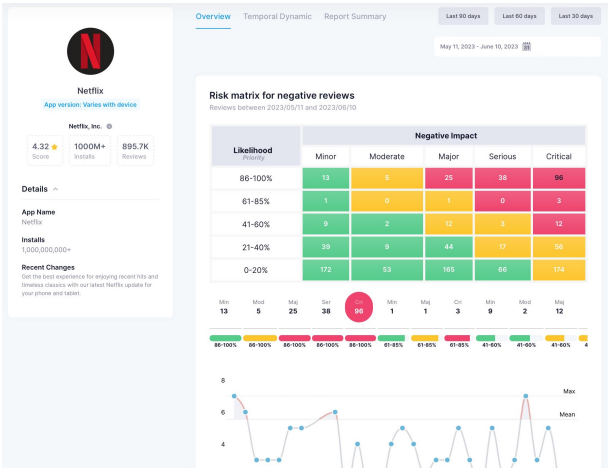

Figure 6: Navigable risk matrix. By clicking on the cell, it is possible to browse the time series of each cell

We introduce the automatic generation of a risk matrix to predict issues with the most significant potential negative impact and probability of occurring. First, we assume that the likelihood of an issue i occurring is related to the similarity distance d_i of the issue i in relation to other nodes in the graph. We also assume that the issue's negative impact is related to how negative the issue's sentiment score is. Therefore, for each detected issue, we need a measure to calculate how many nodes are associated with the issue in the graph and the issue's sentiment score.

3.4 Temporal Dynamic

Considering that the issue time series is a set of observations obtained sequentially over time, we can model the occurrence of issues over time in a time series to detect upward trends in their frequency. Formally, an issue time series I of size s can be represented as an ordered sequence of observations denoted as $I = (i_1, i_2, ..., i_s)$, where each observation i_t at time t belongs to the set of real numbers ($\mathbb{R}$) and represents the number of issues observed at that particular time point [6].

Table 1: Summary of results by window.

Window (days)	Release Related Peak (%)	Before	After
7	82.63 ± 18.85	60.61 ± 21.37	67.26 ± 19.92
6	80.24 ± 19.80	57.74 ± 21.05	65.31 ± 19.58
5	74.67 ± 21.18	53.66 ± 22.36	62.44 ± 20.22
4	68.14 ± 21.49	47.78 ± 22.44	58.35 ± 21.53
3	59.49 ± 22.46	39.19 ± 21.95	50.88 ± 22.59
2	48.58 ± 20.66	30.32 ± 19.66	44.23 ± 22.44
1	35.34 ± 19.89	16.13 ± 14.82	33.78 ± 21.54

In addition to capturing the temporal dynamics of issues, we also explore the temporal dynamics of the risk matrix. Once we have constructed the risk matrix, we examine its temporal behavior for each cell and prioritize level.

3.5 Results and Discussion

Regarding the issue detector classifier, our proposal had the fourth-best performance in the overall evaluation compared to supervised classifiers. The best results were obtained by supervised classifiers SVC, Random Forest, and MLP, as shown in Figure 7.

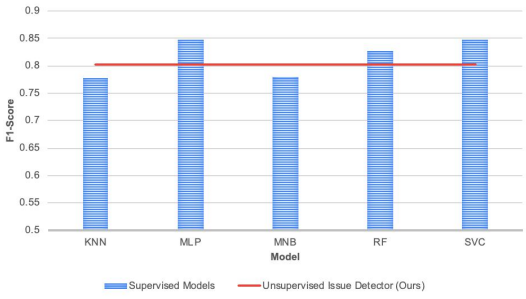

Figure 7: Comparison chart of our approach with supervised approaches

The results show that our unsupervised strategy, without data annotation, obtained a promising result compared to supervised strategies since obtaining annotated data is extremely costly or even impossible.

Regarding our approach for prioritizing issues, we analyzed Table 1 statistically for 3 populations (All, Before, and After). This was accomplished by examining 7 paired samples, each representing a distinct interval of time denoted in days as a window.

Our statistical analysis reveals a strong correlation between peaks and upward trends in the time series and app release dates. However, our approach demonstrates the capability to detect ascending patterns of issues within the temporal sequence leading up to their culmination. This empowers software professionals to proactively anticipate the release of corrective solutions, thereby mitigating potential issues before they escalate.

