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

Vitor Mesaque Alves de Lima

Vitor Mesaque Alves de Lima

Advisor: *Prof. Dr. Ricardo Marcondes Marcacini*

Thesis presented to the Faculty of Computing (FACOM) of the Federal University of Mato Grosso do Sul (UFMS) as part of the requirements necessary to obtain the Doctor of Computer Science degree.

UFMS - Campo Grande
July/2023

My parents,
Josimar and Linda (in memoriam),

My wife and daughter,
Gabriela and Melissa,

to Ricardo Marcondes Marcacini.

Acknowledgments

Firstly, I thank God for giving me the intelligence and wisdom to develop this work.

I am deeply grateful to my beloved wife, Gabriela, who has been the source of endless love, care, and unwavering support, always by my side, encouraging and motivating me. As we expectantly await the arrival of our daughter, Melissa, she is a precious gift from God that fills our hearts with joy.

I would like to honor my parents, Josimar and Linda (in memoriam), for they represent an example of faith and love. Everything I am today is the result of their love and dedication. I am also so grateful to my entire family for their support and shared joyful moments. They brought lightness and happiness to my journey.

To my advisor and friend, Professor Ricardo Marcacini, for his dedication, support, and encouragement in all my academic and professional achievements. I had the privilege of working with him on several innovation projects while teaching colleagues at CPTL/UFMS. I am very grateful for all the learning and companionship in all these years.

I would also like to acknowledge the significant contributions made by Professor Jacson Barbosa to the development of this work.

I am grateful to all my friends and brothers from the Verbo da Vida church for their support and encouragement throughout this journey.

To all my co-workers on the UFMS/CPTL, particularly the professors of the Information Systems course.

To UFMS for providing the necessary support and infrastructure that facilitated the completion of this work.

This study was also supported directly and partially by the Brazilian National Council for Scientific and Technological Development (CNPq), São Paulo Research Foundation (FAPESP), BIRDIE.AI, Brazilian Company of Research and Industrial Innovation (EMBRAPII), Artificial Intelligence (C4AI-USP), and from the IBM Corporation.

“that my heart may sing your praises and not be silent.
LORD my God, I will praise you forever.“
(Psalm 30.12)

Abstract

Lima, V. M. A. **Opinion Mining for App Reviews: Identifying and Prioritizing Emerging Issues for Software Maintenance and Evolution**. 2023. 150p. Thesis (PhD in Computer Science) - Faculty of Computing of the Federal University of Mato Grosso do Sul, Campo Grande - MS, 2023.

Context. 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 doctoral project 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.

Resumo

Lima, V. M. A. **Mineração de Opiniões para Avaliações de Aplicativos: Identificação e Priorização de Problemas Emergentes para Manutenção e Evolução de Software**. 2023. 150p. Tese (Doutorado em Ciência da Computação) - Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande - MS, 2023.

Contexto. A mineração de opinião para avaliações de aplicativos tem como objetivo analisar os comentários dos usuários nas lojas de aplicativos para apoiar as atividades de engenharia de software, principalmente a manutenção e evolução de software. Identificar prontamente problemas emergentes, como *bugs*, é um dos principais desafios na manutenção da qualidade do software. **Lacuna.** No entanto, analisar manualmente esses comentários é um desafio devido à grande quantidade de dados textuais. Métodos baseados em aprendizado de máquina têm sido empregados para automatizar a mineração de opinião e lidar com essa questão. Embora métodos recentes tenham alcançado resultados promissores na extração e categorização de problemas a partir das opiniões dos usuários, os estudos existentes concentram-se principalmente em auxiliar os engenheiros de software a explorar o comportamento histórico dos usuários em relação às funcionalidades do aplicativo e não exploram mecanismos de deteção de tendências e classificação de risco de problemas emergentes. Além disso, os estudos anteriores não abrangem o processo completo de análise de problemas e riscos por meio de uma abordagem não supervisionada. **Contribuição.** Este projeto de doutorado avança o estado da arte na mineração de opinião para reviews de aplicativos, propondo uma abordagem não supervisionada para identificar e priorizar problemas emergentes. Nosso objetivo é minimizar o tempo entre a ocorrência de um problema e sua correção, permitindo uma rápida identificação do problema. Propomos duas novas abordagens que (i) identifica possíveis requisitos de software defeituosos e treina modelos preditivos para antecipar requisitos com maior probabilidade de avaliação negativa e (ii) detecta problemas a partir de avaliações, classifica-

os em uma matriz de risco com níveis de priorização e monitora sua evolução ao longo do tempo. Adicionalmente, apresentamos uma abordagem de construção da matriz de risco usando os recentes *Large Language Models (LLMs)*. **Resultados.** Processamos mais de 6.6 milhões de comentários de usuários para avaliar nossa proposta, identificando e classificando o risco associado a quase 270.000 problemas. Os resultados demonstram a competitividade de nossa abordagem não supervisionada em comparação com modelos supervisionados existentes. **Conclusões.** Comprovamos que as opiniões extraídas dos comentários dos usuários fornecem percepções importantes sobre os problemas e riscos associados aos aplicativos, que podem ser detectados antecipadamente para mitigar seu impacto. Nosso processo de mineração de opinião implementa a análise automatizada de problemas, com priorização baseada em risco e monitoramento temporal.

Contents

List of Figures

List of Tables

List of abbreviations

ABSA *Aspect-based Sentiment Analysis*

AHP *Analytical Hierarchy Process*

ANOVA *Analysis of Variance*

ARIMA *Autoregressive Integrated Moving Average Models*

ASUM *Aspect and Sentiment Unification Model*

BERT *Bidirectional Encoder Representations from Transformers*

CNN *Convolutional Neural Networks*

CRF *Conditional Random Fields*

DC *Dummy Classifier*

DIVER *iDentifying emerging app Issues Via usER feedback*

DNN *Deep Neural Networks*

DistilBERT *a distilled version of BERT*

EAI *Emerging App Issue*

EM *Expectation-Maximization*

Faiss *Facebook AI Similarity Search*

GPT *Generative Pre-trained Transformer*

HMM *Hidden Markov Models*

JAS *Joint Aspect/Sentiment Model*

JST *Joint Sentiment Topic*

KNN *K-Nearest Neighbor*

LDA *Latent Dirichlet allocation*

LLM *Large Language Models*

LR *Logistic Regression*

LSA *Latent Semantic Analysis*

MAE *Mean Absolute Error*

MAPP-Reviews *Monitoring App Reviews*

MApp-IDEA *Monitoring App for Issue Detection and Prioritization*

MAPE *Mean Absolute Percentage Error*

MERIT *iMproved EmeRging Issue deTection*

ME *Maximum Entropy*

MLP *Multi-layer Perceptron*

MLP *Multilayer Perceptron*

MNB *Multinomial Naive Bayes*

MVC *Model-View-Controller*

NB *Naive Bayes*

OPT *Open Pre-trained Transformers*

OPT-IML *OPT Instruction meta-Learning*

PMI *Pointwise Mutual Information*

POS *Part-Of-Speech*

RE-BERT *Requirements Engineering using Bidirectional Encoder Representations from Transformers*

RE *Requirements Engineering*

REST *Representational State Transfer*

RF *Random Forest*

RNN *Recurrent Neural Networks*

LSTM *Long-Short Term Memory*

GRU *Gated Recurrent Unit*

SD *Standard Deviation*

SLDA *Sentence-LDA*

SVC *Support Vector Clustering*

SVM *Support Vector Machines*

TF-RBM *Two-fold Rules-based Model*

VADER *Valence Aware Dictionary for Sentiment Reasoning*

kNN-TSP *k-Nearest Neighbors*

pLSA *Probabilistic Latent Semantic Analysis*

CHAPTER

1

Introduction

This chapter provides a comprehensive thesis overview, describing the addressed problems and research objectives. Section 1.1 presents an outline of the research topic addressed in this thesis. Section 1.2 elucidates the objectives and research questions of the work, which will be addressed in subsequent chapters. Section 1.3 briefly highlights the main contributions of this thesis. Finally, Section 1.4 outlines the organization of the thesis and research timeline, providing a concise description of the content covered in each chapter.

1.1 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) (Martin et al., 2016; AlSubaihin et al., 2019; Dabrowski et al., 2020; Araujo and Marcacini, 2021; Malgaonkar et al., 2022). 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 (Johanssen et al., 2019; Martin et al., 2016). Therefore, to deal with these challenges, opinion mining has been increasingly used for computational analysis of people's opinions from textual data (Liu, 2012a). 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 (Lima et al., 2022).

In this thesis, we define an emerging app issue as follows.

Definition 1.1 (Emerging App Issue) *An issue is an Emerging App Issue (EAI) if it reports in informative reviews from users in a specific time slice that represents bugs, misbehavior, environment failure, or negative user experience with risks associated, where their occurrences distributions in the current time slice correspond to a significant increase in measure terms compared to fluctuations in previous time slices.*

One of the most challenging difficulties for software quality maintenance is quickly identifying emerging issues, such as bugs. These emerging issues can result in enormous costs since users may fail to complete critical tasks or experience dissatisfaction, leading them to uninstall the app. According to a survey (Nayebi et al., 2018b), 78.3% of developers agree that deleting unnecessary and malfunctioning features is equally or more important than introducing new features. According to Lientz and Swanson (1980), maintenance activities are categorized into four classes: i) adaptive - changes in the software environment; ii) perfective - new user requirements; iii) corrective - fixing errors; and iv) preventive - prevent problems in the future. By Bennett and Rajlich (2000), it was demonstrated that a significant proportion, specifically approximately 21%, of the total maintenance effort was allocated towards the final two categories.

In the context of mobile apps, Mcilroy et al. (2016a) 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 (Licorish et al., 2017). 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 (Nayebi et al., 2016). 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, such as device combinations, screen sizes, operating systems, and network conditions (Palomba et al., 2018).

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 (Dabrowski et al., 2020; Araujo

et al., 2022; Lima et al., 2022). 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 (Lima et al., 2022). 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 (Licorish et al., 2017; Groen et al., 2015; Malgaonkar et al., 2022).

A recent study on app review analysis showed that the temporal dynamics of words and expressions associated with app malfunctions behave similarly to trending topics on social networks (Gao et al., 2022). For example, a statistic concept called burstiness often occurs in reviews, in which there is an accelerated growth of expressions reporting app malfunction, as illustrated in Figure 1.1 (app malfunction discovery period) (Lima et al., 2022). This scenario generates negative app evaluations, which can remain even after corrective updates.

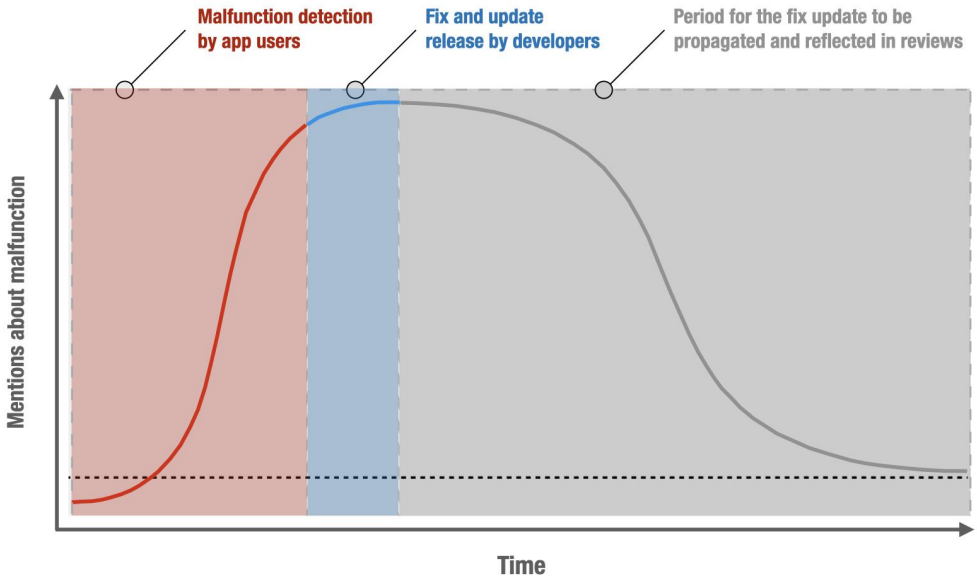

Figure 1.1: Illustration of a temporal evolution pattern of a topic extracted from reviews describing app malfunctions.

Different strategies have been recently proposed to support issue detection and classification, (Zhao et al., 2020), such as issues categorization (Iacob and Harrison, 2013; Galvis Carreño and Winbladh, 2013; Pagano and Maalej, 2013; Mcilroy et al., 2016b; Khalid et al., 2015; Chen et al., 2014; Gómez et al., 2015; Gu and Kim, 2015; Maalej and Nabil, 2015; Villarroel et al., 2016; Nayebi et al., 2017; Noei et al., 2021; Araujo et al., 2022; Herbold et al., 2020; Messaoud et al., 2019; Al Kilani et al., 2019; Gao et al., 2022) and prioritization

(Licorish et al., 2017; Malgaonkar et al., 2022).

Although previous studies are promising, they do not cover the entire process from start to finish of issue analysis, from the automated collection of reviews to the task of detection, prioritization, and analysis using an unsupervised approach.

1.2 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 Chapter 3, while RQ2 was mainly addressed in Chapters 4 and 5 of this thesis.

1.3 Main Contributions

This thesis 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.

This 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.

As a result of the above-mentioned main objectives, we present an analytical data exploration tool with an interactive dashboard and a real-time issue monitor, illustrated in Figure 1.2.

In the following, we present individual summaries of the contributions made by our proposed two-fold opinion mining approach, referred to as **(i)** MApp-Reviews (Monitoring App Reviews) and **(ii)** MApp-IDEA (Monitoring App for Issue Detection and Prioritization), and the proposed risk matrix construction using Large Language Models.

- The MApp-Reviews method contributes to the following points:
  - **Software requirements candidates extraction and clustering**. We introduce software requirements clustering to standardize different variations of user descriptions of software requirements. We utilize contextual word embeddings for software requirements representation, enabling us to quantify accurately negative user mentions over time.
  - **Faulty requirement temporal dynamics analysis**: We propose a methodology for modeling the temporal dynamics of negative ratings for clusters of software requirements using time series analysis. Our approach employs equal-interval segmentation to determine the frequency of software requirement mentions within each time interval. This enables us to examine and visualize temporal variations, with a particular focus on sudden increases in issues that may potentially impact the future evaluation of an app.
  - **Incorporating domain-specific information into the forecast model**. We explore incorporating software domain-specific information in time series forecasting. This information includes factors that affect user behavior, such as holidays, app releases and updates, marketing campaigns, and other external events. We investigate automatic and manual trend changepoint estimation to improve forecasting accuracy.

- The MApp-IDEA method contributes to the following points:
  - **Issue detection approach**. We introduce an issue detection method that explores word embeddings to build acyclic graphs representing app issues.
  - **Prioritizing issues approach**. We introduce an approach that automatically generates a risk matrix by combining sentiment analysis, clustering, and graph theory. This approach helps in prioritizing issues based on their significance and potential impact.

- **Risk temporal dynamics analysis** Using time series analysis, we present a method to generate the temporal dynamics of issues and risks. We utilize interval segmentation to calculate the frequency of problems in each time interval, focusing on identifying intervals with sudden changes.
- **Analytical data exploration tool**. We introduce an interactive data exploration tool, illustrated in Figure 1.2, that allows users to browse the risk matrix, time series, heat map, and issue tree. Additionally, our analytic system includes real-time performance reports, alerts, and notification functionalities to inform users about significant consequences.

- **Large Language Models for issues priorization**. We present a risk matrix construction from app reviews using the recent Large Language Models (LLMs). By utilizing Open Pre-trained Transformers (OPT), our approach enables the use of LLMs in scenarios with limited computational resources and data privacy constraints. We introduce a dynamic prompt generation technique to extract app characteristics mentioned by users. We also create instructions to classify risks into five levels of severity. Experimental results show competitiveness against proprietary models like GPT (Generative Pre-trained Transformer).

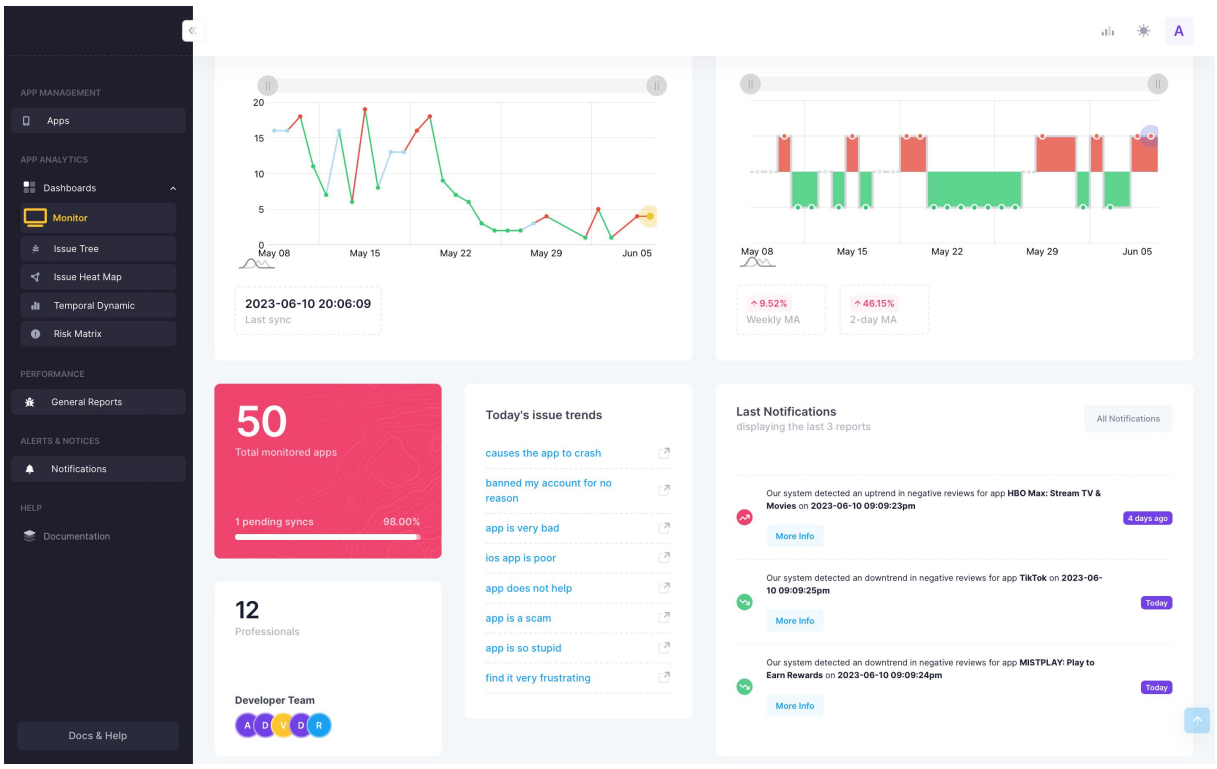

Figure 1.2: Dashboard home screen information about notifications, sync status, and time series of the last updated app

1.4 Organization and Research Timeline

We present the organization and research timeline that ensures a logical progression of proposals throughout the thesis. The proposed methods and approaches build upon one another, with each chapter presenting advancements and improvements over the previous ones. This sequential organization strengthens the coherence and comprehensiveness of our research work. Figure 1.3 presents an overview of the timeline and the chapters' distribution.

Figure 1.3: Research timeline

The first chapter presents the research challenges in opinion mining from the app reviews field, defines the problem, provides an overview of the proposed approaches, and highlights the main contributions.

Chapter 2 offers a comprehensive literature review on mining user opinions to support requirement engineering and identify and prioritize emerging issues.

Chapter 3 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. Chapter 3 describes the initial direction of our research.

In Chapter 4, we examine issue detection and prioritization based on app reviews, which enhances the opinion-mining process employed in the previous approach presented in Chapter 3. We introduce the MApp-IDEA method and experimentally evaluate 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 Chapter 4 is superior and brings more technical advancements and results. In practice, MApp-IDEA represents an evolution of MApp-Reviews.

Chapter 5 investigates how recent Large Language Models such as GPT and OPT can be leveraged to facilitate the automatic construction of risk ma-

trices from app reviews. This investigation encompasses various stages, from extracting review features to classifying them into priority levels. The next research direction is an improvement of the risk matrix construction of MApp-IDEA (Chapter 4) to incorporate the proposed LLM-based approach presented in Chapter 5.

Design patterns and technologies employed in our data mining analytics tool are discussed in Chapter 6. Chapter 6 describes, from an architectural perspective, the tool introduced in Chapter 4.

Lastly, the Conclusions (Chapter 7) summarize this thesis's main contributions, address its limitations, offer final considerations, and suggest directions for future research.

CHAPTER

2

Fundamentals and Related Work

2.1 Introduction

This chapter presents the literature review of the fundamentals and related work. First, in section 2.2, we discuss the problem of opinion mining and sentiment analysis. Next, in section 2.2.1, we present the techniques for extracting aspects into four approaches: Frequency-based, Relation-based, Supervised Learning, and Topic Modeling. In section 2.2.2, we discussed three main approaches for aspect sentiment classification: Supervised Learning, Lexicon-Based Learning, and Deep Learning. Section 2.2.3 covers clustering concepts and approaches, such as partitional and hierarchical clustering and some aspect clustering techniques. Section 2.3 defines time series analysis concepts using machine learning and main components: trend, seasonality, and residual. In section 2.4, we discuss data-driven requirements engineering and point out the limitations of traditional requirements engineering. In section 2.5 we discuss neural language model-based representations and recent large language models. In the section 2.6, we discussed and compared the related works about mining user opinions to support requirement engineering and issue prioritization based on app reviews. Finally, we present the final remarks highlighting the main limitation of the previous works.

2.2 Opinion Mining

Opinions influence people's behavior, being essential in many human activities. For example, individuals want to know what others think before buying

or using a particular product (Pozzi et al., 2017). Opinion mining, also called sentiment analysis, studies people's opinions, feelings, evaluations, and emotions about entities and their attributes, such as products, services, organizations, people, topics, problems, and events (Liu, 2012b).

The detection of sentiment in a text can occur at different granularities, and the decision of the level is subject to context and application. The analysis can be done at the following levels (Liu, 2012b):

- **Document:** At this level, the task is to determine whether a document expresses a positive or negative feeling. This granularity is suitable when the document deals with a single entity, for example, a document that provides an opinion about a given product;
- **Sentence:** This level determines the feeling of a specific sentence in a document. This level is often used when the same document contains opinions about several entities. It identifies and distinguishes objective (facts) and subjective (opinion) sentences.
- **Entity and Aspect:** This level focuses on the opinion expressed, regardless of the constructs used to express it (e.g., documents, sentences, clauses). In this case, the target of the opinion can be an entity or some of its aspects. For example, consider the review "I love my X camera because the quality of its lens is exceptional. Too bad the price is so high". We observe three opinions in two sentences: about the "X camera", and about two of its aspects (price and lens). Only the opinion about the price is negative, and the opinion about the lens and the camera, in general, are positive. This level is the most complex level of analysis, which has been extensively studied in product and service reviews (Tsytsarau and Palpanas, 2011; Ghani et al., 2006; Hu and Liu, 2004).

Aspect expressions that are nouns and noun phrases are called explicit aspect expressions. Aspect expressions that are not nouns or noun phrases are called implicit aspect expressions (Hu and Liu, 2004; Liu, 2012b). In practical terms, explicit aspects are words or expressions that directly refer to a technical characteristic of a product or service. For example, in "The screen has a good brightness", the aspect "screen" is a technical feature explicitly used by the reviewer. On the other hand, implicit aspects are indirect references to characteristics of products or services, usually through expressions about the behavior of the aspect. For example, in the text "My photos are too dark", the opinion is about the smartphone's implicit aspect "camera" (Santos et al., 2021).

The extraction of aspects and entities in texts and opinion polarity classification is called Aspect-Based Sentiment Analysis (ABSA) or Feature-based

Opinion Mining (Zhang et al., 2015). Aspect-level opinion mining is based on the premise that an opinion consists of an opinion term (positive or negative) and a target (opinion target). Entities and their respective aspects define a target. Therefore, it aims to classify the sentiment about the specific aspects of each entity. An opinion identified without its target is of limited use (Liu, 2012b).

The ABSA can be divided into three main subtasks: i) Aspect extraction: this task extracts the target of the opinion, i.e., the aspect or topic extracted from a sentence; ii) Aspect clustering: this task groups synonymous into aspect categories, which each category represents a unique aspect.; and iii) Aspect sentiment classification: this task determines whether the opinions on different aspects are positive, negative, or neutral (Liu, 2012b; Pontiki et al., 2016; Trisna and Jie, 2022). 2.1 shows the main subtasks in ABSA.

The aspect extraction and polarity classification in opinion mining can be divided into machine learning, deep learning, lexicon-based, rule-based, topic-based, deep learning, and hybrid approaches. We introduce the main approaches in the following.

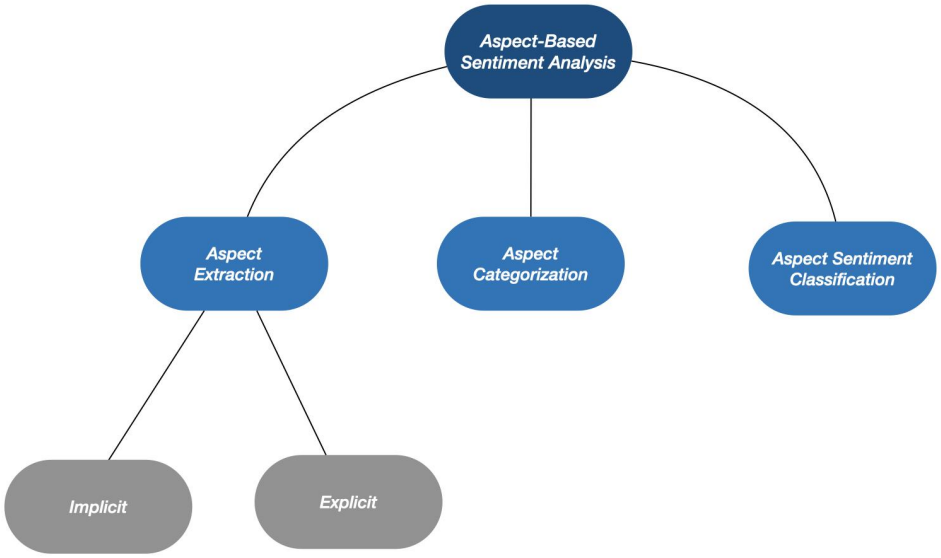

Figure 2.1: The sub-tasks of aspect-based opinion mining

2.2.1 Aspect Extraction

The process of identifying cited characteristics about an entity is called aspect extraction (Khan et al., 2014). The central premise is that an opinion always has a target. A target is often an aspect or topic discussed in the text. Recognizing each expressed opinion and its target in a sentence (Liu, 2012b)

is essential. Among different tasks of ABSA, aspect extraction is the most important task and has been studied by many researchers (Rana and Cheah, 2016).

The techniques for extracting aspects are grouped into four main approaches: Frequency-based, Rule-based, Supervised Learning, Topic Modeling, and Deep Learning. Each approach has a group of techniques that use specific algorithms (Liu, 2012b) (More and Ghotkar, 2016) (Dang et al., 2020). The following is a brief introduction to these approaches.

2.2.1.1 Frequency-based Approach

The Frequency-based approach involves locating nouns, or nominal phrases, frequently in texts as aspects. This approach to extract explicit aspects in a given domain is mainly used. The approach is based on the user commenting on aspects of a particular entity (More and Ghotkar, 2016).

In Hu and Liu (2004), a part-of-speech (POS) tagger identifies noun and noun phrases. Then, the occurrence frequency of these nouns and noun phrases is counted. Finally, a threshold is decided by manual tuning, and only frequent nouns and noun phrases whose count is greater than a threshold are considered. Popescu and Etzioni (2005) used the Pointwise Mutual Information (PMI) similarity measure to compute a score for each candidate based on their frequency and co-occurrence. Blair-Goldensohn et al. (2008) improved the frequency-based approach by considering only noun phrases in an opinion-containing sentence. Ku et al. (2006) used the TF-IDF measure to find frequent terms in reviews that are major topics. Moghaddam and Ester (2010) improved the frequency-based approach by removing non-aspect terms with the help of a syntactic pattern-based filter.

In general, the frequency-based approach has the limitation of missing low-frequency aspects and needing manual configuration and parameter tuning to suit the selected dataset (Ishaq et al., 2020).

2.2.1.2 Relation-based Approach

The Relation-based approach, known as Rule-based and Syntax-based, uses grammatical relation and syntactic patterns between aspect and opinion words to find extraction rules. This approach aims to find a relationship between aspect and opinion words to identify aspects (More and Ghotkar, 2016).

Zhuang et al. (2006) use the dependency parser to identify dependency relations for aspect extraction in reviews. After parsing, a dependency relation links the words in a sentence. In Wu et al. (2009), a phrase dependency parser extracts noun phrases and verb phrases as aspect candidates. A dependency parser helps identify the dependence of individual words, but a phrase depen-

dence parser helps identify the dependence of phrases, which is more accurate for aspect extraction. Qiu et al. (2011) used the double propagation method, in which different relations between opinion words and aspects, opinion words and aspects themselves, are used to design extraction rules. In Poria et al. (2014), a rule-based approach is used to detect both explicit and implicit aspects by exploiting common-sense knowledge and sentence dependency trees. In Rana and Cheah (2017), a Two-fold Rules-based Model (TF-RBM) was proposed, which uses rules defined based on sequential patterns mined from customer reviews and identifies aspects associated with both domain-dependent and independent opinions.

Unlike the Frequency-Based approach, the Rule-Based approach can detect low-frequency aspects. However, More and Ghotkar (2016) argued that this approach might produce many terms which are not real aspects and irrelevant features.

2.2.1.3 Approach based on Supervised Learning

Supervised Learning-based approaches involve building a model from training data and applying it to unlabeled datasets. In this context, identifying aspects, opinions, and their polarity is a labeling problem where patterns are learned from labeled data and applied to unlabeled data.

The most common learning models for aspect extraction are the Hidden Markov Models (HMM) (Rabiner, 1989) and Conditional Random Fields (CRF) (Lafferty et al., 2001) (More and Ghotkar, 2016). Jin and Ho (2009) used lexicalized HMM model to learn patterns to extract aspects and opinion expressions. Jakob and Gurevych (2010) used CRF to train review sentences from different domains for domain-independent extraction. Li et al. (2010) used a variation of CRF, i.e., Skip-CRF and Tree-CRF, to find aspects and opinions. Huang et al. (2012) proposed a CRF-based probabilistic learning model to extract product aspects. Yang and Cardie (2013) formulated the task of opinion entity identification as a sequence labeling problem and employed CRF to learn the probability of a sequence assignment for a given sentence. Other approaches also apply algorithms with SVM, NB, KNN, among others (De Clercq et al., 2015; Guha et al., 2015; Jeyapriya and Selvi, 2015; Pekar et al., 2014).

2.2.1.4 Topic Modeling Approach

The approach called Topic Modeling is considered an unsupervised learning method to discover topics in textual documents considering that documents consist of a mixture of topics and each topic is the probability distribution over the words. That is, documents are considered to have a list of terms related

to a topic and their respective frequency distribution. The probability of a document dealing with one or more topics is seen from the co-occurrence of the frequencies of these terms. The topic modeling helps to group aspects and covers opinion and aspect words (More and Ghotkar, 2016; Wang and Ester, 2014).

Two basic models used are pLSA (Probabilistic Latent Semantic Analysis) (Hofmann, 2013) and LDA (Latent Dirichlet allocation) (Blei et al., 2003). In the context of opinion mining, discovered topics from topic models are aspects. Hence, Topic modeling can be used for aspect extraction. Topic modeling help in aspect grouping and cover both aspect and opinion words.

Zhao et al. (2010) proposed MaxEnt-LDA (Maximum Entropy and LDA combination) hybrid model to discover both aspect words and aspect-specific opinion words jointly. Lin and He (2009) proposed Joint Sentiment Topic (JST) model. JST model considered topics and sentiments together. JST model focuses on the extraction of opinion-aspect pair. Brody and Elhadad (2010) used LDA model to find aspects. After aspect detection, adjectives are selected for opinion word. Jo and Oh (2011) proposed Sentence-LDA (SLDA) model to automate the identification of aspects, assuming words from a single sentence belong to one aspect. The SLDA extends the Aspect and Sentiment Unification Model (ASUM) to identify different opinions. The ASUM model finds results in the form of aspect- opinion pair. Xueke et al. (2013) proposed Joint Aspect/Sentiment Model (JAS). JAS model uses LDA to extract aspect and aspect-related terms. Poria et al. (2016c) extracted implicit aspects through SLDA that integrated common-sense reasoning in the computation of word distributions. Shams and Baraani-Dastjerdi (2017) proposed the Enriched LDA (ELDA) model, which incorporates the aspect co-occurrence relations as prior domain knowledge into LDA for aspect extraction. García-Pablos et al. (2018) proposed W2VLDA, which used LDA combined with an unsupervised pre-trained classification model for aspect identification and separation of opinion words.

2.2.1.5 Deep Learning based Approach

Deep Neural Networks (DNN) based approaches have outperformed traditional or rule-based approaches (Poria et al., 2016a). Recently, DNN, Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and other Neural Networks have been widely used for various Sentiment Analysis tasks, including extraction of aspects.

Most common approaches to aspect extraction include standard and variants of CNN (Poria et al., 2016a; Xu et al., 2016, 2017; Ruder et al., 2016; Toh and Jian, 2016), Long-Short Term Memory (LSTM) (Hochreiter and Schmidhu-

ber, 1997; Liu et al., 2015; Chen et al., 2017; Mai and Le, 2018; Li and Lam, 2017; Jangid et al., 2018; Li et al., 2018; Santos et al., 2021) and Gated Recurrent Unit (GRU) (Cho et al., 2014; Jebbara and Cimiano, 2017; Wang et al., 2017; Cheng et al., 2017; Zeng et al., 2019b). In order to increase the performance of the models, studies have included pre-trained and fine-tuned word embeddings, incorporating linguistic factors in the form of POS and grammatical rules, and exploring concept-based knowledge (Do et al., 2019).

For example, Liu et al. (2015), used RNN with word Embedding to develop a discriminative model for aspect extraction. Similarly, RNN is used in Jebbara and Cimiano (2017) for the Two-Step Aspect-Extraction method. RNN is also applied in Jangid et al. (2018) for the financial domain on tweets and news headlines. CNN is used in both Xu et al. (2016) and Poria et al. (2016b) to carry out aspect-extraction as a multi-label classification problem, using a CNN approach to do automatic extraction of text features. Li et al. (2018) used RNN to extract evaluative sentiment features. Zeng et al. (2019b) proposed an aspect-level sentiment classification model based on a dual memory based on attention mechanism (BMAM) in a GRU-based decoder to achieve Fine-grained feature extraction.

Deep neural language models are currently referred to as state-of-the-art for ABSA Dang et al. (2020). In particular, BERT (Bidirectional Encoder Representations from Transformers) based deep neural language models are widely used for ABSA Song et al. (2019); Zeng et al. (2019a); Rietzler et al. (2019); Karimi et al. (2020).

2.2.2 Aspect Sentiment Classification

Classifying an aspect's opinion involves determining the opinion orientation expressed in each aspect (Liu, 2012b). Polarity classification techniques can be divided into four major groups: a) lexical, with the use of sentiment dictionaries; b) machine learning, with the predominant use of classification or regression techniques; c) statistics, which use techniques to assess the co-occurrence of terms; d) semantics, which define the polarity of words as a function of their semantic proximity to others of known polarity (Tsytsarau and Palpanas, 2011).

The traditional techniques generally fall within two main predominant approaches to the Aspect-based sentiment classification: The supervised Learning approach and the Lexicon-based approach (Liu, 2012b; Tsytsarau and Palpanas, 2011). Deep learning approaches are recently widely used for aspect-based sentiment classification (Wang et al., 2021). Techniques from these different approaches can be combined to improve results.

2.2.2.1 Supervised Learning Approach

In this approach, classification techniques such as Naive Bayes Classifier (NB (Narayanan et al., 2013) (Baskar, 2013), Support Vector Machine (SVM) (Narayanan et al., 2013), decision tree classifier (Narayanan et al., 2013), kNN classifier (Lim, 2004) are used to categorize texts and classify words from opinion about aspects on one of the polarity scales. The problem with supervised learning techniques is that they rely on training data. Therefore, a classifier trained from labeled data in one domain often performs poorly in another domain.

The earliest supervised approach to sentiment classification was proposed by Pang et al. (2002). They used three supervised machine learning approaches: NB, SVM, and Maximum Entropy (ME). Results state that the NB approach outperformed the SVM.

2.2.2.2 Lexicon-based Approach

The methods of this approach are typically unsupervised. The lexicon-based approach uses a sentiment lexicon (which contains a list of sentiment words, phrases, and idioms), composite expressions, rules of opinions, and (possibly) the sentence parse tree to determine the sentiment orientation on each aspect in a sentence (Liu, 2012b).

A lexicon-based method first associates the aspects involved in a sentence with words or phrases. Then, it infers the sentiment polarity of the aspect by analyzing the sentiment polarity of each word or phrase in the lexicon (Liu et al., 2020). In this approach, the lexical sentiment dictionaries carry out the classification. These dictionaries have a set of words and their polarities. In this way, the polarity is assigned if the text has words with the respective polarity. Moreover, dictionaries often have the intensity of each word's sentiment, and the message's sentiment is calculated with the sum of the polarities, considering the weight of each word. If positive terms are greater than negative ones, this technique indicates positive polarity rather than negative (Hu and Liu, 2004; Taboada et al., 2011; Liu et al., 2020).

Hu and Liu (2004) presented a lexicon-based method for aspect-level classification using WordNet (Fellbaum, 1998). Esuli and Sebastiani (2006) and Baccianella et al. (2010) propose using SentiWordNet to classify opinion words in opinion mining.

The lexicon-based method approach can perform well in many cases, but it is insufficient in others. One main shortcoming is that opinion words and phrases do not cover all expressions that imply opinions (Liu and Zhang, 2012).

2.2.2.3 Deep Learning based Approach

Deep learning is a machine learning method that uses algorithms to learn data representations that take the outputs of one layer as inputs to the next layer based on multilayer modules that analyze and classify the inputs.

Deep learning methods treat ABSA as a multiclassification problem, where the sentiment polarity of each aspect is classified into positive, negative, or neutral. However, no handcrafted features are required in deep learning methods; the sentiment polarity of an aspect can be directly identified from end to end.

For deep learning methods to sentiment polarity classification, the methods can be divided into four categories: CNN (Ruder et al., 2016; Du et al., 2016; Wu et al., 2016; Akhtar et al., 2016; Gu et al., 2017; Xu et al., 2017), RNN (Wang et al., 2016a,b; Tay et al., 2018), RecNN (Dong et al., 2014; Nguyen and Shirai, 2015), and MN (Tang et al., 2016). RNN contains basic RNN, LSTM, and GRU.

2.2.3 Aspect Categorization

In natural language text, people often write the same aspect differently. For example, "call quality" and "voice quality" refer to the same aspect for phones. Therefore, aspect expressions need to be grouped into synonymous aspect categories after aspect extraction. Each category represents a unique aspect. The process of grouping aspect expressions into aspect categories (aspects) is called aspect categorization (Liu, 2012b).

2.2.3.1 Cluster Analysis

In cluster analysis, the objective is to organize a set of examples into groups based on proximity measures. As a result, examples from the same group are highly similar but dissimilar to examples from other groups (Wunsch and Xu, 2008). In other words, clustering is based on maximizing the internal similarity of the groups and minimizing the similarity between the groups (Aggarwal and Zhai, 2012). Cluster analysis is also known as observational learning or exploratory data analysis. Examples are organized into groups by observing regularities in the data, without (or with little) human supervision. In clustering processes, there are no predefined classes or labels for training a model, i.e., learning is performed in an unsupervised way (Rokach, 2010).

The choice of proximity measure to define how similar two examples are is fundamental to the clustering task. This choice depends on the dataset's characteristics, mainly the types and scale of the data. For example, there are suitable proximity measures for (1) continuous data, such as Manhanttan,

Euclidean, Pearson, and Cosine; (2) binary data, such as the Jaccard coefficient; and (3) mixing between continuous and binary data, such as the Gower criterion (Gower, 1971). The proximity measures can calculate similarity and dissimilarity (or distance) between examples.

After choosing a proximity measure, a clustering method is selected. Clustering methods can be classified considering different aspects (Marcacini, 2014). In general, clustering strategies can be organized into two types (Rokach, 2010): partitional and hierarchical. In partitional clustering, a set of examples is divided into a simple partition of k groups, while in hierarchical clustering, the examples are organized into groups and subgroups.

In partitional clustering, the objective is to divide the set of examples into k groups, in which k is usually a value previously informed by the user. The K-means algorithm (MacQueen et al., 1967) is the best-known representative for partitional clustering and is widely used in textual data. In K-means, a group representative called centroid is used, which is an average vector computed from the vectors of the group. In this way, the centroid maintains a set of central features of the group, representing all the examples that belong to the group. K-means stop criterion is when there are no more changes in the cluster or a maximum number of iterations. The complexity of K-means is linear in relation to the number of examples, being efficient in different scenarios. However, a disadvantage of the algorithm is the need to inform the number of expected groups previously. In addition, the choice of initial centroids affects the clustering result, which produces variability in the results. A possible solution is to run the K-means with different initializations several times and select the solution with the smallest error (Marcacini, 2014).

Hierarchical clustering algorithms can be agglomerative or divisive (Wunsch and Xu, 2008). In agglomerative hierarchical clustering, initially, each example belongs to a cluster, and in each iteration, the closest pairs of clusters are joined until a single cluster is formed. The divisive hierarchical grouping starts with a group containing all the examples, which is then divided into smaller groups until unitary groups remain (groups with only one example).

The main difference between the agglomerative hierarchical clustering algorithms is the criterion for selecting the pair of closest clusters. The three most popular criteria are (Aggarwal and Zhai, 2012; Marcacini, 2014):

- Single-Link: uses the nearest neighbor criterion, in which the distance between two groups is determined by the distance of the pair of closest examples, with each example belonging to one of these groups.

- Complete-Link: uses the criterion of furthest neighbor, where the distance between two groups is the greatest distance between a pair of examples, with each example belonging to a distinct group.

- Average-Link: the distance between two groups is defined as the average of the distances between all pairs of examples in each group, each pair being composed of one example from each group.

2.2.3.2 Aspect Categorization Techniques

Several techniques were applied to group similar aspects in opinion mining, such as Topic Modeling, K-Means clustering, Expectation-Maximization (EM), and similarity metrics defined using similarity of strings, synonyms, and lexical distances using a lexical database.

Carenini et al. (2005) used the first method to deal with this problem. Their method was based on several similarity metrics measured using WordNet (Fellbaum, 1998). Guo et al. (2009) use multilevel latent semantic analysis to group aspects. Zhai et al. (2011a) uses some information to group expressions into categories for each aspect, such as lexical similarity by WordNet, the similarity of the distribution of words in the corpus, and syntactic constraints. Mukherjee and Liu (2012) use a semi-supervised model for grouping similar aspects using contextual information from the co-occurrence of these terms. Some works (Pavlopoulos and Androutsopoulos, 2014; Zhao et al., 2014) use hierarchical clustering to produce multi-granular summaries, which can be customized according to the user's needs. Toh and Su (2016) trained word embeddings and processed the embedding files by generating K-means clusters from them. Vargas and Pardo (2018) used linguistic resources to provide relations between aspects. Xu et al. (2020) applied K-means clustering for putting sentence embeddings into groups and selecting the center words from clusters as aspects. Topic modeling approach employs techniques, such as Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) (Blei et al., 2003), in order to group similar aspects taking into account the semantic similarity between aspects (Zhai et al., 2011b; Guo et al., 2009).