The findings revealed that approximately 64% of the releases are associated with issue peaks in the analyzed time series. Upon identification of a peak in the time series, merely half (50%) of the app releases are performed within three days or less, and approximately two-thirds (66%) within seven days. Our findings indicate that issues detected early by our approach are associated with later fix releases by developers, and issues caused by app releases can take more days to increase the volume of reported issues significantly. Nonetheless, by utilizing a risk matrix and temporal modeling, MApp-IDEA can effectively establish priorities and detect an ascending trend of potential issues before their culmination. The expeditious resolution of prioritized issues in a fiercely competitive environment is imperative in preventing unfavorable app evaluations. The MApp-IDEA promotes the anticipation of issue-fix releases by software engineers.

We have shown that opinions extracted from app reviews provide essential information about the app's issues and risks. We experimentally evaluated our unsupervised issue detection approach with state-of-the-art supervised methods, and the findings indicate that our approach is competitive.

Regarding the issue prioritization approach, we empirically evaluated a sample of 50 apps to validate our proposal. We process over 6.6 million reviews in 20 domains to evaluate our proposal, identifying and ranking the risk associated with nearly 270,000 issues. The findings show that issues detected and prioritized early with our approach are associated with later fix releases by developers.

Finally, we introduce an analytical data exploration tool that allows you to interactively browse the risk matrix, time series, heat map, and issue tree. The tool dashboard generates a complete real-time report of the relationship between releases and issue peaks, reports on average updates, average issues detected per day, and the average interval between releases. Additionally, our analytic system triggers alerts and notifications. More details are in [28] [27] [11].

Our main contributions of the MApp-IDEA method are briefly summarized below:

(1) **Automated Risk Matrix Generation:** We introduce the prioritizing issues approach that automatically generates a risk matrix, combining sentiment analysis, clustering, and graph theory.
(2) **Time Series-based Risk Dynamics:** We present a method to generate the temporal dynamics of issues and the risk matrix using time series. Our method uses interval segmentation to calculate the frequency of problems in each time interval, where we are especially interested in intervals where abrupt changes occur.
(3) **Interactive Risk Analysis Tool:** Finally, we introduce an analytical data exploration tool that allows you to interactively browse the risk matrix, time series, heat map, and issue tree. Additionally, our analytic system triggers alerts and notifications.

4 LEARNING RISK FACTORS FROM APP REVIEWS: A LARGE LANGUAGE MODEL APPROACH FOR RISK MATRIX CONSTRUCTION

A simple and intuitive way to organize and prioritize actions for software maintenance, aiming to reduce negative ratings, is through a risk matrix [49, 44]. This matrix consists of a graphical representation where risks are positioned on a Cartesian plane based on their

probability of occurrence and impact/severity. Risks are classified according to their importance and potential to harm app quality, as shown in Figure 8. Thus, it assists software engineering professionals in identifying the most critical areas that require prioritized attention. However, manual construction of a risk matrix often consumes a significant amount of time as stakeholders [43], such as project managers and product owners, need help understanding the context of risks recorded by the development team. For example, using different descriptions to report the same risk and the large volume of reviews make risk assessment challenging. Therefore, there is a need for automatic machine learning-based methods to extract risks from reviews and classify their priority.

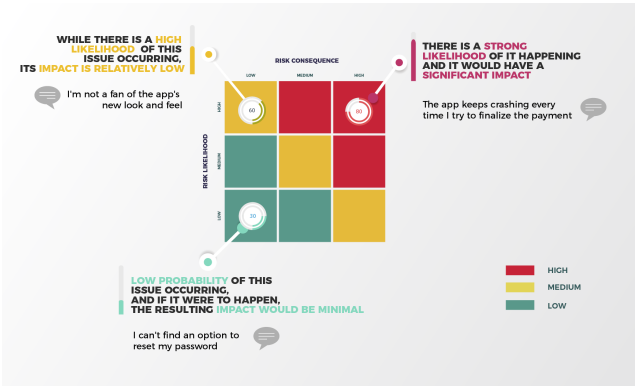

Figure 8: Risk matrix

This section presents a novel approach for generating Risk Matrix from App Reviews using Large Language Models, specifically the OPT model (Open Pre-trained Transformer Language Models) [50]. While large-scale language models like GPT [5] are widely used, we opted for OPT, an open-access language model. By providing specific instructions to the model through prompt engineering, it is possible to direct its attention to particular aspects of reviews, such as app features mentioned by users and the evaluation of risks' impact/severity associated with the apps.