2.3 Machine Learning for Time Series

Time Series (TS) is a set of observations obtained sequentially over time, as illustrated in 2.2. Thus, a TS Z of size m can be represented as an ordered sequence of observations, i.e., $Z = (z_1, z_2, ..., z_m)$ where $z_t \in \mathbb{R}$ represents an observation z at time t (Chatfield, 2003). The time series can be deterministic or stochastic (Cheng et al., 2015). A series is deterministic when a function represents its data in terms of time $y = f(time)$, i.e., it has a regular and predictable behavior. A stochastic series has an additional term ε, represented by the function $y = f(time, \varepsilon)$, responsible for producing a series of non-regular behavior. Many practical applications are based on stochastic time series,

requiring more robust computational methods to perform the predictive task (Naing and Htike, 2015).

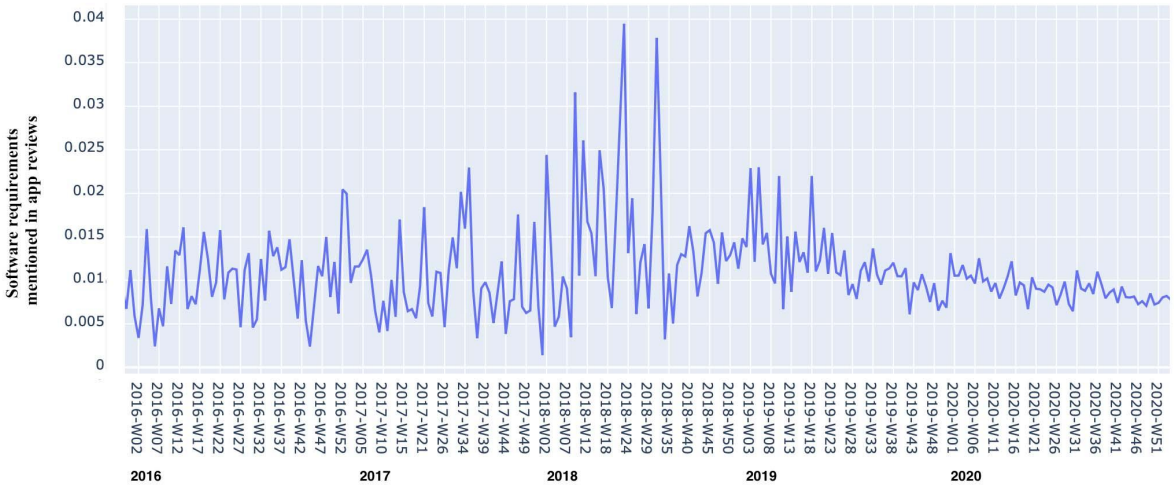

Figure 2.2: Time series (normalized) of mentions of the software requirement "Delivery Time Estimation" in negative reviews of the Foodpanda app

Time series prediction methods can be statistical or based on machine learning. For decades, statistical methods were considered state-of-the-art in this type of task, particularly methods based on moving averages and autoregression. In general, the objective is to identify the coefficients of a model to fit a function to the series data (Box et al., 2015). The most used statistical method in the literature is ARIMA (Autoregressive Integrated Moving Average Models), which considers both the autocorrelation between past and future observations and moving averages to identify the trend (Montgomery et al., 2015). The main characteristic of statistical methods is that they are parametric, e.i., it is necessary to assume a priori that the observations follow a certain distribution. On the other hand, this feature is also a limitation of these models, as it requires specialist knowledge in both the application domain and computational methods (Box et al., 2015).

On the other hand, methods based on machine learning have a differential because they are non-parametric. Among a variety of methods, the use of Neural Networks (MLP - Multilayer Perceptron), Support Vector Machines (SVM), and k-Nearest Neighbors (kNN-TSP) stands out (Ahmed et al., 2010; Chatfield and Xing, 2019). A common point of these methods is that the time series is divided into subseries. Subsets are used as a training set for learning, generally using a regression strategy, in which the class attribute of the learning task is a numerical value. For example, given a substring of the series, its class attribute is the next value of that substring (Rodrigues et al., 2018).

The main components of a time series are the trend, seasonality, and residual. Based on these elements, the TS Z can be formulated, according to

Equations 2.1 and 2.2, by an additive or multiplicative decomposition of its components (Cowpertwait and Metcalfe, 2009). In these equations, T, S, and N correspond to the trend, the seasonality, and the residual at an instant t, respectively.

$$Z_t = T_t + S_t + N_t \tag{2.1}$$

$$Z_t = T_t \cdot S_t \cdot N_t \tag{2.2}$$

In the additive model (2.2), the value of the variable of interest is constituted by the result of the sum of the values of the components, which contemplate the same unit of the observation Z_t. In contrast, in the multiplicative model (2.2), only the trend has the same unit as the investigated variable. The other components exhibit values that can modify the trend, i.e., they assume values greater than, less than, or exactly equal to 1. It is important to note that not always an ST, even when the classical decomposition is considered, will cover the three components mentioned (Parmezan and Batista, 2016).

The trend can be defined as the regular and slowly developed movement along with the series. In other words, this component encompasses a behavior of extensive duration, which can be either increasing or decreasing and assume a wide range of patterns, among which stand out (Parmezan and Batista, 2016): i) Linear Growth: It comprises a constant growth rate for the data, which obeys a linear proportion; ii) Exponential Growth: Characterized by the progressive percentage increase in data over a period of time. The particulars of the growth rate are equivalent to the properties of an exponential function; and iii) Damped Growth: Occurs when the growth rate of future data is less than the current data, such as in situations where the expected growth is 70% of the previous year for a given year.

A behavior that tends to repeat itself at different periods in the TS is known as seasonality. Oscillations along the trend component represent seasonal variations according to a given particularity (Brocklebank and Dickey, 2003). Identifying regularly spaced peaks and troughs, which have a consistent direction and approximately the same magnitude in each cycle, is an important procedure in the subject of TS analysis. The detection of the seasonality component can reveal valuable information, and its removal can highlight useful patterns of TS (Chatfield, 2003). The seasonality component can be categorized into two types: i) Additive Seasonality, where the series presents a stable seasonal fluctuation without considering the global level of the series, and ii) Multiplicative Seasonality is when the size of the seasonal fluctuation varies according to the global level of the series (Parmezan and Batista, 2016).

The residual is represented in TS by random movements caused by fortu-

itous and unexpected events. These events, which are not regular and also do not repeat themselves in a particular pattern, may compromise the results of some studies (Kirchgässner et al., 2013). The TS analysis assumes that the systematic components, i.e., trend and seasonality, are not influenced by stochastic disturbances and can be summarized by deterministic time functions. Therefore, the residual component is what remains after the trend and seasonality components have been estimated and removed from the series (Cowpertwait and Metcalfe, 2009).

2.4 Data-driven Requirements Engineering

Data-driven requirements engineering allows developers, requirements analysts, and managers to systematically use user feedback to support requirements engineering decisions (Maalej et al., 2016). Requirements Engineering (RE) identifies, documents, negotiates, and manages the desired properties and constraints of (Davis, 2003) software systems. Requirements are a verbalization of decision alternatives about the functionality and quality of a system (Aurum and Wohlin, 2003).

RE primarily focuses on engaging system users, capturing their needs, and getting feedback. Conventional RE typically engages users through interviews and workshops. Currently, RE decision-making is usually based on stakeholder intuition and experience, logic (e.g., criteria, options, and arguments), or both (Davis, 2003; Aurum and Wohlin, 2003). Intuition is subjective, potentially inconsistent, and in need of explanation. Logic changes over time and is difficult to capture and externalize. A large amount of user feedback available with the emergence of app stores suggests that exploring the combination of computational intelligence and the human experience is promising (Maalej et al., 2016). A range of information is available from the systematic observation of user reviews to support requirements decisions (Maalej and Pagano, 2011).

The trend is that future research will combine known and used data sources (business requirements, technical and system requirements, stakeholder preferences, and requirements interdependencies) with aggregated user data. These trends suggest a shift towards user-centric and data-driven identification, prioritization, and software requirements management. For this, it is necessary to propose new methods of data analysis to synthesize information and provide insights and constructive recommendations. In particular, predictive modeling, scenario analysis, and adaptive group decision-making technologies are expected to support this process (Maalej et al., 2016).

2.5 Neural Language Model-based Representations

A widespread use of language models is to predict a word (or sentence) given previous words (or sentences) (Mikolov et al., 2013). Recent methods use the idea of distributed representations, in which low-dimensional (real-valued) vectors represent the texts (e.g., words or sentences) (De Mulder et al., 2015). Thus, the correlation or distances between these vectors can help compare the texts capturing syntactic and semantic relationships. Learning well-distributed representations is a challenge, and neural networks are currently used for this task, called Neural Language Models (NLM) (Otter et al., 2021). NLM may be context-free or context-dependent. Context-free NLMs, such as word2vec (Mikolov et al., 2013), map each word, sentence, or document into a static feature vector (e.g., word embeddings).

Context-dependent NLMs, on the other hand, map each word, sentence, or document into a dynamic feature vector, in which the final embedding of a word depends on the sentence in which it occurs (Devlin et al., 2018). Context-dependent NLMs perform better than the traditional language models (Devlin et al., 2018; Otter et al., 2021; Araujo et al., 2022). NLMs may be unidirectional or bidirectional. Unidirectional models analyze the text in a single way, like the reading way. On the other hand, Bidirectional models analyze the text in both ways, from left to right and vice-versa. Bidirectional NLMs perform better than unidirectional NLMs (Devlin et al., 2018; Liu et al., 2019; Sanh et al., 2019; Araujo et al., 2022).

Examples of bidirectional models are BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), and DistilBERT (Sanh et al., 2019). BERT-based models use a masking strategy for training, in which a special token called [MASK] replaces some words in a sentence. The model train to predict the masked words by using non-masked word contexts. During the training stage, BERT-based models use an attention mechanism that learns contextual relations between words (Vaswani et al., 2017a).

The RoBERTa, DistilBERT, and Multilingual DistilBERT models are derivations of the BERT model. These derivations propose improving the results and decreasing the BERT model's computational cost (Devlin et al., 2018)). RoBERTa model uses greater sequences of textual data than the original model (Liu et al., 2019). This model also removes the next sentence prediction goal and dynamically changes the masking pattern (Liu et al., 2019). The DistilBERT model, on the other hand, differs from BERT in the pre-training stage by using a smaller number of parameters, a knowledge distillation technique (training of a compacted model), and the triple loss technique (Sanh et al., 2019). The Multilingual DistilBERT is a DistilBERT model that considers dif-

ferent idioms. Our issue detector proposed in this work uses DistilBERT.

Recently, Generative Pre-trained Transformer (GPT) (Radford et al., 2018, 2019; Brown et al., 2020, 2021) and Open Pre-trained Transformer Language Models (OPT) (Zhang, 2022) techniques for NLP tasks have been widely used.

GPT (Radford et al., 2018) is a multilayer transformer decoder (Vaswani et al., 2017b). Its approach combines two existing ideas: transformers (Vaswani et al., 2017b) and unsupervised pre-training (Howard and Ruder, 2018). GPT is a large auto-regressive language model pre-trained with LM, a simple architecture that can be trained faster than an LSTM-based model. GPT uses the BookCorpus dataset (Zhu et al., 2015), which contains over 7000 books from various genres. It can learn complex patterns in the data by using the attention mechanism. The total number of trained parameters is 110M parameters. GPT-2 (Radford et al., 2019) is a large auto-regressive language model. This large transformer-based language model has 1.5 billion parameters. GPT-2 was trained on a dataset of 8 million web pages (40 GB of Internet text) called WebText (10 × larger than GPT), and it was trained simply to predict the next word (Language Model objective). GPT2 showcased zero-shot task transfer capabilities for various tasks, such as machine translation and reading comprehension. GPT-3 is an autoregressive model proposed by Brown et al. (2020). About 3 trillion words were utilized to train GPT-3. In GPT-3, there are 175 billion parameters. There are ten times more parameters than the MT-NLG language model and 100 times more parameters than GPT-2.

OPT, proposed by Zhang (2022), is a suite of decoder-only pre-trained transformers ranging from 125M to 175B parameters. OPT-175B is trained on public datasets. Results show that OPT-175B performance is comparable to GPT-3 (Brown et al., 2020), requiring only 1/7th of the carbon footprint to develop. To enable researchers to study the effect of scale alone, Zhang (2022) released smaller-scale baseline models, trained on the same dataset and using similar settings as OPT-175B.

Although NLM-based representations have achieved state-of-the-art results in various natural language processing tasks, their use for the textual representation of app reviews is still underexplored (Araujo et al., 2022).

2.6 Related Works

2.6.1 Opinion Mining to Support Requirement Engineering

The opinion mining of app reviews can involve several steps, such as software requirements organization from reviews (Araujo and Marcacini, 2021), grouping similar apps using textual features (Al-Subaihin et al., 2016; Harman et al., 2012), reviews classification in categories of interest to develop-

ers (e.g., Bug and New Features) (Araujo et al., 2020), sentiment analysis of the users' opinion about the requirements (Dragoni et al., 2019; Malik et al., 2020), and the prediction of the review utility score (Zhang and Lin, 2018). The requirements extraction has an essential role in these steps since the failure in this task directly affects the performance of the other steps.

Dabrowski et al. (2020) evaluated the performance of the three state-of-the-art requirements extraction approaches: SAFE (Johann et al., 2017), ReUS (Dragoni et al., 2019) and GuMa (Guzman and Maalej, 2014). These approaches explore rule-based information extraction from linguistic features. GuMa (Guzman and Maalej, 2014) used a co-location algorithm, thereby identifying expressions of two or more words that correspond to a conventional way of referring to things. SAFE (Johann et al., 2017) and ReUS (Dragoni et al., 2019) defined linguistic rules based on grammatical classes and semantic dependence. The experimental evaluation of (Dabrowski et al., 2020) revealed that the low accuracy presented by the rule-based approaches could hinder its use in practice.

Araujo and Marcacini (2021) proposed RE-BERT (Requirements Engineering using Bidirectional Encoder Representations from Transformers) method for software requirements extraction from reviews based on Local Context Word Embeddings (i.e., deep neural language model). RE-BERT models the requirements extraction as a token classification task from deep neural networks. To solve some limitations of rule-based approaches, RE-BERT allows the generation of word embeddings for reviews according to the context of the sentence in which the software requirement occurs. Moreover, RE-BERT explores a multi-domain training strategy to enable software requirements extraction from app reviews of new domains without labeled data.

After extracting requirements from app reviews, there is a step to identify and organize more relevant requirements into groups of similar requirements. Traditionally, requirements obtained from user interviews are prioritized with manual analysis techniques, such as the MoSCoW (Tudor and Walter, 2006) method that categorizes each requirement into groups and applies the AHP (Analytical Hierarchy Process) decision-making (Saaty, 1980). These techniques are unsuitable for prioritizing large numbers of software requirements because they require domain experts to categorize each requirement. Therefore, recent studies have applied data mining approaches and statistical techniques (Pagano and Maalej, 2013).

The statistical techniques have been used to find issues such as to examine how app features predict an app's popularity (Chen and Liu, 2011), to analyze the correlations between the textual size of the reviews and users' dissatisfaction (Vasa et al., 2012), lower rating and negative sentiments (Hoon et al.,

2012), correlations between the rating assigned by users and the number of app downloads (Harman et al., 2012), to the word usage patterns in reviews (Gómez et al., 2015; Licorish et al., 2017), to detect traceability links between app reviews and code changes addressing them (Palomba et al., 2018), and explore the feature lifecycles in app stores (Sarro et al., 2015). Some work also focuses on defining taxonomies of reviews to assist mobile app developers with planning maintenance and evolution activities (Di Sorbo et al., 2016; Ciurumelea et al., 2017; Nayebi et al., 2018b). In addition to user reviews, previous works (Guzman et al., 2016, 2017; Nayebi et al., 2018a) explored how a dataset of tweets can provide complementary information to support mobile app development.

From a labeling perspective, previous works classified and grouped software reviews into classes and categories (Iacob and Harrison, 2013; Galvis Carreño and Winbladh, 2013; Pagano and Maalej, 2013; Mcilroy et al., 2016b; Khalid et al., 2015; Chen et al., 2014; Gómez et al., 2015; Gu and Kim, 2015; Maalej and Nabil, 2015; Villarroel et al., 2016; Nayebi et al., 2017), such as feature requests, requests for improvements, requests for bug fixes, and usage experience. Noei et al. (2021) used topic modeling to determine the key topics of user reviews for different app categories.

These approaches are descriptive models, i.e., they analyze historical data to interpret and understand the behavior of past reviews. In our proposed MApp-Reviews method, we are interested in predictive models that anticipate the growth of negative reviews that can impact the app's evaluation.

These approaches are concerned only with past reviews and acting in a corrective way, i.e., these approaches do not have preventive strategies to anticipate problems that can become frequent and impact more users in the coming days or weeks. Analyzing the temporal dynamics of a requirement from app reviews provides information about a requirement's future behavior.

These current approaches focus solely on addressing past reviews and taking corrective actions. However, they lack preventive strategies to anticipate and mitigate problems that may occur more frequently in the near future, affecting a larger user base. By analyzing the temporal dynamics of app reviews for a specific requirement, we can gain insights into its future behavior. This information is valuable for developing proactive measures to anticipate and address emerging issues.

2.6.2 Issue Prioritization based on App Reviews

Reviews classification is a strategic task for developers to plan actions for the improvement and evolution of the app. These actions may involve: (i) fixing a critical issue; (ii) designing a new feature; and (iii) prioritizing opportunities

and urgent demands from a crowd of users (Al Kilani et al., 2019).

A significant portion of the prior research emphasizes classification and categorization into classes and categories, such as feature requests, requests for improvements, requests for bug fixes, and usage experience (Iacob and Harrison, 2013; Galvis Carreño and Winbladh, 2013; Pagano and Maalej, 2013; Mcilroy et al., 2016b; Khalid et al., 2015; Chen et al., 2014; Gómez et al., 2015; Gu and Kim, 2015; Maalej and Nabil, 2015; Villarroel et al., 2016; Nayebi et al., 2017; Araujo et al., 2022; Herbold et al., 2020; Messaoud et al., 2019; Al Kilani et al., 2019), and determine key topics (Phong et al., 2015; Vu et al., 2016; Gao et al., 2022; Noei et al., 2021).

A study by Messaoud et al. (2019) analyzes which feature classes are relevant for review classification. The authors concluded that different review classes are ambiguous and that removing the Rating class improves the classification model. Similarly, Al Kilani et al. (2019) analyzed the impact of the classification model considering only training subsets labeled with greater confidence by users. Both studies show that the quality of textual representation significantly affects the review classification.

Regarding analyzing emerging issues from app reviews, existing studies are usually based on topic modeling or clustering techniques. For example, LDA (Latent Dirichlet Allocation) (Blei et al., 2003), DIVER (iDentifying emerging app Issues Via usER feedback) (Gao et al., 2019) and IDEA (Gao et al., 2018) approaches were used for app reviews. The LDA approach is a topic modeling method used to determine patterns of textual topics, i.e., to capture the pattern in a document that produces a topic. LDA is a probabilistic distribution algorithm for assigning topics to documents. A topic is a probabilistic distribution over words, and each document represents a mixture of latent topics (Guzman and Maalej, 2014). In the context of mining user opinions in app reviews, especially to detect emerging issues, the documents in the LDA are app reviews, and the extracted topics are used to detect emerging issues. The IDEA approach improves LDA by considering topic distributions in a context window when detecting emerging topics by tracking topic variations over versions (Gao et al., 2018). In addition, the IDEA approach implements an automatic topic interpretation method to label each topic with the most representative sentences and phrases (Gao et al., 2021). In the same direction, the DIVER approach was proposed to detect emerging app issues, but mainly in beta test periods (Gao et al., 2019). The IDEA, DIVER, and LDA approaches have not considered the sentiment of user reviews. Recently, the MERIT (iMproved EmeRging Issue deTection) (Gao et al., 2022) approach was proposed and explored word embedding techniques to prioritize phrases/sentences of each positive and negative topic. Phong et al. (2015) and Vu et al. (2016)

grouped the keywords and phrases using clustering algorithms and then determined and monitored over time the emergent clusters based on the occurrence frequencies of the keywords and phrases in each cluster. Palomba et al. (2015) proposes an approach to tracking informative user reviews of source code changes and monitoring the extent to which developers address user reviews.

Recently, a proposal regarding issue prioritization was developed by Pilliang et al. (2022), which proposes a risk matrix model for software development projects. By employing BERT for sentence embedding, K-Means for grouping and frequency calculation, and TextBlob for sentiment analysis, the model determines risk priorities and quantifies their probability and impact.

2.6.3 Comparison of Related Works

In short, app reviews formed the basis for many studies and decisions ranging from feature extraction to release planning of mobile apps.

However, when analyzing the temporal evolution of issues regarding software requirements, previous related works focused on extracting aspects do not explore the temporal dynamics with a predictive software requirements model in reviews. Related works that incorporate temporal dynamics cover only descriptive models, as shown in Table 2.1. Concerning works that focus on identifying bugs, related works that approach risk classification use a supervised approach, as shown in Table 2.2.

Lin et al. (2022) provided a systematic literature review that analyzed 185 primary studies related to opinion mining for software development. Their review presents studies that focused on identifying issues in app reviews (e.g., feature request, problem discovery, user experiences) (Lin et al., 2022). However, few primary studies focused on identifying bug reviews.

In Table 2.2, we present the works related with an emphasis on issue detection and prioritization, in particular, bug reporting, and compare with the MApp-IDEA method.

2.7 Final remarks

Despite the related studies presenting relevant strategies, existing studies focus on only a few steps of the opinion-mining process from app reviews, which hinders its use in real-world scenarios. None of the existing proposals comprehensively cover the automated issue analysis process, nor do they encompass the process from start to finish through an unsupervised approach. Our opinion mining process implements an entire automated issue analysis, including automatic issue detection, risk-based prioritization, temporal moni-

Table 2.1: Summary table of related works focused on extracting aspects (e.e., software features/requirements)

Reference	Data Representation	Pre-processing and Extraction of Requirements	Requirements/Topics Clustering and Labeling	Temporal Dynamics	Year
(Araujo and Marcacini, 2021)	Word embeddings.	Token Classification.	No.	No.	2021
(Gao et al., 2022)	Word embeddings.	Rule-based and Topic modeling.	Yes. It combines word embeddings with topic distributions as the semantic representations of words.	Yes. Descriptive Model.	2022
(Malik et al., 2020)	Bag-of-words.	Rule-based.	No.	No.	2020
(Gao et al., 2019)	Vector space.	Rule-based and Topic modeling.	Yes. Anomaly Clustering Algorithm.	Yes. Descriptive model.	2019
(Dragoni et al., 2019)	Dependency tree.	Rule-based.	No.	No.	2019
(Gao et al., 2018)	Probability vector.	Rule-based and Topic modeling.	Yes. AOLDA - Adaptively Online Latent Dirichlet Allocation. The topic labeling method considers the semantic similarity between the candidates and the topics.	Yes. Descriptive model.	2018
(Johann et al., 2017)	Keywords.	Rule-based.	No.	No.	2017
(Vu et al., 2016)	Word embeddings.	Pre-defined.	Yes. Soft Clustering algorithm that uses vector representation of words from Word2vec.	Yes. Descriptive model.	2016
(Villarroel et al., 2016)	Bag-of-words.	Rule-based.	Yes. DBSCAN clustering algorithm. Each cluster has a label composed of the five most frequent terms.	No.	2016
(Gu and Kim, 2015)	Semantic Dependence Graph.	Rule-based.	Yes. Clustering aspect-opinion pairs with the same aspects.	Yes. Descriptive model.	2015
(Phong et al., 2015)	Vector space.	Rule-based.	Yes. Word2vec and K-means.	Yes. Descriptive model.	2015
(Guzman and Maalej, 2014)	Keywords.	Rule-based and Topic modeling.	Yes. LDA approach.	No.	2014
(Chen et al., 2014)	Bag-of-words.	Topic modeling.	Yes. LDA and ASUM approach with labeling.	Yes. Descriptive model.	2014
(Iacob and Harrison, 2013)	Keywords.	Rule-based and Topic modeling.	Yes. LDA approach.	No.	2013
(Galvis Carreño and Winbladh, 2013)	Bag-of-words.	Topic modeling.	Yes. Aspect and Sentiment Unification Model (ASUM) approach.	No.	2013
(Harman et al., 2012)	Keywords.	Pre-defined.	Yes. Greedy-based clustering algorithm.	No.	2012
(Palomba et al., 2018)	Bag-of-words.	Topic-modeling.	Yes. AR-Miner approach with labeling.	No.	2018
MApp-Reviews	Word embeddings.	Token Classification.	Yes. K-means.	Yes. predictive models	This study

Table 2.2: Summary table of related work concerning issue detection and prioritization, with a focus on bug reporting

Author	Software Document	Classification Method	Category	Data Representation	Risk Classification	Year
Messaoud et al. (2019)	Review	Multi-label Active Learning with Auxiliary Learner (Logistic Regression, SVM)	Feature requests / Bug reports / User experiences	Bag-of-words	No	2019
Al Kilani et al. (2019)	Review	Supervised Approaches: RF, NB, NBM, SVM	Bug /Usability/ New Feature/ Performance/ Security	Bag-of-words	No	2019
Herbold et al. (2020)	Issue	Supervised Approaches: Random Forest (RF), Naïve Bayes (NB), fastText	Bug / Non-Bug	Bag-of-words, word embedding	No	2020
Araujo et al. (2022)	Review	Supervised Approaches: k-Nearest Neighbors (kNN), Multinomial NB (MNB), Support Vector Machines (SVM), Multilayer Perceptron (MLP), and One-Class SVM (OCSVM)	Feature requests / Bug report / User experiences/ Irrelevant	Bag-of-words, Neural Language Models	No	2022
Gao et al. (2022)	Review	Unsupervised Approaches: BTM	Issue / Non-issue	Bag-of-words, Word embedding	No	2022
Malgaonkar et al. (2022)	Review	Supervised Approaches: NBM and Multi-criteria heuristic function	Useful / Non-useful	Bag-of-words	Yes	2022
Pilliang et al. (2022)	Question	BERT and K-means	Issue Risk (high, medium and low)	Neural Language Models	Risk Matrix	2022
MApp-IDEA	Review	Unsupervised Approach	Issue Risk (high, medium and low)	Neural Language Models	Risk Matrix	This study

toring of issues and risks, and utilizing predictive models for identifying faulty feature trends.

In response to these limitations, our proposal introduces a comprehensive opinion-mining process to address these gaps in related work. Our proposal introduces a two-fold approach: (i) MApp-Reviews, extracting software requirements and integrating their temporal dynamics into predictive models, and (ii) MApp-IDEA, detecting, monitoring, and prioritizing emerging issues while also performing risk classification. Additionally, we present a risk matrix construction approach from app reviews using the recent Large Language Models (LLMs).

CHAPTER

3

Temporal Dynamics of Requirements Engineering from Mobile App Reviews

3.1 Introduction

In this chapter, we present the MAPP-Reviews (Monitoring App Reviews) method. MAPP-Reviews explores the temporal dynamics of software requirements extracted from app reviews. 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.

The remainder of this chapter is organized as follows: Section 3.2 summarizes the research objectives, outlining the specific goals to be achieved. Section 3.3 highlights the main contributions of the work, discussing the novel insights and advancements made in the field. Section 3.4 introduces the MApp-Reviews method, presenting its architecture and key components. Section 3.5 focuses on the MAPP-Reviews in action, which includes the five stages: App Reviews, Requirements Extraction, Requirements Clustering, Time Series Generation, and Predictive Models. In Section 3.6, the Experimental Eval-

uation is conducted, encompassing the definition of research questions, the datasets used, the experimental setup, the results obtained, the discussion of the findings, and the limitations encountered. Finally, Section 3.7 concludes the chapter with final remarks, summarizing the key points discussed and offering insights for future research.

3.2 Summary of Objectives

The main objective of MAPP-Reviews is to explore the temporal dynamics of software requirements extracted from app reviews:

- Collect, pre-process and extract candidate software requirements from large review datasets.
- Using the clustering technique, organize the software requirements associated with negative reviews into groups according to their content similarity.
- A time series models the temporal dynamics of each requirement group, which indicates the time-frequency of a software requirement from negative reviews.
- Train predictive models on historical time series to forecast future points.
- Identify requirements with higher negative evaluation trends to mitigate the negative impact.

3.3 Main Contributions

Our main contributions presented in this chapter are briefly summarized below:

- 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.

- 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.

- 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.

3.4 MApp-Reviews Method Architecture

To analyze the temporal dynamics of software requirements, we present the MAPP-Reviews approach with five stages, as shown in Figure 3.1. First, we collect mobile app reviews in app stores through a web crawler. Second, we extract candidate software requirements from these reviews. Third, we group similar extracted requirements by using clustering methods. Fourth, the most relevant clusters are identified to generate time series from negative reviews. Finally, we train the predictive model from time series to forecast software requirements involved with negative reviews, potentially impacting the app's rating.

3.5 MApp-Reviews in Action

3.5.1 App Reviews

The app stores provide the textual content of the reviews, the publication date, and the rating stars of user-reported reviews. In the first stage of MAPP-Reviews, raw reviews are collected from the app stores using a web crawler tool through a RESTful API. At this stage, there is no pre-processing in the textual content of reviews. Data is organized in the appropriate data structure and automatically batched to be processed by the requirements extraction stage of MAPP-Reviews.

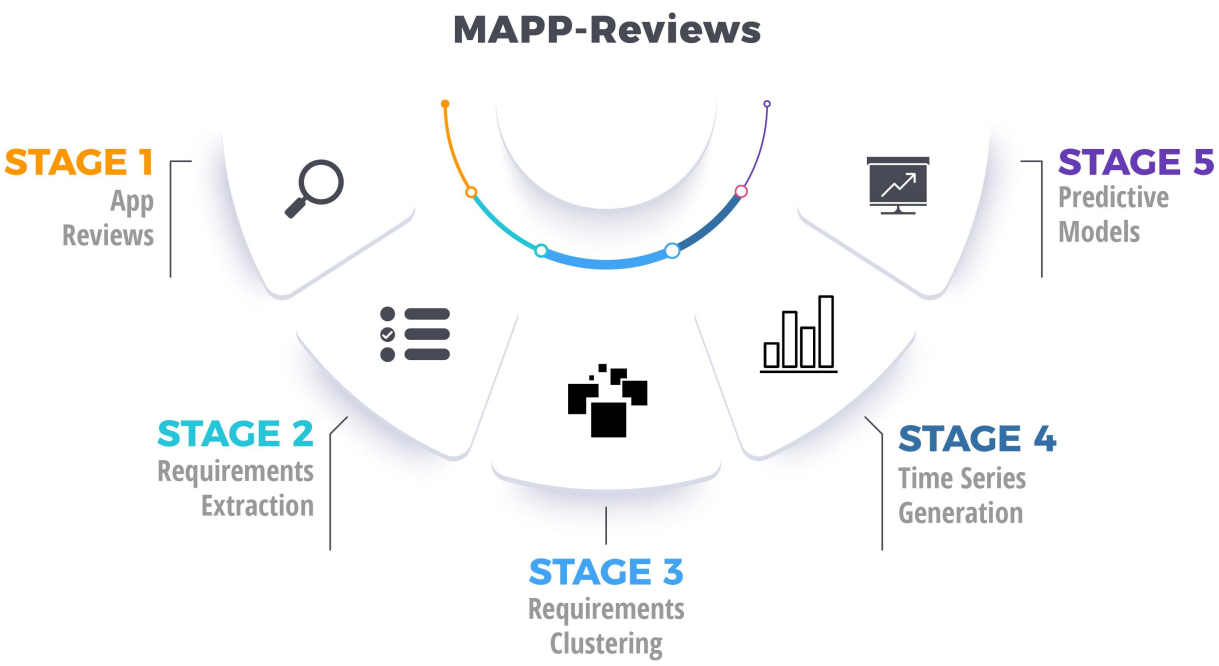

Figure 3.1: Overview of the proposed method for analyzing temporal dynamics of requirements engineering from mobile app reviews

3.5.2 *Requirements Extraction*

In this section, we focus on stage 2 of the MAPP-Reviews method, which involves the extraction of candidate software requirements from app reviews and the subsequent text pre-processing using contextual word embeddings.

MAPP-Reviews uses the pre-trained RE-BERT (Araujo and Marcacini, 2021) 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 Dabrowski et al. (2020). The reviews are from 8 apps of different categories as showed in Table 3.1. 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 (Johann et al., 2017), ReUS (Dragoni et al., 2019) and GuMa (Guzman and Maalej, 2014). 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 3.2 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.

Table 3.1: Statistics about the datasets from 8 apps of different categories used to train the RE-BERT model.

	eBay	Evernote	Facebook	Netflix	Photo editor	Spotify	Twitter	WhatsApp
Reviews	1,962	4,832	8,293	14,310	7,690	14,487	63,628	248,641
Category	Shopping	Productivity	Social	Entertainment	Photography	Music and Audio	Social	Communication

Figure 3.2: Example of a review and extracted requirements

RE-BERT returns the probability that each token (e.g. word) is a software requirement. Consecutive tokens in a sentence are concatenated to obtain software requirements expressions composed of two or more tokens. We filter reviews that are more associated with negative comments through user feedback. Consider that the user gives a star rating when submitting a review for an app. Generally, the star rating ranges from 1 to 5. This rating can be considered as the level of user satisfaction. In particular, we are interested in defective software requirements, and only reviews with 1 or 2 rating stars were considered. Thus, we use RE-BERT to extract only software requirements mentioned in reviews that may involve complaints, bad usage experience, or malfunction of app features.

RE-BERT extracts software requirements directly from the document reviews and we have to deal with the drawback that the same requirement can be written in different ways by users. Thus, we propose a software requirement semantic clustering, in which different writing variations of the same requirement must be standardized. However, the clustering step requires that the texts be pre-processed and structured in a format that allows the calculation of similarity measures between requirements.

We represent each software requirement through contextual word embedding. Word embeddings are vector representations for textual data in an embedding space, where we can compare two texts semantically using similarity measures. Different models of word embeddings have been proposed, such as Word2vec (Mikolov et al., 2013), Glove (Pennington et al., 2014), FastText (Bojanowski et al., 2017) and BERT (Devlin et al., 2018). We use the BERT Sentence-Transformers model (Reimers and Gurevych, 2019) to maintain an neural network architecture similar to RE-BERT. BERT is a contextual neural language model, where for a given sequence of tokens, we can learn a word embedding representation for a token. Word embeddings can calculate the semantic proximity between tokens and entire sentences, and the embeddings can be used as input to train the classifier. BERT-based models are promising to learn contextual word embeddings from long-term dependencies between tokens in sentences and sentences (Araujo and Marcacini, 2021). However, Araujo and Marcacini (2021) highlight that a local context more impacts the extraction of software requirements from reviews, i.e., tokens closer to those of software requirements are more significant (Araujo and Marcacini, 2021). Therefore, RE-BERT explores local contexts to identify relevant candidates for software requirements. Formally, let $E = \{r_1, r_2, ..., r_n\}$ be a set of n extracted software requirements, where $r_i = (t_1, ..., t_k)$ are a sequence of k tokens of the requirement r_i. BERT explore a masked language modeling procedure, i.e., 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] (Araujo and Marcacini, 2021). One of the training objectives is the noisy reconstruction defined in Equation 3.1,

$$p(\bar{r}|\hat{r}) = \sum_{j=1}^{k} m_j \frac{exp(\mathbf{h}_{c_j}^{\top} \mathbf{w}_{t_j})}{\sum_{t'} exp(\mathbf{h}_{c_j}^{\top} \mathbf{w}_{t'})} \tag{3.1}$$

where $\hat{r}$ is a corrupted token sequence of requirement r, $\bar{r}$ 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 token t_j, 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 $p(\bar{r}|\hat{r})$ of the Equation 3.1. Figure 3.3 illustrates a set of software requirements in a two-dimensional space obtained from contextual word embeddings. Note that the vector space of embeddings preserves the proximity of similar requirements, but written in different ways by users such as "*search items*", "*find items*", "*handles my searches*" and "*find special items*".

Figure 3.3: Set of software requirements in a two-dimensional space obtained from contextual word embeddings

3.5.3 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.

Formally, let $R = \{r_1, r_2, ..., r_n\}$ a set of extracted software requirements, where each requirement r is a m-dimensional real vector from an word embedding space. The K-means clustering aims to partition the n requirements into k $(2 \leq k \leq n)$ clusters $C = \{C_1, C_2, ..., C_k\}$, thereby minimizing the within-cluster sum of squares as defined in Equation 3.2, where μ_i is the mean vector of all requirements in C_i.

$$\sum_{C_i \in C} \sum_{r \in C_i} \|r - \mu_i\|^2 \tag{3.2}$$

We observe that not all software requirements cluster represents a functional requirement in practice. Then, we evaluated the clustering model using a statistical measure called silhouette (Rousseeuw, 1987) to discard clusters with many different terms and irrelevant requirements. The silhouette value of a data instance is a measure of how similar a software requirement is to

its own cluster compared to other clusters. The silhouette measure ranges from −1 to +1, where values close to +1 indicate that the requirement is well allocated to its own cluster (Vendramin et al., 2010). Finally, we use the requirements with higher silhouette values to support the cluster labeling, i.e., to determine the software requirement's cluster name. For example, Table 3.2 shows the software requirement cluster "Payment" and some tokens allocated in the cluster with their respective silhouette values.

Table 3.2: Example of software requirement cluster "Payment" and some tokens allocated in the cluster with their respective silhouette values.

Cluster Label	**Tokens with Silhouette (s)**
Payment	"payment getting" (s = 0.2618), "payment get" (s = 0.2547), "getting payment" (s = 0.2530), "take payment"(s = 0.2504), "payment taking" (s = 0.2471), "payment" (s = 0.2401)

To calculate the silhouette measure, let $r_i \in C_i$ a requirement r_i in the cluster C_i. Equation 3.3 compute the mean distance between r_i and all other software requirements in the same cluster, where $d(r_i, r_j)$ is the distance between requirements r_i and r_j in the cluster C_i. In the equation, the expression $\frac{1}{|C_i|-1}$ means the distance $d(r_i, r_i)$ is not added to the sum. A smaller value of the silhouette measure $a(i)$ indicates that the requirement i is far from neighboring clusters and better assigned to its cluster.

$$a(r_i) = \frac{1}{|C_i| - 1} \sum_{r_j \in C_i, r_i \neq r_j} d(r_i, r_j) \tag{3.3}$$

Analogously, the mean distance from requirement r_i to another cluster C_k is the mean distance from r_i to all requirements in C_k, where $C_k \neq C_i$. For each requirement $r_i \in C_i$, Equation 3.4 defines the minimum mean distance of r_i for all requirements in any other cluster, of which r_i is not a member. The cluster with this minimum mean distance is the neighbor cluster of r_i. So this is the next best-assigned cluster for the r_i requirement. The silhouette (value) of the software requirement r_i is defined by Equation 3.5.

$$b(r_i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{r_j \in C_k} d(r_i, r_j) \tag{3.4}$$

$$s(r_i) = \frac{b(r_i) - a(r_i)}{\max\{a(r_i), b(r_i)\}}, \text{ if } |C_i| > 1 \tag{3.5}$$

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. In addition, each cluster has a representative text (cluster label) obtained according to the re-

quirements with higher silhouette values.

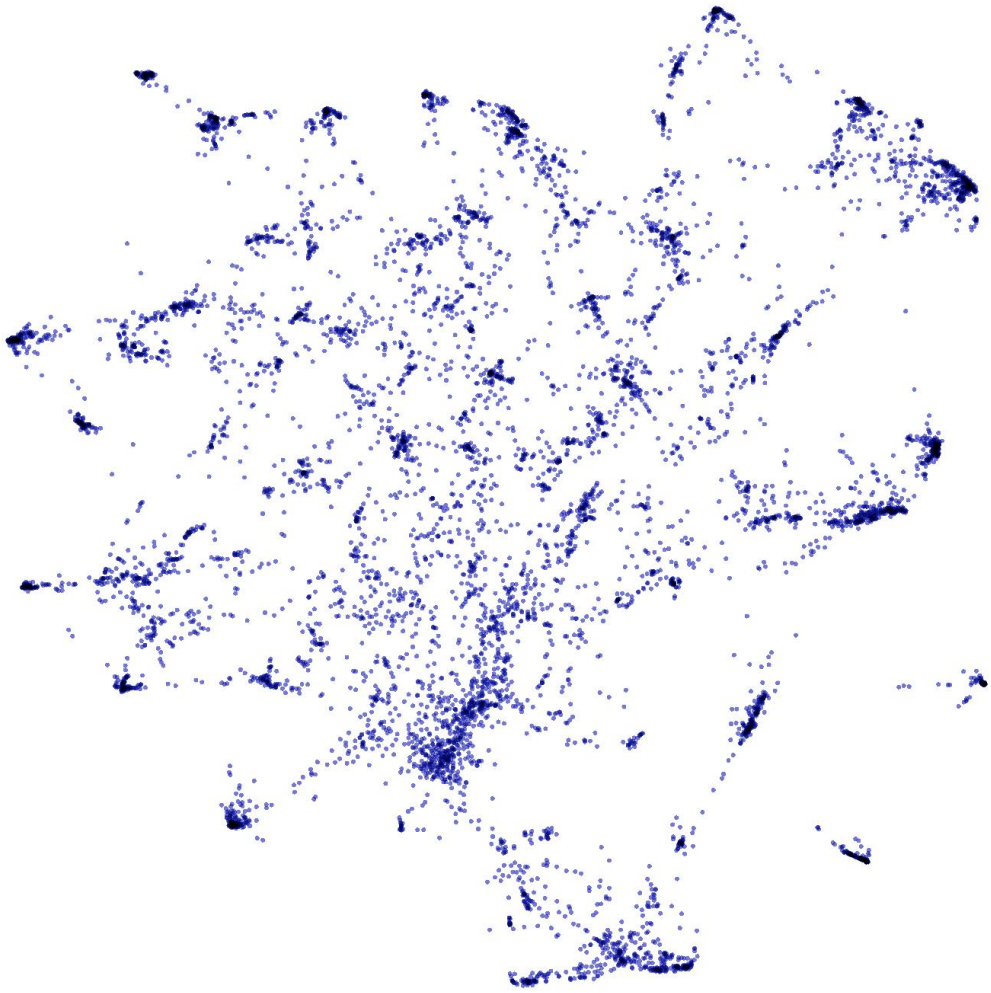

Figure 3.4: Two-dimensional projection of clustered software requirements from approximately 86,000 food delivery app reviews

Figure 3.4 shows a two-dimensional projection of clustered software requirements from approximately 86,000 food delivery app reviews, which were used in the experimental evaluation of this work. High-density regions represent clusters of similar requirements that must be mapped to the same software requirement during the analysis of temporal dynamics.

In the next section, techniques for generating the time series from software requirements clusters are presented, as well as the predictive models to infer future trends.

3.5.4 Time Series Generation

Time series can be described as an ordered sequence of observations (Chatfield and Xing, 2019). 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.

Some software requirements are naturally more frequent than others, as well as the tokens used to describe these requirements. For the time series analysis to be compared uniformly, we generate a normalized series for each requirement. Each observation in the time series is normalized according to Equation 3.6,

$$x_{normalized} = \frac{x}{z_p} \tag{3.6}$$

where $x_{normalized}$ is the result of the normalization, where x is the frequency of cluster (time series observation) C in the period p, z_p is the total frequency of the period.