4.1 Dynamic Prompt Construction for Feature Extraction

The first step of the proposed method involves the dynamic construction of prompts from a knowledge base of reviews from other apps, different from the target app, thereby avoiding the need for labeled data from the target application to be analyzed.

The knowledge base is represented through embeddings of reviews, which are numerical vectors that capture the semantics and context of words and phrases. These embeddings are obtained using deep learning algorithms, such as Sentence-BERT [45], which map texts into vector representations in latent spaces. Formally, given a set of software reviews in the knowledge base, we can represent them as $R_1, R_2, ..., R_n$, where R_i represents a specific review.

Each review R_i is converted into a vector representation using a pre-trained embedding model. This representation is denoted as $e(R_i)$, where $e()$ represents the embedding function. In this way, we have a set of vectors representing the reviews in the knowledge base: $e(R_1), e(R_2), ..., e(R_n)$.

To retrieve the most similar reviews to a target review of interest, we employ the k-nearest neighbors technique. In this approach, we calculate the similarity between the embedding vector of the target review and the embedding vectors of all the reviews in the knowledge base. The k-nearest neighbors are then used to generate prompts related to the extraction of text snippets that describe software features. This nearest neighbors search approach allows the method to leverage the existing knowledge base and learn from similar examples, becoming a type of few-shot learning for the task of feature extraction from software reviews.

4.2 Estimating Review Impact

Building upon the previous step, we have a list of features extracted from software reviews. Thus, the second step of the method utilizes each extracted feature from the previous step into a prompt to instruct the LLM to identify the severity or impact on five levels: negligible, minor, moderate, major, and critical.

This zero-shot learning process enables the model to identify the severity of features even without receiving specific prior examples for each feature. Although the model has not been explicitly trained on specific examples of severity classification in software reviews, it is capable of inferring patterns and generalizing based on the information captured during model pre-training.

4.3 Estimating Occurrence Likelihood

While the first two steps allow mapping reviews onto the "impact" dimension of the risk matrix, the third step is responsible for mapping reviews onto the "occurrence likelihood" dimension. In this step, a graph-based strategy is employed.

The reviews and extracted features from the previous step are represented as textual expressions of interest and treated as vertices in a graph. Similar pairs of vertices are connected through edges. The similarity between the expressions is measured using embeddings and cosine similarity. In this case, consider a set of expressions extracted from software reviews, represented as $E = \{t_1, t_2, ..., t_m\}$, where each t_i is an expression from the review containing the extracted feature. Similar to the first step, these expressions are converted into embeddings, which maps each expression to a feature vector. The similarity between two embedding vectors is calculated using a metric such as cosine similarity.

The degrees of the graph's vertices identify expressions that have a higher likelihood of occurrence. The degree values are discretized into five levels representing different levels of occurrence likelihood. For this purpose, the discretization also considers the average degree of the graph, using this value for normalization following a normal distribution. This normalization allows mapping the node degree values onto a standardized scale. Using the mean and standard deviation of the degree values, the normal distribution function is applied, where values close to the mean have a higher probability and values farther from the mean have a lower probability.

4.4 Results and Discussion

The experimental results are analyzed considering two main aspects: (1) the performance of the F1 score in the matching of feature extraction from app reviews, and (2) the error (MAPE and MAE)

in constructing the risk matrix, particularly in the impact dimension. The likelihood dimension in the reference risk matrices was obtained in the same way as the proposed method. Hence there are no significant variations for comparison.

Regarding the first aspect, we analyze the proposed dynamic prompt generation for OPT and the few-shot prompt learning, compared to a supervised reference approach based on RE-BERT and classical rule-based methods. The aim is to demonstrate the performance of OPT models in the absence of labeled data and the generalization capability of LLMs for new tasks and domains.