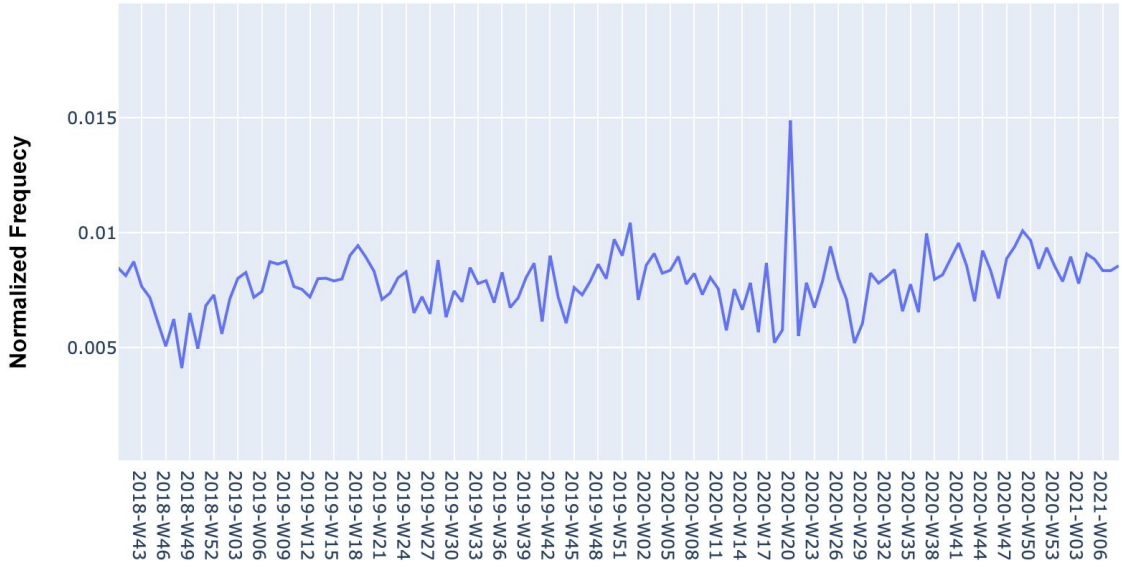

Figure 3.5: Time series with the normalized frequency of “Arriving time” requirement from Zomato App in negative reviews

Figure 3.5 shows an example of one of the generated time series for a software requirement. The time dynamics represented in the time series indicate the behavior of the software requirement concerning negative reviews. Note that there are significant increases in the mention of the requirement in some periods, thereby indicating that users have negatively evaluated the app for that requirement. Predicting the occurrence of these periods for software maintenance, aiming to minimize the number of future negative reviews, is the objective of the MAPP-Reviews predictive model discussed in the next section.

3.5.5 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 (Taylor and Letham, 2018). 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. The Prophet model consists of three main components, as defined in Equation 3.7,

$$y(t) = g(t) + s(t) + h(t) + t_\varepsilon \tag{3.7}$$

where $g(t)$ represents the trend, $s(t)$ represents the time series seasonality, $h(t)$ represents significant events that impacts time series observations, and the error term t_ε represents noisy data.

A time series can be divided into training and testing during model training. The terms $g(t)$, $s(t)$, and $h(t)$ can be automatically inferred by classical statistical methods in the area of time series analysis, such as the Generalized Additive Model (GAM) (Hastie and Tibshirani, 1987) used in Prophet. In the training step, the terms are adjusted to find an additive model that best fits the known observations in the training time series. Next, we evaluated the model in new data, i.e., the testing time series.

In the case of the temporal dynamics of the software requirements, domain knowledge is represented by specific points (e.g., changepoints) in the time series that indicate potential growth of the requirement in negative reviews. Figure 3.6 shows the forecasting for a software requirement. Original observations are the black dots, and the blue line represents the forecast model. The light blue area is the confidence interval of the predictions. The vertical dashed lines are the time series changepoints.

Changepoints play an important role in forecasting models, as they represent abrupt changes in the trend. changepoints can be estimated automatically during model training. However, domain knowledge, such as the date of app releases, marketing campaigns, and server failures, are changepoints

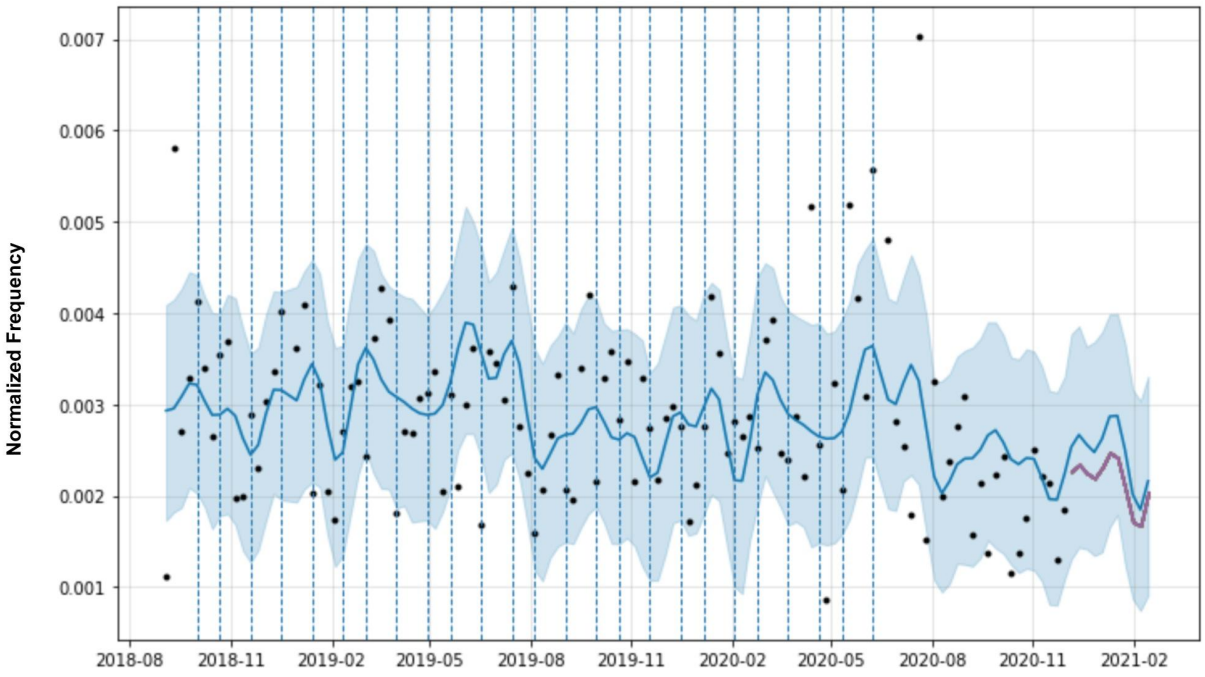

Figure 3.6: Prophet forecasting with automatic changepoints of a requirement

that software engineers can manually add. Therefore, the analyst could specify the changepoints using known dates of product launches and other growth-altering events or may be automatically selected given a set of candidates. In MAPP-Reviews, we have two options for selecting changepoints in the predictive model. The first option is automatic changepoint selection, where the Prophet specifies 25 potential changepoints, which are uniformly placed in the first 80% of the time series. The second option is the manual specification with a set of dates provided by a domain analyst. In this case, the changepoints could be entirely limited to a small set of dates. Suppose no available dates are provided by default. In that case, we use the most recent observations, which have a value greater than the average of the observations, i.e., we want to emphasize the highest peaks of the time series, as they indicate critical periods of negative revisions from the past.

In the experimental evaluation, we show the MAPP-Review's ability to predict perceptually important points in the software requirements time series, allowing the identification of initial trends in defective requirements to support preventive strategies in software maintenance.

Table 3.3 shows an emerging issue being predicted 6 weeks in advance in the period from October 2020 to January 2021. The table presents a timeline represented by the horizon (h) in weeks, with the volume of negative raw reviews ($Vol.$). An example of a negative review is shown for each week until reaching the critical week (peak), with $h = 16$. The table row with $h = 10$ highlighted in bold shows when MAPP-Reviews identified the uptrend. In this case, we show the MAPP-Reviews alert for the "*Time of arrival*" requirement of the

Uber Eats app. In particular, the emerging issue identified in the negative reviews is the low accuracy of the estimated delivery time in the app. The text of the user review samples has been entered in its entirety without any pre-treatment. Figure 3.7 shows a graphical representation of this prediction.

Table 3.3: Example of emerging issue prediction alert for the "*Time of arrival*" requirement of the Uber Eats app reviews triggered by MAPP-Reviews.

h	Vol.	Token	Review
1	768	Delivery time	*Listed delivery times are inaccurate majority of the time.*
2	849	Time frame	*This app consistently gives incorrect, shorter delivery time frame to get you to order, but the deliveries are always late. The algorithm to predict the delivery time should be fixed so that you'll stop lying to your customers.*
3	896	Arrival time	*Ordered food and they told me it was coming. The wait time was supposed to be 45 minutes. They kept pushing back the arrival time, and we waited an hour and 45 minutes for food, only to have them CANCEL the order and tell us it wasn't coming. If an order is unable to be placed you need to tell customers BEFORE they've waited almost 2 HOURS for their food.*
4	1247	Delivery time	*The app was easy to navigate but the estimated delivery time kept changing and it took almost 2hrs to receive food and I live less than 4 blocks away pure ridiculousness if I would of know that I would of just walked there and got it.*
5	1056	Estimated time	*Everyone cancels and it ends up taking twice the estimated time to get the food delivered. You dont get updated on delays unless you actively monitor. Uber has failed at food delivery.*
6	997	More time	*Uber Eats lies. Several occasions showed delays because "the restaurant requested more time" but really it was Uber Eats unable to find a driver. I called the restaurants and they said the food has been ready for over an hour!*
7	939	Delivery time	*Your app is unintuitive. Delivery times are wildly inaccurate and orders are canceled with no explanation, information or help.*
8	854	Estimated time	*This service is terrible. Delivery people never arrive during the estimated time.*
9	994	Time	*Delivery times increase significantly once your order is accepted. 25-45 mins went up to almost 2 hours! Not easy to cancel. Also one restaurant that looked available said I was too far away after I had filled my basket. Other than that the app is easy to use.*
10	**1257**	**Time estimate**	***Use door dash or post mates, uber eats has definitely gone down in quality. Extremely inaccurate time estimates and they ignore your support requests until its to late to cancel an order and get a refund.***
11	1443	Delivery time	*Delivery times are constantly updated, what was estimated at 25-35 minutes takes more than two hours. I understand it's just an estimate, but 4X that is ridiculous.*
12	1478	Delivery time	*Inaccurate delivery time*
13	1376	Estimated time	*Used to use this app a lot. Ever since they made it so you have to pay for your delivery to come on time the app is useless. You will be stuck waiting for food for an hour most of the time. The estimated time of arrival is never accurate. Have had my food brought to wrong addresses or not brought at all. I will just take the extra time out of my day to pick up the food myself rather than use this app.*
14	1446	Estimated time	*Terrible, the estimated time of arrival is never accurate and has regularly been up to 45 MINUTES LATE with no refund. Doordash is infinitely better, install that instead, it also has more restaurants*
15	1354	Estimed time	*App is good but this needs to be more reliable on its service. the estimated arrival time needs to be matched or there should be a option to cancel the order if they couldnt deliver on estimated time. Continuesly changing the estimated delivery time after the initial order confirmation is inappropriate.*
16	1627	Estimed time	*I use this app a lot and recently my order are always late at least double the time im originally quoted. Every time my food is cold. Maybe the estimated time should be adjusted to reflect what the actual time may be.*

In our proposal, we use the Prophet model to forecast time series. However, we will investigate other time series prediction methods to improve the accuracy of the results.

3.6 Experimental Evaluation

In this section, we present an evaluation and discussion of the key findings from the MAPP-Reviews method. 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.

3.6.1 Definition of Research Questions

We raise the following research question: how do we predict initial trends on defective requirements from users' opinions before negatively impacting the overall app's evaluation?

3.6.2 Datasets

For this experimental evaluation, we used a dataset with 86,610 reviews of three popular food delivery apps from the Google Play store: Uber Eats, Foodpanda, and Zomato. These apps were chosen based on their popularity and the number of reviews available. Furthermore, these apps represent a dynamic and complex environment consisting of restaurants, food consumers, and drivers operating in highly competitive conditions (Williams et al., 2020). In addition, this environment means a real scenario of commercial limitations, technological restrictions, and different user experience contexts, which makes detecting emerging issues early an essential task.

The dataset was obtained in the first stage (App Reviews) of MAPP-Reviews and is available at https://github.com/vitormesaque/mapp-reviews. The reviews are from September 2018 to January 2021.

3.6.3 Experimental Setup

Initially, we iterated the first step of the method to collect only negative app reviews. Then, the second step extracts the software requirements using the negative reviews with the RE-BERT tool. After extracting the software requirements, we again iterate the first stage to collect all reviews regardless of your rating, i.e., reviews with all ratings. At this point, 149.635 requirements were extracted, excluding duplicates. After that, we run the clustering stage with k=300, count how many times each cluster appears in reviews, and rank the most frequent clusters. We consider all reviews and not just negative reviews to calculate this amount.

As a result, we obtain a time series that captures the evolution of each requirement over time. In order to focus on the more significant clusters, we discard the time series of clusters with low volumes, i.e., infrequent clusters.

The most frequent clusters are identified and labeled based on the silhouette value of the requirements assigned to each cluster. Subsequently, we generate a normalized time series by app and cluster.

In stage four, the general idea is to use time series points to estimate the following points. However, in our context would not be feasible to predict the following months as it is tough to find a correlation between what happens today and what will happen in the next few months regarding bug reports. Therefore, it makes sense for our experiment to make predictions at the weekly level. That way, we can take problems starting to happen and predict whether they will get serious in the coming weeks.

We then discard the time series data before week 37 of the year 2018 ($2018W37$) so that all series start at the same point. Then, we train the model with n previous points for each time series and perform forecasts with the horizon ranging from $h = 1$ to $h = 4$, i.e., we want to forecast the next four weeks. We evaluate the MAPE prediction error (average of all predictions) for each requirement/cluster, organized by prediction horizon (h). After that, we calculate the points/periods that show a significant increase in the time series.

We utilize the Prophet model to perform time series forecasting, considering a range of horizon values h from 1 to 4. This analysis involves the incorporation of both automatic and custom changepoints. The custom changepoints, denoted as cp, are determined based on the predicted values y of the training set. Specifically, data points in the training set with y values exceeding the mean plus the standard deviation of all y values are identified as custom cp. These custom changepoints play a significant role in capturing significant deviations and trends in the time series data, enhancing the accuracy of the forecasting process.

The forecast with the Prophet was made with the following parameters: $growth = \text{“}linear\text{''}, weekly\_seasonality = True, daily\_seasonality = True, yearly\_seasonality = True$.

3.6.4 Results

After the software requirements extraction and clustering stage (with $k = 300$ clusters), the six most popular (frequent) requirements clusters were considered for time series prediction. The following software requirements clusters were selected: “Ordering”, “Go pick up”, “Delivery”, “Arriving time”, “Advertising”, and “Payment”. The requirements clusters are shown in Table 3.4 with the associated words ordered by silhouette.

In the MAPP-Reviews prediction stage, we evaluated two scenarios using Prophet. The first scenario is the baseline, where we use the automatic parameters fitting of the Prophet. By default, Prophet will automatically detect

Table 3.4: Software requirements clusters for food delivery apps used in the experimental evaluation. Tokens well allocated in each cluster (silhoutte measure) were selected to support the cluster labeling.

Cluster Label	**Tokens with Silhouette values (s)**
Ordering	"ordering" ($s = 0.1337$), "order's" ($s = 0.1250$), "order from" ($s = 0.1243$), "order will" ($s = 0.1221$), "order" ($s = 0.1116$), "the order", ($s = 0.1111$)
Go pick up	"go pick up"($s = 0.1382$)", "pick up the" ($s = 0.1289$)", "pick up at", ($s = 0.1261$), "to take" ($s = 0.1176$), "go get" ($s = 0.1159$)
Delivery	"delivering parcels" ($s = 0.1705$), "delivery options" ($s = 0.1590$), "waive delivery" ($s = 0.1566$), "delivery charges" ($s = 0.1501$), "accept delivery" ($s = 0.1492$)
Arriving time	"arrival time" (s=0.3303), "waisting time" ($s = 0.3046$), "arriving time" ($s = 0.3042$), "estimate time" ($s = 0.2877$), "delievery time" ($s = 0.2743$)
Advertising	"anoyning ads" ($s = 0.3464$), "pop-up ads" ($s = 0.3440$), "ads pop up" ($s = 0.3388$), "commercials advertise" ($s = 0.3272$), "advertising" ($s = 0.3241$)
Payment	"payment getting" ($s = 0.2618$), "payment get" ($s = 0.2547$), "getting payment" ($s = 0.2530$), "take payment"($s = 0.2504$), "payment taking" ($s = 0.2471$), "payment" ($s = 0.2401$)

the changepoints. In the second scenario, we specify the potential changepoints, providing domain knowledge for software requirements rather than automatic changepoint detection. Therefore, the changepoint parameters are used when we provide the dates of the changepoints instead of the Prophet determining them. In this case, we use the most recent observations that have a value greater than the average of observations, i.e., critical periods with high frequencies of negative reviews in the past.

We used the MAPE (Mean Absolute Percentage Error) metric to evaluate the forecasting performance (Makridakis, 1993), as defined in Equation 5.4,

$$MAPE = \frac{1}{h}\sum_{t=1}^{h}\frac{|real_t - pred_t|}{real_t} \tag{3.8}$$

where $real_t$ is the real value and $pred_t$ is the predicted value by the method, and h is the number of forecast observations in the estimation period (prediction horizon). In practical terms, MAPE is a measure of the percentage error that, in a simulation, indicates how close the prediction was made to the known values of the time series. We consider a prediction horizon (h) ranging from 1 to 4, with weekly seasonality.

Table 3.5 summarizes the main experimental results. The first scenario (1) with the default parameters obtains superior results compared to the second scenario (2) for all forecast horizons. In general, automatic changepoints obtains 9.33% of model improvement, considering the average of MAPE values from all horizons ($h = 1$ to $h = 4$).

In particular, we are interested in the peaks of the series since our hypothesis is that the peaks represent potential problems in a given software requirement. Thus, Table 3.6 shows MAPE calculated only for time series peaks

Table 3.5: Comparison of MAPE in General.

* h	MAPE (Mean ± SD) (1) Automatic changepoint	MAPE (Mean ± SD) (2) Specifying the changepoints
1	13.82 ± 16.42	15.47 ± 14.42
2	15.58 ± 19.09	16.94 ± 17.20
3	16.26 ± 20.18	17.60 ± 18.71
4	16.09 ± 19.24	17.47 ± 18.37

during forecasting. In this case, predictions with the custom changepotins locations (scenario 2) obtained better results than the automatic detection for all prediction horizons ($h = 1$ to $h = 4$), obtaining 3.82% of forecasting improvement. These results provide evidence that domain knowledge can improve the detection of potential software requirements to be analyzed for preventive maintenance.

Table 3.6: MAPE analysis (at the peaks of the time series) of each scenario considering the software requirements.

* h	MAPE (Mean ± SD) (1) Automatic changepoint	MAPE (Mean ± SD) (2) Specifying the changepoints
1	10.65 ± 8.41	10.30 ± 8.06
2	11.61 ± 8.80	11.00 ± 8.71
3	11.81 ± 8.86	11.42 ± 8.52
4	11.49 ± 8.71	11.19 ± 8.34

In particular, analyzing the prediction horizon, the results show that the best predictions were obtained with $h = 1$ (1 week). In practical terms, this means the initial trend of a defective requirement can be identified one week in advance. We can note that a prediction error rate (MAPE) of up to 20% is acceptable. For example, consider that the prediction at a given point is 1000 negative reviews for a specific requirement, but the model predicts 800 negative reviews. Even with 20% of MAPE, we can identify a significant increase in negative reviews for a requirement and trigger alerts for preventive software maintenance.

Finally, to exemplify MAPP-Reviews forecasting, Figure 3.7 shows the training data (Arriving time software requirement) represented as black dots and the forecast as a blue line, with upper and lower bounds in a blue shaded area. At the end of the time series, the darkest line is the real values plotted over the predicted values in blue. The lines plotted vertically represent the changepoints.

For reproducibility purposes, we provide a GitHub repository at https://github.com/vitormesaque/mapp-reviews containing the source code and

details of each stage of the method, as well as the raw data and all the results obtained.

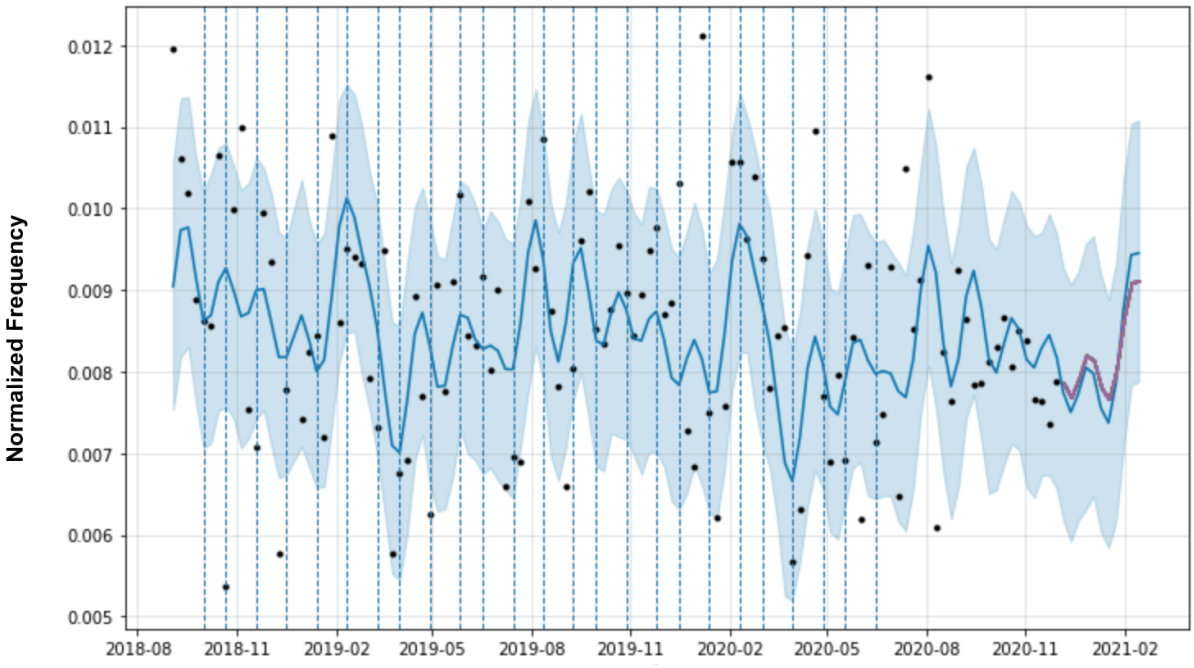

Figure 3.7: Forecasting for software requirement cluster (Arriving time) from Uber Eats App reviews

3.6.5 Discussion

An issue related to a software requirement reported in user reviews is defined as an emerging issue when there is an upward trend for that requirement in negative reviews. Our method trains predictive models to identify requirements with higher negative evaluation trends, but a negative review will inevitably impact the rating. However, our objective is to mitigate this negative impact.

The prediction horizon (h) is crucial in detecting emerging issues and mitigating their negative impacts. Software engineers and the development team must be aware of software problems as early as possible to proactively address them. However, predicting issues several months in advance proves challenging, as finding a correlation between current occurrences and future bug reports becomes difficult. As a result, our approach, MAPP-Reviews, focuses on weekly predictions. By doing so, we can identify emerging issues and forecast whether they will escalate in the upcoming weeks.

Even at the weekly level, having the shortest forecast horizon possible is most effective, such as one week ($h = 1$). Using longer horizons, like three weeks ($h = 3$) or four weeks ($h = 4$), may be too late to prevent an issue from worsening and causing a significant impact on the overall app rating. Experimental

evaluations demonstrate that our method achieves the most accurate predictions with the shortest horizon ($h = 1$). Practically speaking, MAPP-Reviews can identify the initial trend of a defective requirement one week in advance.

Furthermore, it is worth noting that a prediction error rate (MAPE) of up to 20% is deemed acceptable. For instance, suppose the model predicts 800 negative reviews for a specific requirement at a given point, while the actual number is 1000 negative reviews. Even with a 20% MAPE, we can still identify a significant increase in negative reviews for that requirement and trigger alerts for preventive software maintenance. In other words, when MAPP-Reviews predict an upward trend, the software development team should receive an alert.

In Figure 3.7, the time series forecast illustrates how the model successfully predicted the peak of negative reviews for the "Arriving time" requirement one week in advance.

An emerging issue detection system based only on the frequency of a topic could trigger many false detections, i.e., it would not detect defective functionality but issues related to the quality of services offered. Analyzing user reviews, we found that some complaints are about service issues rather than defective requirements. For example, the user may complain about the delay in the delivery service and negatively rate the app. However, they are complaining about the restaurant, i.e., a problem with the establishment service. We have seen that this pattern of user complaints is repeated across other app domains, not just the food delivery service. In delivery food apps, these complaints about service are constant, uniform, and distributed among all restaurants available in the app. In Table 3.3, it is clear that the emerging issue refers to the deficient implementation of the estimated delivery time prediction functionality. Our results show that when there is a problem in the app related to a defective software requirement, there are increasing complaints associated with negative reviews regarding that requirement.

An essential feature in MAPP-Reviews is changepoints. Assume that a time series represents the evolution of a software requirement over time, observing negative reviews for this requirement. Also, consider that time series frequently have abrupt changes in their trajectories. Given this, the changepoints describe abrupt changes in the time series trend, i.e., a specific date indicating a trend change. Therefore, specifying custom changepoints becomes significantly important for the predictive model because the uptrend in time series can also be associated with domain knowledge factors. By default, our model will automatically detect these changepoints. However, we have found that specifying custom changepoints improves prediction significantly in critical situations for the emerging issue detection problem. The

automatic changepoint detection generally had better MAPE results in most evaluations. However, the custom changepoints obtained the best predictions at the time series peaks for all horizons ($h = 1$ to $h = 4$) of experiment simulations. Our experiment suggests a greater interest in identifying potential defective requirements trends in the time series peaks. As a result, we conclude that specifying custom changepoints in the predictive model is the best strategy to identify potential emerging issues.

Furthermore, the results indicate the potential impact of incorporating changepoints into the predictive model using the information of app developers, i.e., defining specific points over time with a meaningful influence on app evaluation. In addition, software engineers can provide sensitive company data and domain knowledge to potentially explore and improve the predictive model. For this purpose, we depend on sensitive company data related to the software development and management process, e.g., release planning, server failures, and marketing campaigns. In particular, we can investigate the relationship between the release dates of app updates and the textual content of the update publication with the upward trend in negative evaluations of a software requirement. In a real-world scenario in the industry, software engineers using MAPP-Reviews will provide domain-specific information.

3.6.6 Limitations

Despite the significant results obtained, we can still improve the predictive model. In the scope of our experimental evaluation, we only investigate the incorporation of software domain-specific information through trend changepoints. The predictive model did not consider company-sensitive information and the development team's domain knowledge because we don't have access to this information. Therefore, we intend to evaluate our proposed method in the industry and explore more specifics of the domain knowledge to improve the predictive model.

Another issue that is important to highlight is sentiment analysis in app reviews. We assume that it is possible to improve the classification of negative reviews by incorporating sentiment analysis techniques. We can incorporate a polarity classification stage (positive, negative, and neutral) of the extracted requirement, allowing a software requirements-based sentiment analysis. The MApp-Reviews only consider negative reviews with low ratings and associate them with the software requirements mentioned in the review.

Finally, to use MAPP-Reviews in a real scenario, there must be already a sufficient amount of reviews distributed over time, i.e., a minimum number of time-series observations available for the predictive model to work properly. Therefore, in practical terms, our method is more suitable when large volumes

of app reviews can be analyzed.

3.7 Final Remarks

In this chapter, we introduce the MAPP-Reviews method. It provides software engineers with tools to perform software maintenance activities, mainly preventive maintenance, by automatically monitoring the temporal dynamics of software requirements.

We 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.
- The results indicate the potential impact of incorporating changepoints into the predictive model using the information of app developers.
- Our method can provide important information to software engineers regarding software development and maintenance.
- The model can predict the peaks of negative reviews for the software requirement one week in advance.
- Software engineers can act proactively through the proposed MAPP-Reviews approach and reduce the impacts of a defective requirement.

CHAPTER

4

Issue Detection and Prioritization based on App Reviews

4.1 Introduction

The MApp-Reviews method presented in the previous Chapter has certain limitations that need to be addressed. Firstly, the analysis primarily focuses on negative reviews with low ratings and their association with software requirements, disregarding sentiment analysis that can provide a more nuanced understanding of user sentiment. By incorporating sentiment analysis techniques to classify reviews into positive, negative, and neutral categories, the accuracy of the classification can be significantly improved. Another limitation of MApp-Reviews is its dependence on a substantial volume of app reviews distributed over time. This requirement makes the method less suitable for scenarios with limited review data, as a minimum number of observations is necessary to generate reliable predictions.

To overcome these limitations and offer alternative approaches to enhance issue analysis, we introduce the MApp-IDEA method. MApp-IDEA incorporates sentiment analysis and is designed to be effective even with smaller datasets. Additionally, we expand the scope of issue detection in MApp-IDEA to capture general issues that may not be directly associated with specific software requirements.

This chapter introduces the 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 is-

sues. 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.

The rest of this chapter is structured as follows. In Section 4.2, we highlight the key contributions of this chapter. Our approach, MApp-IDEA (Monitoring App for Issue Detection and Prioritization), is presented in Section 4.3, followed by a demonstration of MApp-IDEA in action in Section 4.4. The experiment design used to validate our approach is described in Section 4.5, and the validation results and discussion are presented in Section 4.6. We address the threats to the validity of our investigation in Section 4.7. Lastly, Section 5.6 provides our conclusions and final remarks.

4.2 Main Contributions

Our main contributions presented in this chapter are briefly summarized below:

1. We introduce the prioritizing issues approach that automatically generates a risk matrix, combining sentiment analysis, clustering, and graph theory.

2. 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. 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.

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.

4.3 MApp-IDEA Method Architecture

To detect and prioritize EAI, we introduce MApp-IDEA. Our method is divided into five stages, as shown in Figure 4.1. 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 4.1.

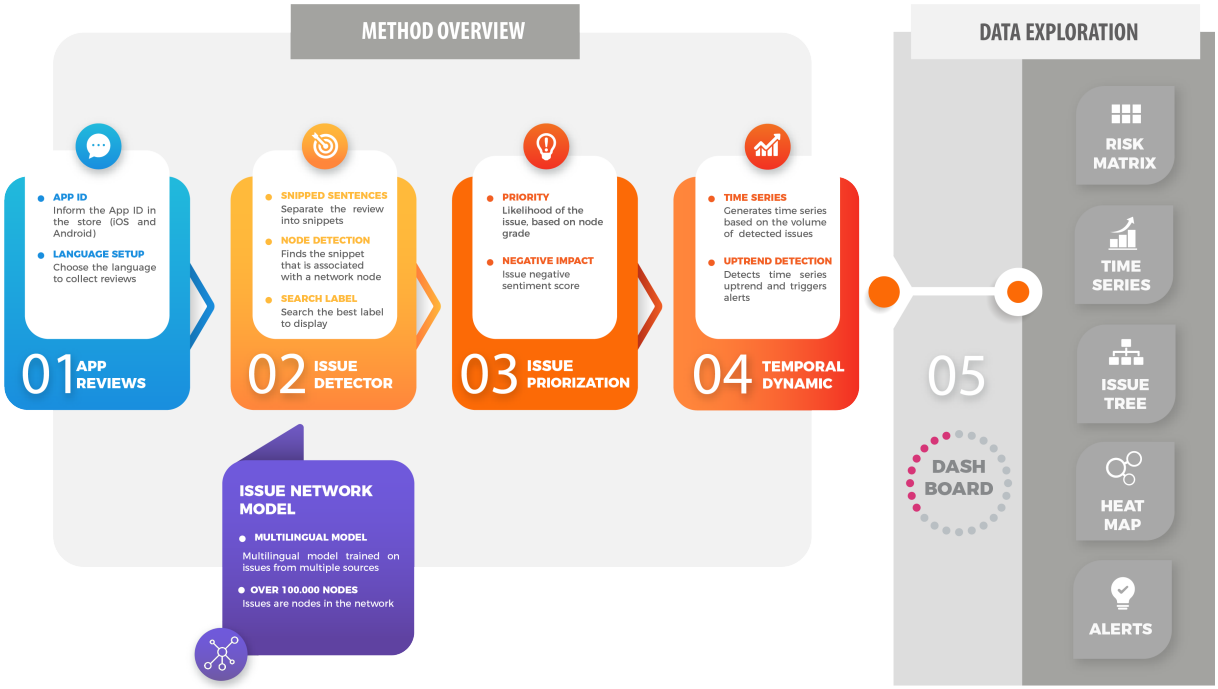

Figure 4.1: The architecture of the MApp-IDEA framework for detecting and prioritizing emerging issues

4.3.1 App Reviews

In the first stage of MApp-IDEA, raw reviews are collected from app stores using a web crawler tool via RESTful API. The app stores provide the textual content of the reviews, the publication date, and the rating stars of the reviews reported by users. In addition to reviews, we collect app metadata and metrics such as release dates, versions, number of installs, overall rating, and the total number of reviews. We use app metadata to drive overall performance and provide additional analysis of the temporal dynamics of metrics.

In this step, all configuration is done through a user-friendly interface, as

shown in Figure 4.3. The developer enters the app ID of the app store, the language, and the number of reviews to be processed. The crawler process runs in the background, and upon completion, the data is organized into the appropriate data structure for processing by the MApp-IDEA issue detection stage, as shown in Figure 4.2.

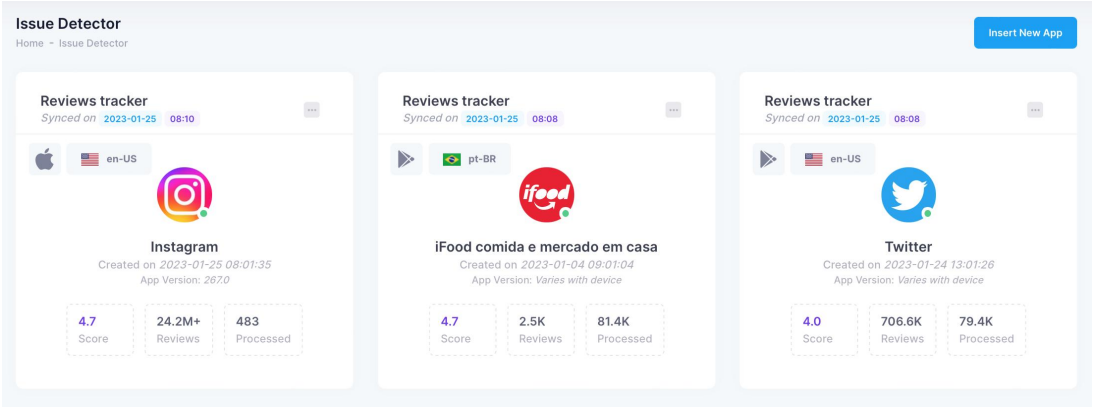

Figure 4.2: Control panel to manage all apps

After app registration and initial processing, new app reviews are automatically processed and synchronized in the background over time, as shown in Figure 4.4. From this moment on, all app information is automatically tracked.

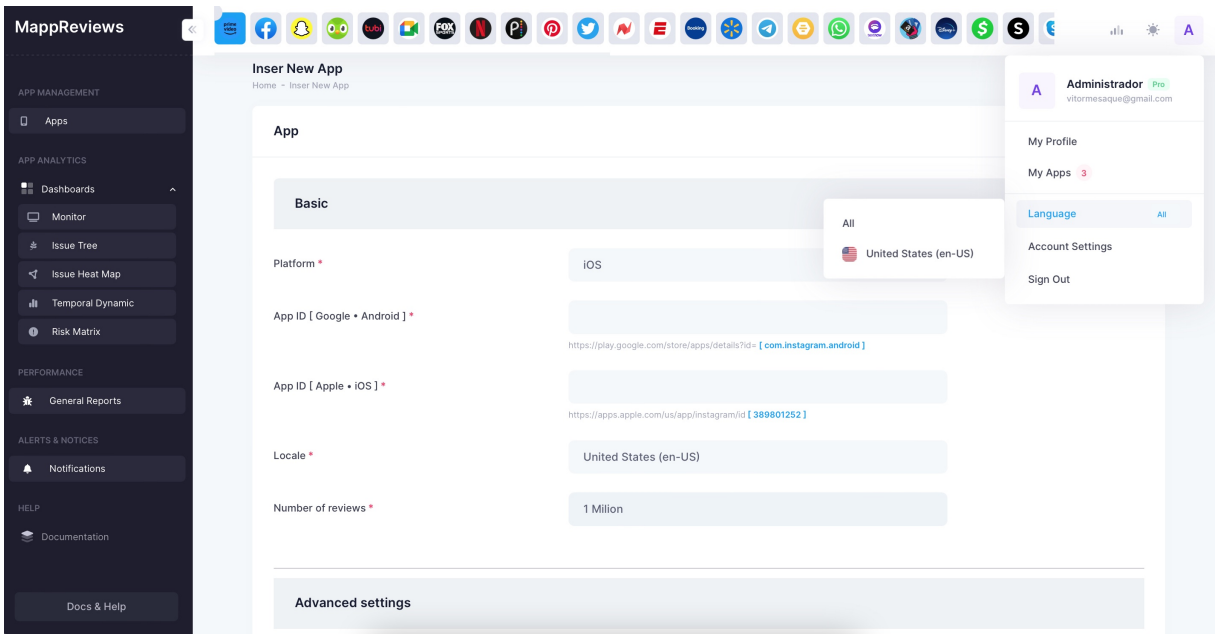

Figure 4.3: Initial app settings

4.3.2 Issue Detector

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

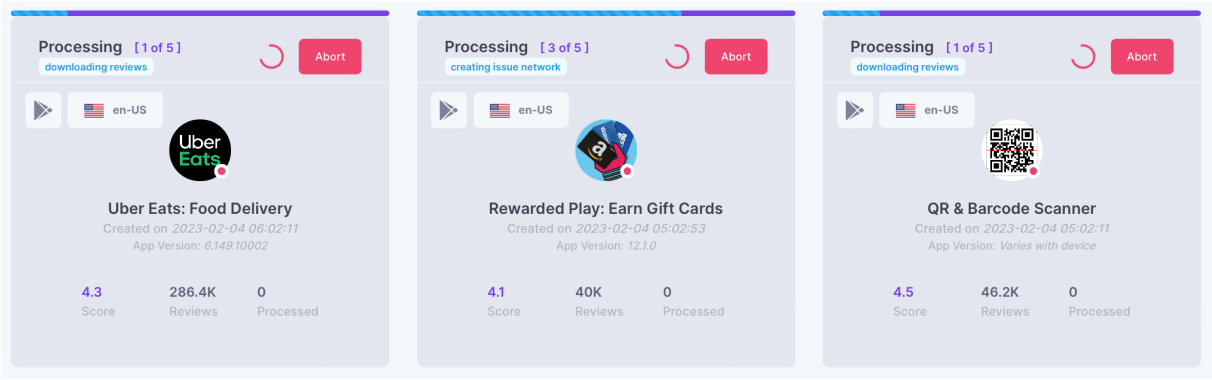

Figure 4.4: Synchronization process and automatic update of new reviews and issues

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 (Mihalcea and Radev, 2011).

In our approach, we have an acyclic graph, where each issue is a vertex, and edges are defined by the similarity between issues based on the proximity of the issues vectors. In practice, we have a tree where each issue is a node, and similar issues with spelling variations or related issues are adjacent and connected by an edge. Therefore, we represent each issue through word embedding.

In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path or, equivalently, a connected acyclic undirected graph. A forest is an undirected graph in which any two vertices are connected by at most one path, an acyclic undirected graph, or a disjoint union of trees (Williamson, 2010).

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. 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, as shown in Figure 4.5. However, a forest can have singleton graphs consisting of a single isolated node with no edges. In our approach, we can have issues that are not connected to any other.

To generate the issues graph, we used a multilingual model based on the BERT (Devlin et al., 2018), 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

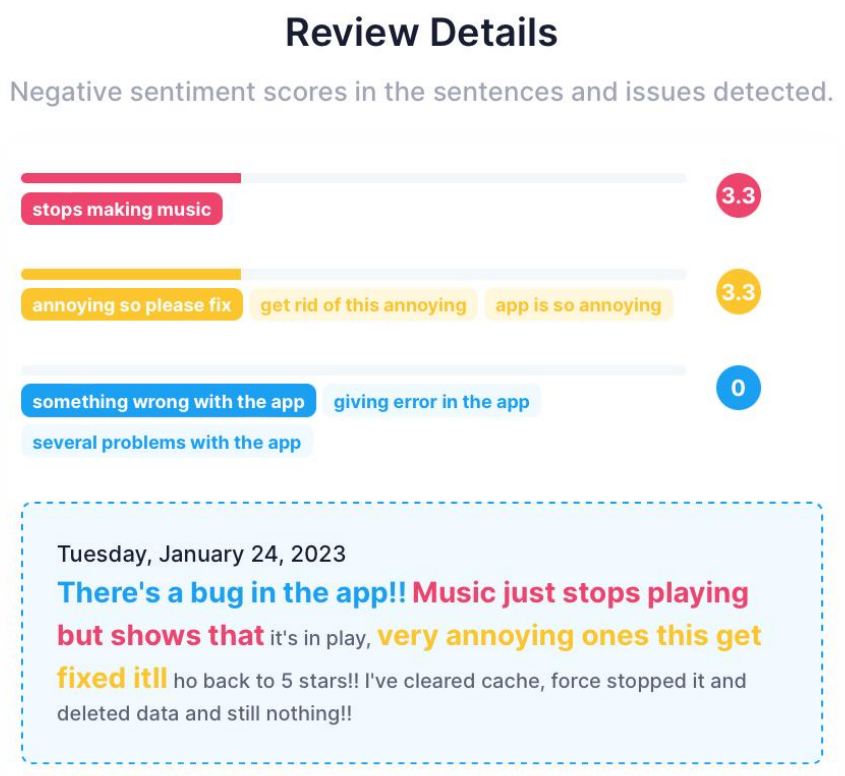

Figure 4.5: Details of issues detected in a review. For each review, MApp-IDEA extracts the review that is associated with the issue

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] (Araujo and Marcacini, 2021). One of the training objectives is the noisy reconstruction defined in Equation 4.1,

$$p(\bar{i} \| \hat{i}) = \sum_{j=1}^{k} m_j \frac{exp(\mathbf{h}_{c_j}^{\top} \mathbf{w}_{t_j})}{\sum_{t'} exp(\mathbf{h}_{c_j}^{\top} \mathbf{w}_{t'})} \tag{4.1}$$

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 4.1. 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*".

In our architecture, we use a version of the BERT model called DistilBERT (a distilled version of BERT) (Sanh et al., 2019), which has the same architecture as the BERT model. In summary, the DistilBERT is a general-purpose pre-trained version of BERT, 40% smaller and 60% faster, that retains 97% of the language understanding capabilities (Sanh et al., 2019).

We use the Facebook AI Similarity Search (Faiss) (Johnson et al., 2017) to detect similarity between vectors. This library allows us to search high-dimensional vectors similar to each other using nearest-neighbor search implementations. Given a query vector, the similarity search returns a list of vectors closest to that vector in terms of Euclidean distance. Our issue detector classifier receives the correlation threshold and n-gram size parameters set to 0.8 and 7, respectively. The correlation threshold parameter defines the minimum similarity correlation between the vectors. A snippet is an issue if the distance between the most similar vectors exceeds the minimum correlation threshold. The n-gram size parameter is the maximum size of words that an issue can assume.

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.

4.3.3 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 4.6.

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.