We observed that the proposed approach achieves superior results compared to rule-based methods but inferior results to the supervised RE-BERT model. However, it is important to note that supervised models require a significant amount of annotated data, necessitating the annotation of all features in each review of the training set for a model generation — a very time-consuming task. Although this strategy shows promising results, it may not be feasible in scenarios with a lack of domain experts or in dynamic settings with frequent review updates, which is common in mobile application quality monitoring and maintenance through reviews.

Our approach yielded promising results compared to the proprietary GPT model with zero-shot learning. In addition to requiring less labeled data than fully supervised models, our few-shot prompt learning strategy is based on open models, without restrictions on proprietary APIs or limitations on processing private or sensitive data.

Concerning the second aspect of evaluation, we compared the proposed approach with GPT. In this scenario, both models operate in the zero-shot learning format. However, it should be noted that we used OPT-IML (instruction meta-learning), which is fine-tuned with hundreds of instructions but with a smaller number of parameters. In this case, the utilized OPT-IML model has 1.3 billion parameters, and we analyzed the risk matrices generated with the features extracted from the previous step.

In summary, the experimental results suggest that open and accessible Large Language Models (LLMs) can play an important role in developing automated tools for analyzing mobile application reviews, facilitating risk identification, as well as contributing to monitoring and prioritizing software maintenance tasks. More details are in [26].

Our contributions to risk matrix construction using LLMs are three-fold:

(1) **Dynamic and automatic prompt generation:** We introduce an approach that enables the creation of customized instructions for each review to be analyzed, allowing the OPT model to extract app features as described by users in natural language. This enables more accurate and automated review analysis through few-shot learning, resulting in feature extraction with limited labeled data.
(2) **Prompt instructions to identify risk impact:** We develop suitable instructions to automatically identify the severity or impact of risks mentioned in the reviews, classifying them into five levels: negligible, minor, moderate, major, and critical. In this case, we employ zero-shot learning, meaning there is no need to provide examples to the model.
(3) **Evaluation of Open Pre-trained Large Language Models:** We evaluate how prompt engineering for OPT-based models compares to large proprietary language models such as GPT. By adopting OPT, we enable the use of large language models in scenarios with limited computational resources and constraints involving sensitive and private user data. This democratizes access to the usage of LLMs in more restricted contexts.

5 MAPP-IDEA TOOL

The tool is designed to provide a user-friendly real-time interface for monitoring and maintaining mobile apps, as illustrated in Figure 9. An overview can be seen in Figure 10. The front-end (interface and data exploration) was developed in PHP using the Laminas Framework [1], and the back-end (detection and prioritization algorithms) was developed using Python. Asynchronous tasks and front-end and back-end integration are performed by RESTful Bus. An overview of the tool's Model-View-Controller (MVC) architecture is illustrated in Figure 10.

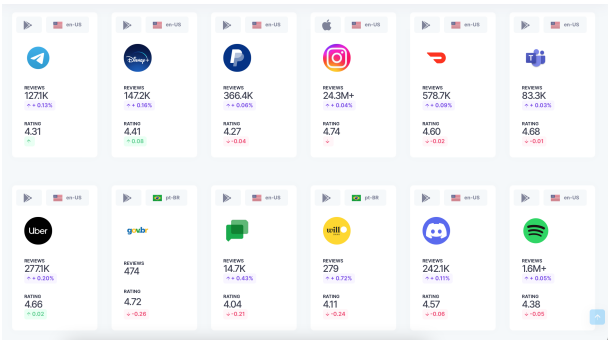

Figure 9: Dashboard with daily performance information for each app on the star rating and number of reviews

Figure 10: MVC architecture of the MApp-IDEA tool

[1] https://getlaminas.org

6 CONCLUSIONS

We introduce a two-fold approach, called MApp-Reviews and MApp-IDEA, to explore emerging issues from user feedback to proactively detect, predict, prioritize, and monitor issues and risks over time. We also introduce an approach using the recent LLM for the risk matrix construction, incorporating a dynamic and automatic prompt generation technique for classifying. These approaches enable us to effectively address reviews on time, mitigate negative impacts on the overall app rating, and maintain the app's competitiveness, ensuring timely maintenance and facilitating software evolution.

This work contributes to the field of software maintenance and evolution by providing a novel approach for detecting and predicting emerging software requirements issues based on user reviews. It offers valuable insights into the temporal dynamics of issues and associated risks and emphasizes the importance of proactive maintenance to ensure software quality and user satisfaction.