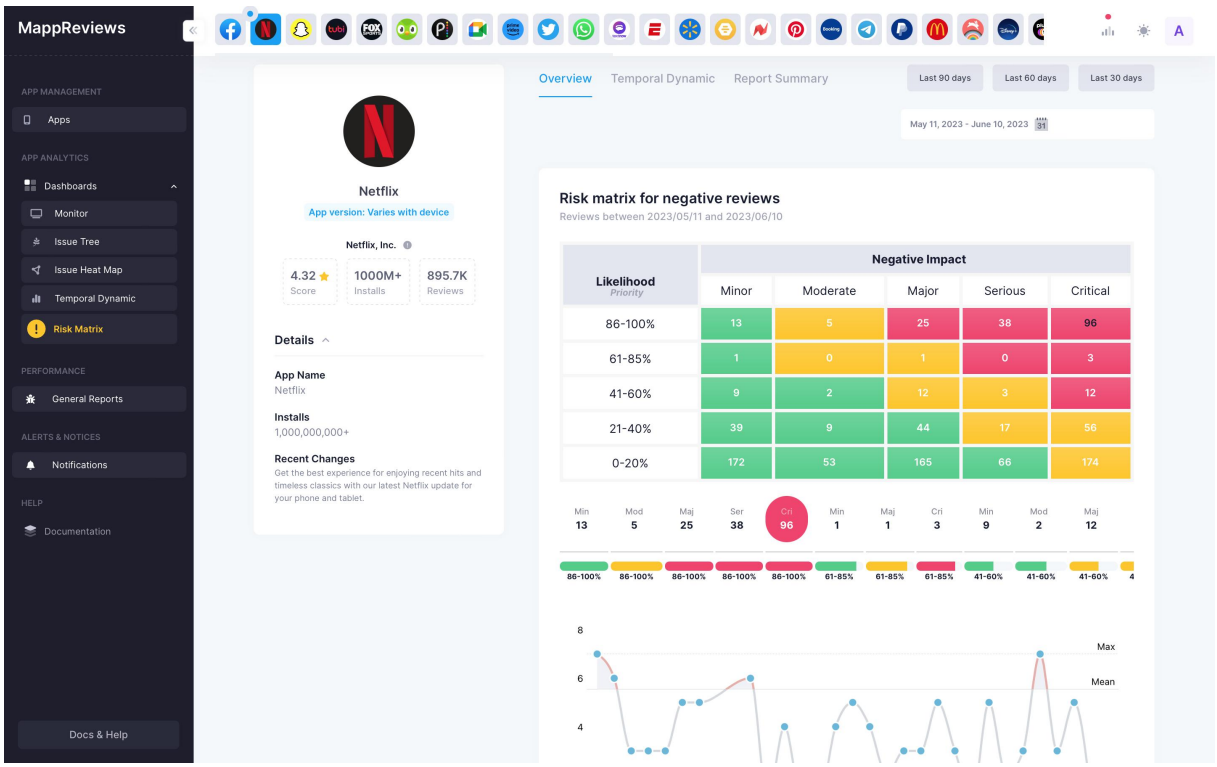

Figure 4.6: Navigable risk matrix. By clicking on the cell, it is possible to browse the time series of each cell

4.3.3.1 Negative Impact

To calculate the issue sentiment score, we use VADER (Valence Aware Dictionary for Sentiment Reasoning), a model used for text sentiment analysis sensitive to both polarities (positive/negative) (Hutto and Gilbert, 2014). VADER sentiment analysis relies on a dictionary that maps lexical features to emotion intensities known as sentiment scores. The sentiment score of a text can be obtained by summing the intensity of each word in the text.

The compound score is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1 (most extreme negative) and $+1$ (most extreme positive): (i) positive sentiment $(compoundScore \geq 0.05)$, (ii) neutral sentiment $(compoundScore > -0.05)$, and (iii) $(compoundScore < 0.05)$ negative sentiment $(compound_{score} \leq -0.05)$. For our risk matrix, we consider negative sentiment, i.e., the compound score less than or equal to zero $(C_s \leq 0)$. On the x-axis of the matrix is the discretization of the negative sentiment score into four levels.

To show the result more understandably, we present the negative impact in absolute (positive) values, ranging from 0 to 10, as shown in Figure 4.7.

4.3.3.2 Likelihood

On the y-axis (likelihood), we have the discretization of the degree of similarity of the issue with other issues in the graph in four levels. The higher the

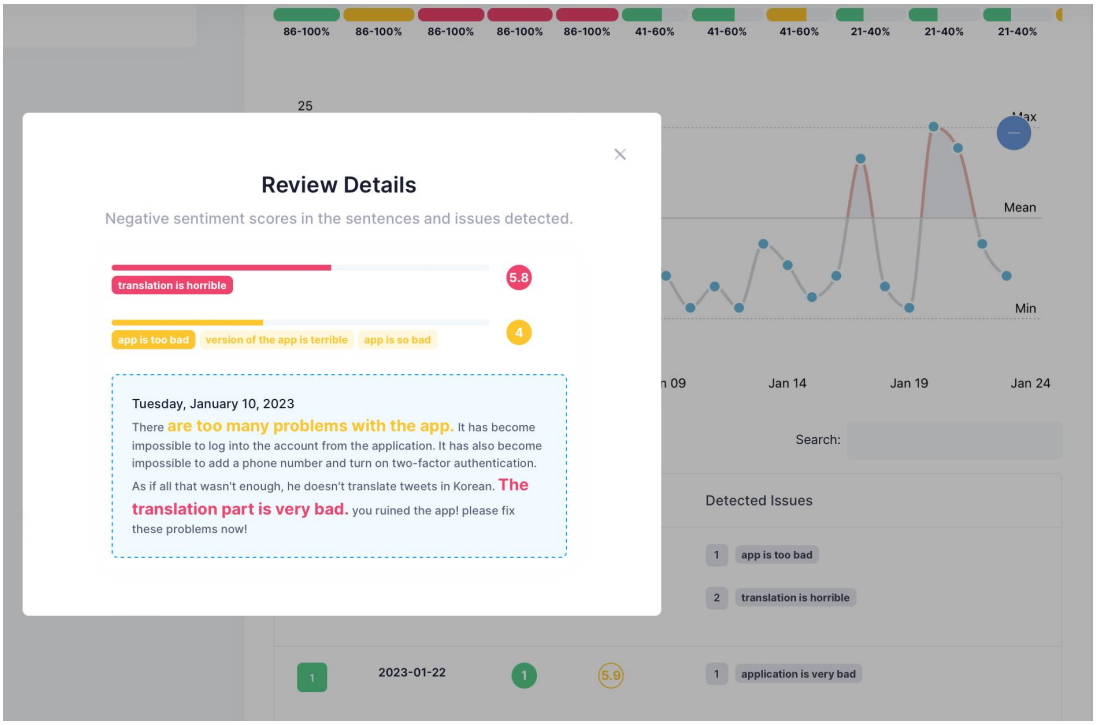

Figure 4.7: Navigable risk matrix. By clicking on the cell, it is possible to browse the time series of each cell

level, we have more related issues. The lower the level, the less similarity the issue has to other issues.

We use a partitional clustering strategy to calculate the issue degree measure in the graph. We cannot simply compute the degree of the node in the graph to generate the x-axis value because, in our model, we may have too many singleton graphs, which would unbalance the matrix. Therefore, we need to consider this to calculate the likelihood. Let V be the number of vertices in the graph. We group all vertices of the graph into $\sqrt{|V|}$ clusters and define the x-axis value of each node as the size of its cluster, i.e., the number total number of issues that are in the cluster.

In partitional clustering, the main goal is to partition a set of examples into k groups, where the user typically determines the value of k. MApp-Reviews utilizes the K-means algorithm (MacQueen et al., 1967), which is a widely recognized and extensively employed method for partitional clustering, particularly in the analysis of textual data.

Formally, let $R = \{i_1, i_2, ..., i_n\}$ a set of issues, where each issue i is a m-dimensional real vector from an word embedding space. The K-means clustering aims to partition the n issues into k $(2 \leq k \leq n)$ clusters $C = \{C_1, C_2, ..., C_k\}$, thereby minimizing the within-cluster sum of squares as defined in Equation 4.2, where μ_i is the mean vector of all issues in C_i.

$$\sum_{C_i \in C} \sum_{r \in C_i} \|r - \mu_i\|^2 \tag{4.2}$$

We defined the cluster number k as $\sqrt{n}$, where n is the total issues (nodes). Then, for each issue, we define its likelihood as the size of the cluster in which it is inserted. We then normalize this value between 0 and 1 using Min-Max (4.3). Therefore, the likelihood of the issue occurring is associated with the similarity between the nodes in the network, as a very frequent issue will have many occurrences, variations of writing, and different ways of describing the same problem by users.

4.3.3.3 Risk Matrix Construction

Our algorithm for calculating the risk matrix places the reviews with one or more issues in their respective cell [i,j] of the matrix according to the thresholds of negative impact and likelihood.

With the algorithm's output, we prioritized the analysis, observing the most likely and critical problems, as shown in Figure 4.8. The algorithm subdivides the matrix into three priority levels: level 1 (low), level 2 (medium), and level 3 (high). Implementation details follow in the algorithm's pseudocode 1.

Algorithm 1: Algorithm that calculates the i,j position of each issue detected in the Risk Matrix

```
1:  function RISKMATRIXGENERATOR(reviews)
2:      M ← [5][5]
3:      levels ← [L, M, H]
4:      I ← [0.15, 0.3, 0.6, 0.7, 1]
5:      P ← [1, 0.7, 0.6, 0.3, 0.15]
6:      issueCollection ← issueDetector(reviews)
         for issue ← 1 to issueCollection.length do
7:      priority ← issueCollection[issue].degree
8:      impact ← issueCollection[issue].negativeScore
         for i ← 1 to 5 do
             for j ← 1 to 5 do
                 if (level ← checkInterval(priority, impact, P, I, i, j)) then
9:                   M[i][j] ← M[i][j] + 1
10:                  levels[level] ← levels[level] + 1
                 end
             end
         end
         end
11:     return M
12: end function
13: function CHECKINTERVAL(priority, impact, P, I, i, j)
14:     level ← checks the range of (priority, impact) matches the threshold range
    (P, I) in (i, j) of the matrix
15:     return level
16: end function
```

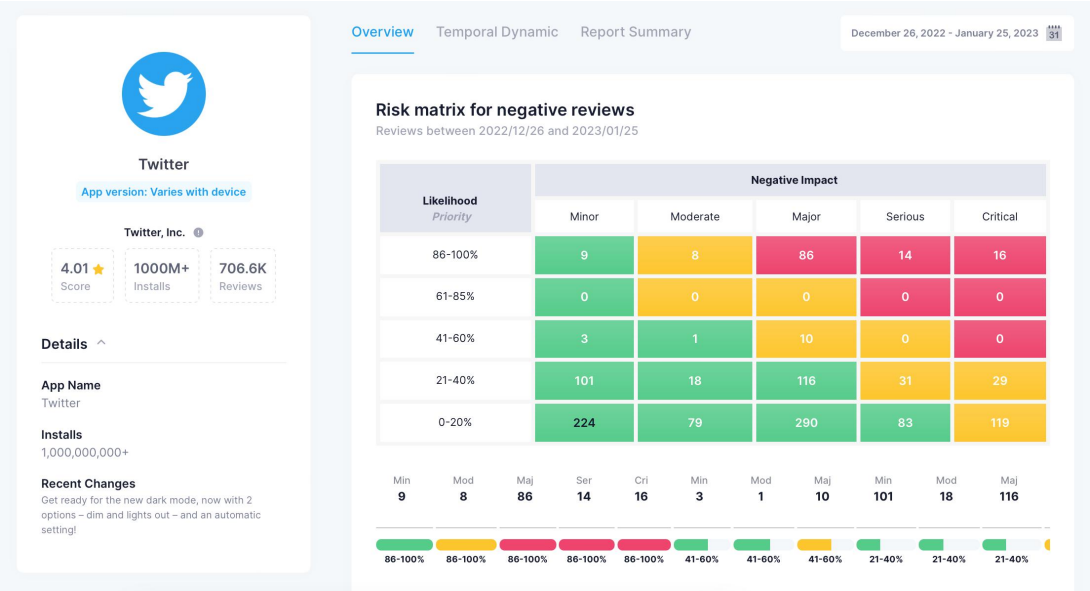

Figure 4.8: Risk matrix where each cell contains the number of reviews with issues. Colors vary from green to red, indicating three critical and priority levels

4.3.4 Temporal Dynamic

Considering that the issue time series is a set of observations obtained sequentially over time, as illustrated in Figure 4.9, we can model the occurrence of issues over time in a time series to detect upward trends in their frequency, as shown in Figure 4.10. 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 (Chatfield, 2003).

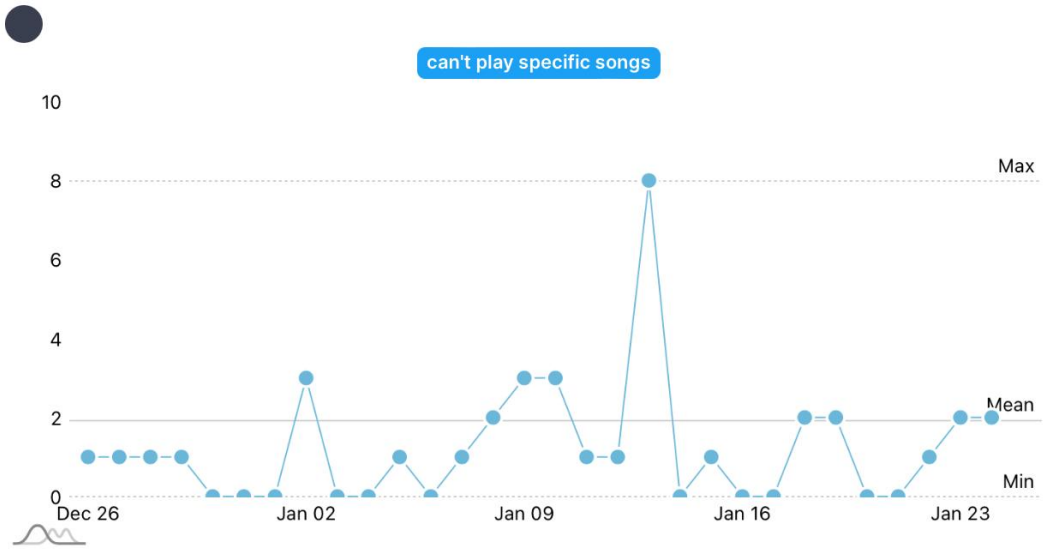

Figure 4.9: Issue time series “can’t play specific songs”

Figure 4.10: Time series of Spotify app issues with peak issues on January 10th and 23rd. Red lines indicate an uptrend in the time series.

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. Figure 4.11 visually depicts the time series of priority levels associated with the identified issues. The green, yellow, and red lines in the figure correspond to low, medium, and high priority levels, respectively. This analysis provides insights into how the risk levels evolve over time and allows us to monitor the changing prioritization of issues.

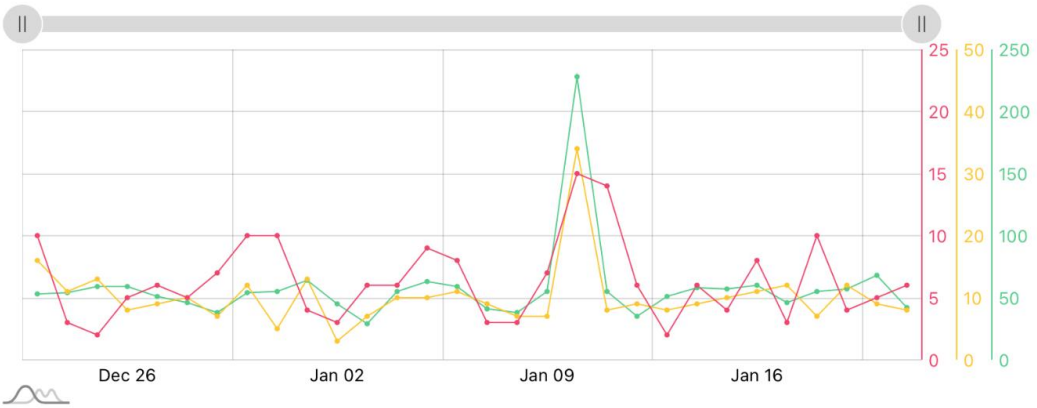

Figure 4.11: Time series of issues with three lines representing issues in priority levels. The colors green, yellow, and red represent levels 1, 2, and 3

4.4 MApp-IDEA in Action

We present an example of MApp-IDEA's usage in this section.

4.4.1 Monitor

On the home screen, MApp-IDEA presents all monitored applications, with daily performance statistics of apps concerning comment volume and star rating, as shown in Figure 4.13. We also provide a summary of notifications and flips, the daily trend of top issues, the volume of issues detected for all apps, and changelog statistics, as shown in Figure 4.12 and Figure 4.14.

Figure 4.12: Dashboard with daily performance information for each app on the star rating and number of reviews

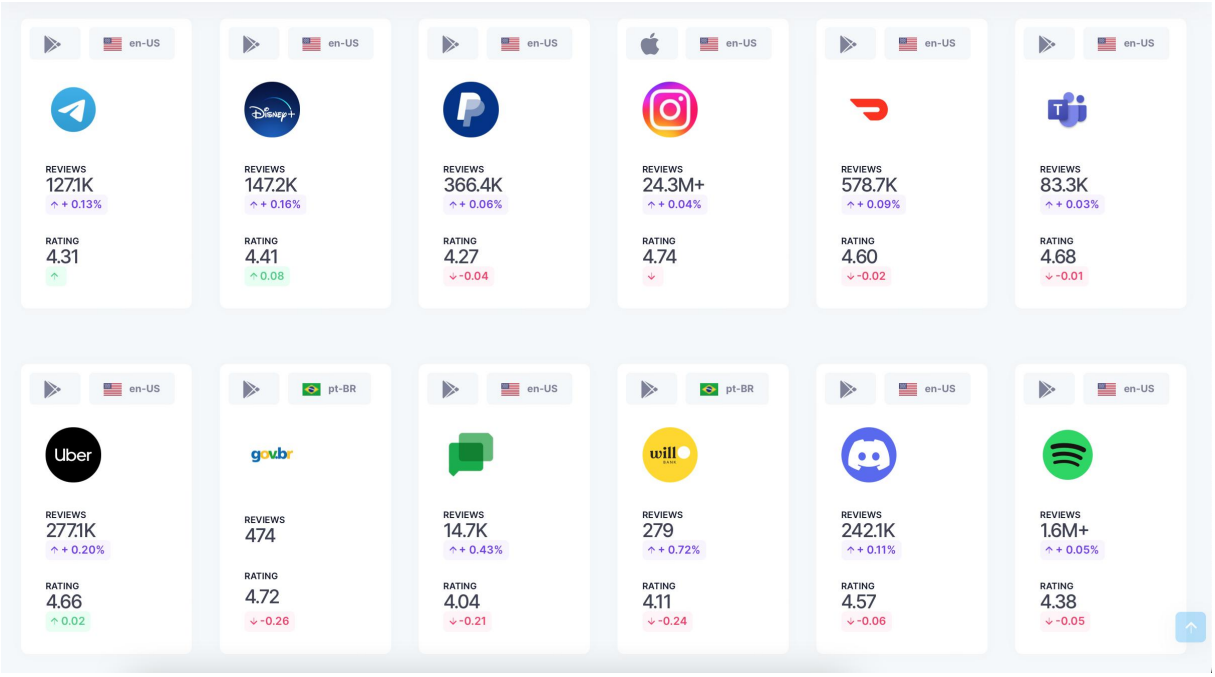

Figure 4.13: Dashboard with daily performance information for each app on the star rating and number of reviews

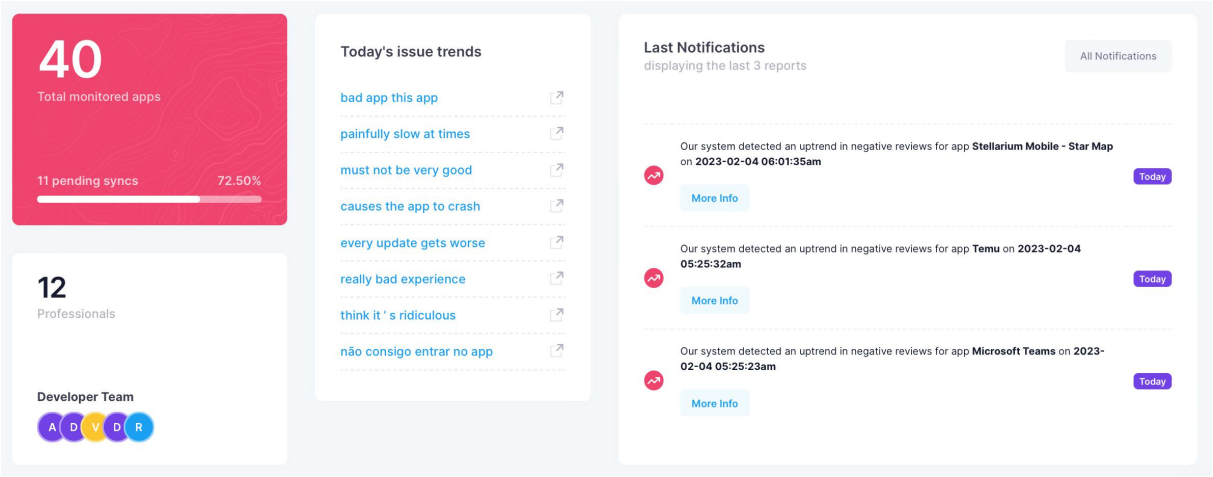

Figure 4.14: Summary dashboard of top-n issues of the day for monitored apps

4.4.2 3-tier Navigable Tree

In this visualization approach, interactive navigation across the three layers of the tree is facilitated, enhancing the understanding of the underlying data structure. The size (diameter) of each node in the visualization corresponds to the volume of issues associated with the subtree rooted at that node, as illustrated in Figure 4.16. Larger nodes indicate a higher frequency of issues. Furthermore, each distinct tree in the visualization is assigned a color, representing a group of related issues categorized within up to three levels, as illustrated in Figure 4.15. This visualization technique enables users to explore and analyze similar issue groups effectively.

4.4.3 Network Exploratory Heatmap

We developed an exploratory issue network called a heat map, which emphasizes issues with higher priority according to the volume of occurrences and degree of the node in the graph. In this form of visualization, we use the most representative nodes of the tree, from layers 1 and 2, to generate a new graph of issues with approximately 600 nodes.