Our research results show new promising prospects for the future, and new possibilities for innovation research in this area have emerged with our results so far.

SCIENTIFIC IMPACT AND OUTCOMES

The research has significantly advanced the field of software quality within software engineering through the development of innovative tools and methodologies. Notably, the MApp-IDEA tool has garnered recognition for its ability to systematically collect and analyze millions of app reviews, thus enhancing the processes of monitoring, issue detection, and prioritization. Additionally, MApp-Reviews introduces a method for clustering software requirements to model their temporal dynamics, utilizing contextual word embeddings to quantify negative user mentions over time. This approach helps predict software requirements associated with negative reviews and improves forecasting accuracy by incorporating domain-specific information. Furthermore, the research has pioneered using LLMs for the dynamic generation of the risk matrix, automating identifying and classifying risks in user reviews.

As a result of this research, recent works such as iRisk [10] and MApp-TIME [12] have been proposed to enable a microservices-oriented architecture, scaling the approaches presented in this article. The introduction of MApp-TIME and iRisk further underscores the impact of this work. MApp-TIME employs a microservices architecture to monitor the temporal dynamics of issues and app releases, reducing the time between issue detection and resolution. iRisk leverages Large Language Models (LLMs) to construct scalable risk matrices from large datasets, offering automated dashboards and visualizations for effective decision-making and risk mitigation. These advancements collectively elevate software quality standards, influence industry best practices, and contribute to more efficient software maintenance and evolution.

Highlighted Publications

(1) Temporal dynamics of requirements engineering from mobile app reviews. 2022 [25].
(2) MApp-IDEA: Monitoring App for Issue Detection and Prioritization. 2023 [28].
(3) Monitoring Temporal Dynamics of Issues in Crowdsourced User Reviews and their Impact on Mobile App Updates. 2024 [12].
(4) iRisk: A Scalable Microservice for Classifying Issue Risks Based on Crowdsourced App Reviews. 2024 [10].

Public Domain Softwares

MApp-IDEA (Monitoring App for Issue Detection and Prioritization)[2] is available online at *https://mappidea.com*. A detailed video demonstrating its functionalities can be found in this video[3]. The tool is available for use in the industry, assisting software engineers in the evolution and maintenance of applications.

Repositories

- MApp-Reviews
- MApp-IDEA
- Risk Matrix
- MApp-Time
- Illama-Client
- iRisk

ACKNOWLEDGMENTS

This study was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) [process number 426663/2018-7], Federal University of Mato Grosso do Sul (UFMS), São Paulo Research Foundation (FAPESP) [process number 2019/25010-5 and 2019/07665-4], BIRDIE.AI (Project CEIA/UFG - PEIA-2105.0011), Brazilian Company of Research and Industrial Innovation (EMBRAPII), Artificial Intelligence (C4AI-USP), and from the IBM Corporation.

REFERENCES

[1] Nadeem Al Kilani, Rami Tailakh, and Abualsoud Hanani. 2019. Automatic classification of apps reviews for requirement engineering: exploring the customers need from healthcare applications. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, 541–548. DOI: 10.1109/SNAMS.2019.8931820.
[2] Afnan AlSubaihin, Federica Sarro, Sue Black, Licia Capra, and Mark Harman. 2019. App store effects on software engineering practices. *IEEE Transactions on Software Engineering*.
[3] Adailton Araujo and Ricardo Marcondes Marcacini. 2021. Re-bert: automatic extraction of software requirements from app reviews using bert language model. In *The 36th ACM/SIGAPP Symposium On Applied Computing*. DOI: 10.1145/3412841.3442006.
[4] Adailton F Araujo, Marcos PS Gôlo, and Ricardo M Marcacini. 2022. Opinion mining for app reviews: an analysis of textual representation and predictive models. *Automated Software Engineering*, 29, 1, 1–30.
[5] Tom B. Brown et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. Vol. 33, 1877–1901.
[6] Chris Chatfield. 2003. *The analysis of time series: an introduction*. Chapman and hall/CRC.
[7] Chris Chatfield and Haipeng Xing. 2019. *The analysis of time series: an introduction with R*. CRC press.
[8] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, 767–778.
[9] Jacek Dabrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2020. Mining user opinions to support requirement engineering: an empirical study. In *Advanced Information Systems Engineering*. Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, (Eds.) Springer International Publishing, Cham, 401–416. ISBN: 978-3-030-49435-3.

[2] Awarded paper at the XXXVI Brazilian Symposium on Software Engineering (SBES)
[3] https://youtu.be/xkWUAN-NDgw

[10] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, and Ricardo Marcondes Marcacini. 2024. Irisk: a scalable microservice for classifying issue risks based on crowdsourced app reviews. In *Proceedings of the 40th International Conference on Software Maintenance and Evolution (ICSME)*. Flagstaff, AZ, United States.

[11] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, and Ricardo Marcondes Marcacini. 2023. Issue detection and prioritization based on app reviews. (2023). DOI: 10.21203/rs.3.rs-2838568/v1.

[12] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, and Ricardo Marcondes Marcacini. 2024. Monitoring temporal dynamics of issues in crowdsourced user reviews and their impact on mobile app updates. In *Proceedings of the 40th International Conference on Software Maintenance and Evolution (ICSME)*. Flagstaff, AZ, United States.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. http://arxiv.org/abs/1810.04805 arXiv: 1810.04805.

[14] Mauro Dragoni, Marco Federici, and Andi Rexha. 2019. An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Information Processing and Management*, 56, 3, 1103–1118. DOI: https://doi.org/10.1016/j.ipm.2018.04.010.

[15] Laura V. Galvis Carreño and Kristina Winbladh. 2013. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering* (ICSE '13). IEEE Press, San Francisco, CA, USA, 582–591. ISBN: 9781467330763.

[16] C. Gao, J. Zeng, Z. Wen, D. Lo, X. Xia, I. King, and M. R. Lyu. 2022. Emerging app issue identification via online joint sentiment-topic tracing. *IEEE Transactions on Software Engineering*, 48, 08, (Aug. 2022), 3025–3043. DOI: 10.1109/TSE.2021.3076179.

[17] María Gómez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier. 2015. A recommender system of buggy app checkers for app store moderators. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems* (MOBILESoft '15). IEEE Press, Florence, Italy, 1–11. ISBN: 9781479919345.

[18] Xiaodong Gu and Sunghun Kim. 2015. "what parts of your apps are loved by users?" (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 760–770.

[19] Emitza Guzman and Walid Maalej. 2014. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)*. IEEE, 153–162.

[20] Steffen Herbold, Alexander Trautsch, and Fabian Trautsch. 2020. On the feasibility of automated prediction of bug and non-bug issues. *Empirical Software Engineering*, 25, 6, (Nov. 2020), 5333–5369. https://doi.org/10.1007/s10664-020-09885-w.

[21] Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (MSR '13). IEEE Press, San Francisco, CA, USA, 41–44. ISBN: 9781467329361.

[22] Timo Johann, Christoph Stanik, Walid Maalej, et al. 2017. Safe: a simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 21–30.

[23] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. 2015. What do mobile app users complain about? *IEEE Software*, 32, 3, 70–77. DOI: 10.1109/MS.2014.50.

[24] Sherlock A. Licorish, Bastin Tony Roy Savarimuthu, and Swetha Keertipati. 2017. Attributes that predict which features to fix: lessons for app store mining. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (EASE'17). Association for Computing Machinery, Karlskrona, Sweden, 108–117. ISBN: 9781450348041. DOI: 10.1145/3084226.3084246.

[25] Vitor Lima, Adailton Araújo, and Ricardo Marcacini. 2022. Temporal dynamics of requirements engineering from mobile app reviews. *PeerJ Computer Science*, 8, (Mar. 2022), e874. DOI: 10.7717/peerj-cs.874.

[26] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, Ricardo Marcondes Marcacini, et al. 2023. Learning risk factors from app reviews: a large language model approach for risk matrix construction. *PREPRINT*, 1, (July 2023). Version 1. https://doi.org/10.21203/rs.3.rs-3182322/v1.

[27] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, and Ricardo Marcondes Marcacini. 2023. Mapp-idea: design and architecture of the analytical data exploration tool from app reviews. (2023).

[28] Vitor Mesaque Alves de Lima, Jacson Rodrigues Barbosa, and Ricardo Marcondes Marcacini. 2023. Mapp-idea: monitoring app for issue detection and prioritization. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (SBES '23). Association for Computing Machinery, <conf-loc>, <city>Campo Grande</city>, <country>Brazil</country>, </conf-loc>, 180–185. ISBN: 9798400707872. DOI: 10.1145/3613372.3613417.

[29] Bing Liu. 2012. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5, 1, 1–167.

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

[31] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* number 14. Vol. 1. Oakland, CA, USA, 281–297.

[32] Saurabh Malgaonkar, Sherlock A. Licorish, and Bastin Tony Roy Savarimuthu. 2022. Prioritizing user concerns in app reviews – a study of requests for new features, enhancements and bug fixes. *Information and Software Technology*, 144, 106798. DOI: https://doi.org/10.1016/j.infsof.2021.106798.

[33] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2016. A survey of app store analysis for software engineering. *IEEE transactions on software engineering*, 43, 9, 817–847.

[34] Stuart Mcilroy, Nasir Ali, and Ahmed E. Hassan. 2016. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Softw. Engg.*, 21, 3, (June 2016), 1346–1370. DOI: 10.1007/s10664-015-9388-2.

[35] Stuart Mcilroy, Nasir Ali, Hammad Khalid, and Ahmed E. Hassan. 2016. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Softw. Engg.*, 21, 3, (June 2016), 1067–1106. DOI: 10.1007/s10664-015-9375-7.

[36] Montassar Ben Messaoud, Ilyes Jenhani, Nermine Ben Jemaa, and Mohamed Wiem Mkaouer. 2019. A multi-label active learning approach for mobile app user review classification. In *Knowledge Science, Engineering and Management: 12th International Conference, KSEM 2019, Athens, Greece, August 28–30, 2019, Proceedings, Part I*. Springer-Verlag, Athens, Greece, 805–816. DOI: 10.1007/978-3-030-29551-6_71.

[37] Rada Mihalcea and Dragomir Radev. 2011. *Graph-based natural language processing and information retrieval*. Cambridge university press.

[38] Maleknaz Nayebi, Bram Adams, and Guenther Ruhe. 2016. Release practices for mobile apps – what do users and developers think? In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1, 552–562. DOI: 10.1109/SANER.2016.116.

[39] Maleknaz Nayebi, Mahshid Marbouti, Rache Quapp, Frank Maurer, and Guenther Ruhe. 2017. Crowdsourced exploration of mobile app features: a case study of the fort mcmurray wildfire. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Society Track (ICSE-SEIS)*, 57–66.

[40] Ehsan Noei, Feng Zhang, and Ying Zou. 2021. Too many user-reviews! what should app developers look at first? *IEEE Transactions on Software Engineering*, 47, 2, 367–378. DOI: 10.1109/TSE.2019.2893171.

[41] D. Pagano and W. Maalej. 2013. User feedback in the appstore: an empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, 125–134. DOI: 10.1109/RE.2013.6636712.

[42] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2018. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software*, 137, 143–162. DOI: https://doi.org/10.1016/j.jss.2017.11.043.

[43] Nicola Paltrinieri, Louise Comfort, and Genserik Reniers. 2019. Learning about risk: machine learning for risk assessment. *Safety science*, 118, 475–486.

[44] Marzuki Pilliang, Munawar, Muhammad Abdullah Hadi, Gerry Firmansyah, and Budi Tjahjono. 2022. Predicting risk matrix in software development projects using bert and k-means. In *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 137–142. DOI: 10.23919/EECSI56542.2022.9946637.

[45] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3973–3983.

[46] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. (2019). DOI: 10.48550/ARXIV.1910.01108.

[47] Sean J. Taylor and Benjamin Letham. 2018. Forecasting at Scale. *The American Statistician*, 72, 1, (Jan. 2018), 37–45.

[48] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 14–24.

[49] Li Xiaosong, Liu Shushi, Cai Wenjun, and Feng Songjiang. 2009. The application of risk matrix to software project risk management. In *2009 International Forum on Information Technology and Applications*. Vol. 2. IEEE, 480–483.

[50] Susan Zhang et al. 2022. Opt: open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

[51] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. 2020. Natural language processing (nlp) for requirements engineering: a systematic mapping study. *arXiv preprint arXiv:2004.01099*.