Mathematically, a network is defined as an undirected graph $G = (V, E)$, formed by a set $V = (v_1, v_2, ..., v_n)$ of nodes (words). A set $E = (e_1, e_2, ..., e_n)$ of edges that are represented by an adjacency matrix A, whose elements A_{ij} are equal to 1 whenever there are an edge connecting nodes (words) i and j, and equal to 0 otherwise.

This new graph allows us to identify critical regions of the graph through the volume of issues related to each node and categorize the visualization into different critical levels. Each node of this graph is connected to similar nodes, calculated by the distance of similarity between the vectors. The “payment

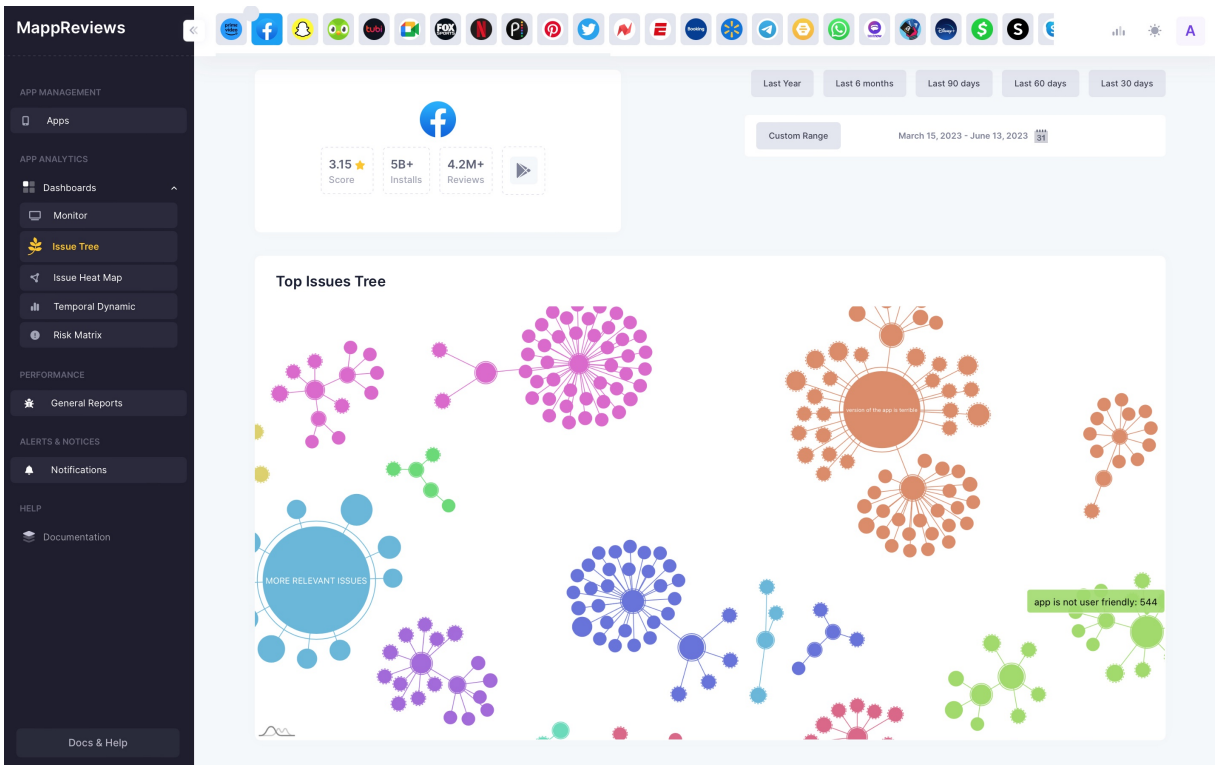

Figure 4.15: Facebook app's issue tree into a dynamic and interactive 3-tier structure between March 15 and June 13, 2023

problem" node will be close to the "credit card problem" node, with an edge connecting these two nodes. We call this new graph a Heat Map, representing the critical nodes in 4 levels, from the least critical to the most critical, as shown in Figure 4.17 and Figure 4.18.

For example, consider app A, with 10,000 reviews in the last 30 days. After detecting the issues, we already know which tree nodes are related to the issues. The identified application A issues can belong to any of the three layers of the original tree, with layers 1 and 2 being the most representative. Suppose 200 issues of application A were detected. These detected issues are distributed in the graph according to their representation. Consider that 50% of these issues are related to the "app stopped working" node, 25% are related to the "payment problem" node, 15% are related to the "I can't access my account" node, and 10% are related to other nodes. The Heat Map Graph has 600 nodes in all. However, these three detected nodes will be highlighted, as exemplified in Figure 4.19. In this example, the "app stopped working" will be marked with red color, flashing on the screen, and with a larger diameter than all the others.

The "payment problem" node will be marked red, but without flashing and with a smaller diameter. The "I can't access my account" node will be marked in yellow. The other nodes in the network will be marked in blue or gray. Nodes marked in blue are nodes not detected by the issue detector for application

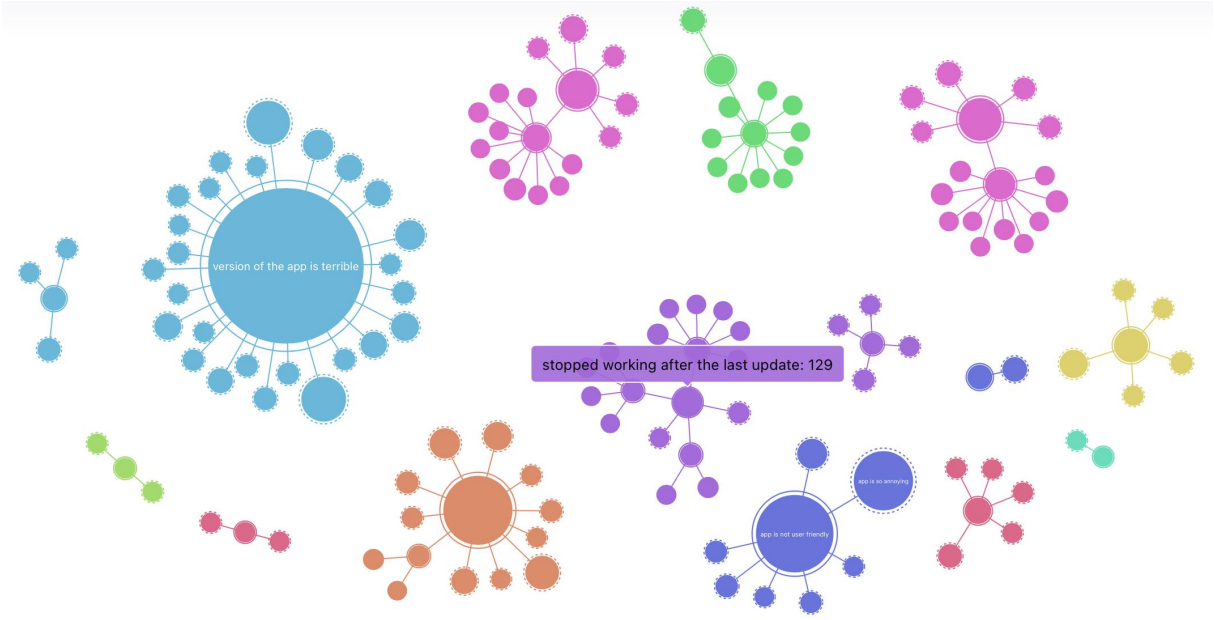

Figure 4.16: 3-tier tree dashboard

A, with a high vertex degree, i.e., they are more strongly connected to other nodes. If these blue nodes turn yellow or red, it could be a severe problem. The nodes marked in gray were not detected by the issue detector and had a low vertex degree value, i.e., they are problems of lesser impact.

To calculate the threshold between the critical levels of the graph, we calculate the frequency of occurrence of each node. Then we normalize the frequency to obtain values between 0 and 1, using the Min-max scale, according to equation 4.3.

$$m = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{4.3}$$

where m is the new value, x is the original value of the cell, x_{min} is the minimum frequency value, and x_{max} is the maximum value of the frequency.

Subsequently, the computed value of m is compared to a threshold value determined by the software engineer. The choice of the threshold influences the sensitivity of the heat map graph coloring. A lower threshold increases the level of sensitivity, resulting in more evident variations in the color scheme of the heat map graph, as shown in Figure 4.20.

The expectation is that denser regions at higher critical levels will receive prioritization. If there are numerous red dots clustered in a specific region, it indicates that app users are reporting a common and frequent problem. Such frequent issues tend to occur repeatedly, and spelling variations will be a “hub” in the network.

Our platform offers interactive navigation within the graph, allowing visualization of reviews associated with each node and access the top-n issues

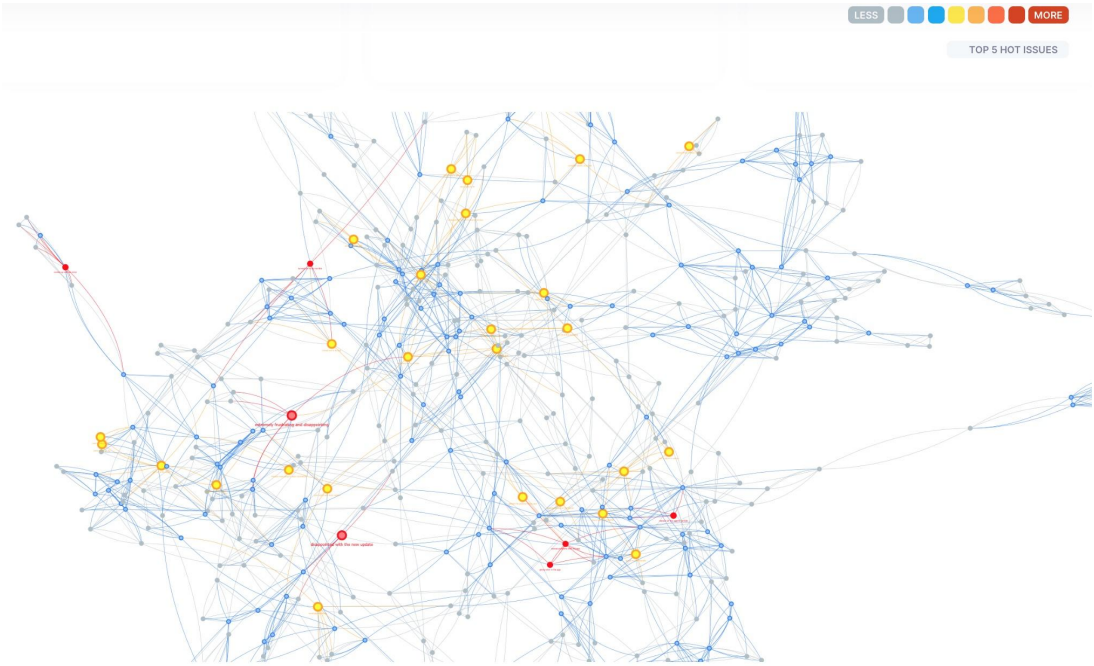

Figure 4.17: Heatmap network where each node represents an issue and is connected by similarity

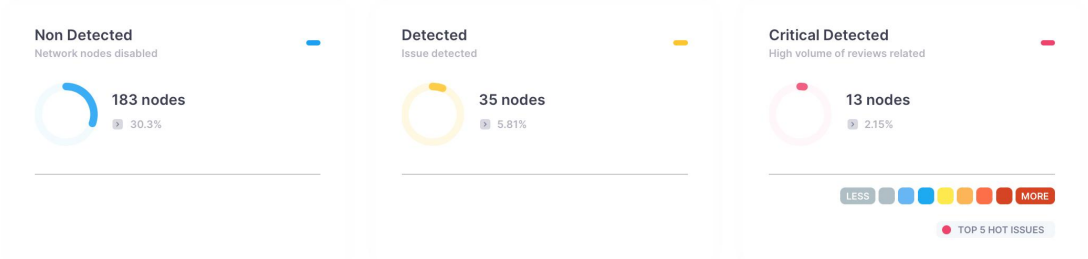

Figure 4.18: Description of color variation in heatmap

linked to those reviews.

4.4.4 Interactive Risk Matrix

The risk matrix generated by our algorithm is presented in a dynamic and navigable way. Each cell of the matrix is clickable, where we can view the issues related to each cell and the reviews and sentences linked to the issues. We can also sort issues by priority, likelihood, and negative impact levels. For each cell in the matrix, we also display the temporal modeling, i.e., the evolution of that cell over time, as shown in Figure 4.21.

Additionally, we provide summary reports of issue priority levels, as shown in the screen example (Figure 4.22).

4.4.5 Time series

The MApp-IDEA presents different types of time series to observe the time evolution of issues and identify trends.

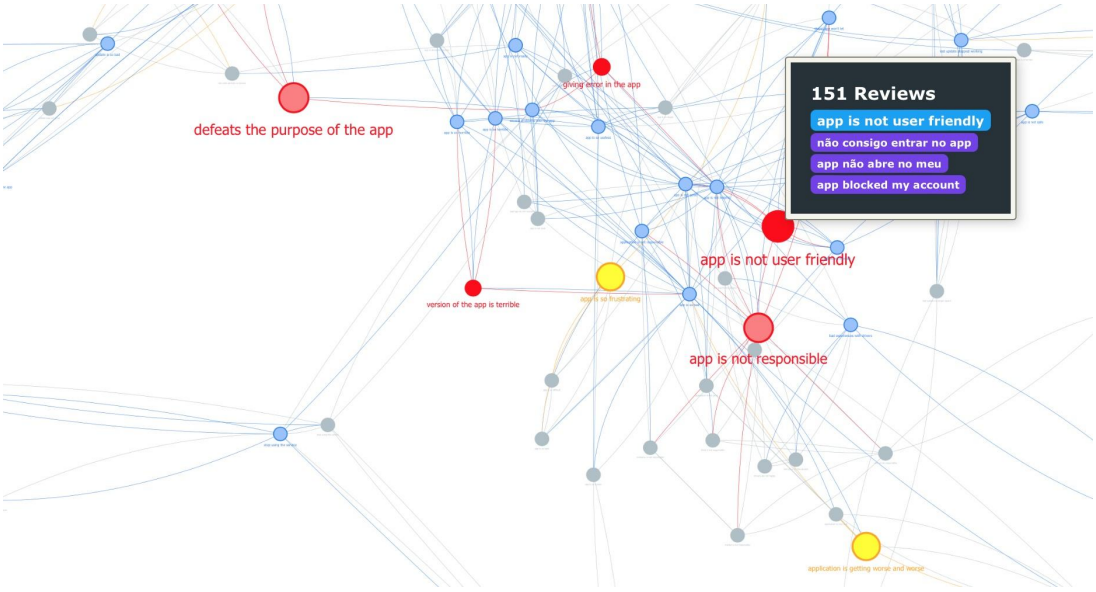

Figure 4.19: Heat map details. The size and color of each node represent the volume and the critical issue. The larger the node diameter, the greater the number of reviews associated with that issue. Colors vary from light gray to red

The trend detector time series is based on the moving average displayed in three colors on the lines. Red represents the uptrend, green represents the downtrend, and light blue represents stability, as shown in Figure 4.23.

We use a peak detection algorithm based on the dispersion principle (Z-score). If a new data point is a given x number of standard deviations from some moving average, the algorithm triggers $+1$ or -1. The system triggers an Uptrend alert if a $+1$ signal is returned. If a -1 signal is triggered, we alert a Downtrend.

Formally, a raw score x is converted into a standard score (Z-score) according to equation 4.4:

$$z = \frac{x - \mu}{\sigma} \tag{4.4}$$

where μ is the issues mean and σ is the issues standard deviation.

The absolute value of z represents the distance between this raw score x and the issues mean in standard deviation units. z is negative when the raw score is below the mean and positive when it is above.

The issue detector algorithm receives three parameters: (i) the moving window, (ii) threshold (the Z-score on which the algorithm signals), and (iii) influence (value between 0 and 1 that implies the influence of new signals on the average and in standard deviation).

With the risk matrix output, we generate a stacked area time series with three priority levels of issues stacked on each other. Therefore, we can compare the evolution of the whole and the contributions of individual parts over

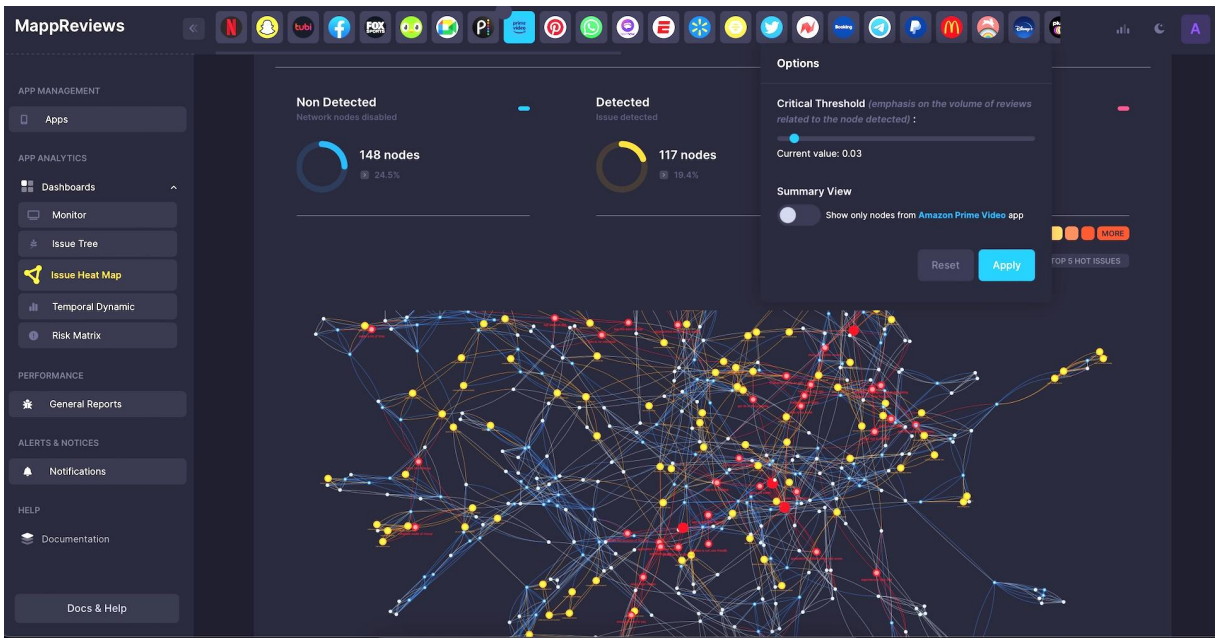

Figure 4.20: Variations in the color scheme of the heat map graph based on the sensitivity threshold

time, as shown in Figure 4.24.

In addition, we present trend detection signals, i.e., a time series emphasizing trend detection points. Red areas represent uptrends, and downtrends are represented by green areas, as shown in Figure 4.25. Each point in the time series has a complete report of all reviews and detected issues, as shown in Figure 4.26.

4.4.6 Notification System

Issue alerts occur when MApp-IDEA monitoring detects risks associated with an increased volume of issues. MApp-IDEA generates intelligent alerts to notify the team about changes, high-risk issues, or environmental failures, as shown in Figure 4.27.

Due to the way the issue detector module is implemented, even if there is no defective functionality related to a software requirement in the app, the MApp-IDEA alert system will still be able to identify failures in the environment, e.g., offline resources or processing web requests taking longer than usual, decreasing database latency, or app policy changes. These environmental issues are also frequently reported by users, although described differently.

The purpose of alerts is to quickly identify and resolve issues that affect app availability, speed, and functionality without manual monitoring. Using automatic features to monitor apps and generate alerts, developers can minimize downtime and reduce diagnostic time and the associated high cost.

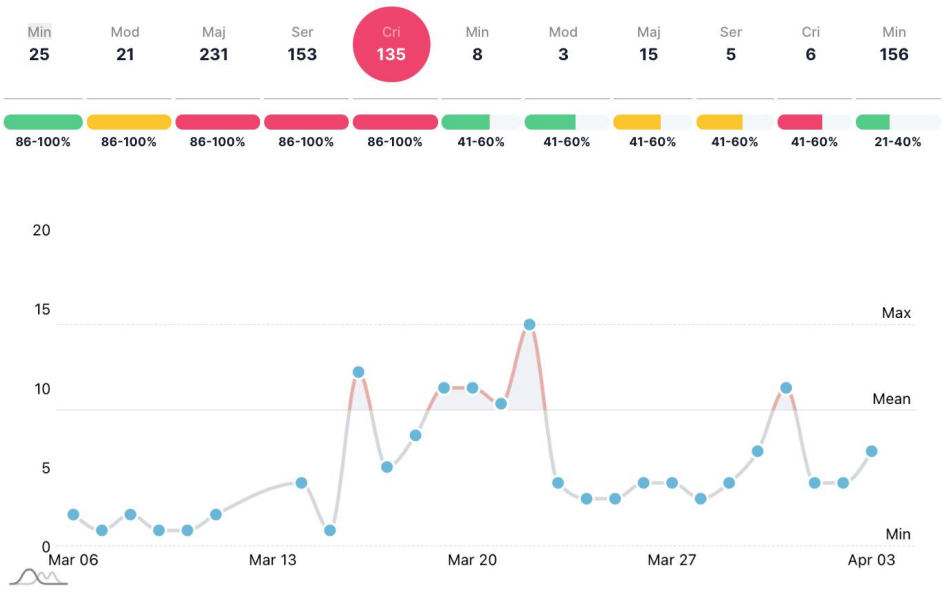

Figure 4.21: Time evolution of a risk matrix cell

Figure 4.22: General report of issue classification distribution in 3 critical levels

4.4.7 Online Performance Reports

The MApp-IDEA 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. In addition, it presents a summary of the total reviews collected and processed and the total number of issues detected. A sample screen is shown in Figure 4.28.

4.5 Experiment Design

We conducted a quasi-experiment (a type of empirical study where the assignment of treatments to subjects is not random) to evaluate the approach presented in this paper (Wohlin et al., 2012). To do so, we followed the guidelines proposed by Wohlin et al. (2012). The experimental design is detailed in

Figure 4.23: Uptrend detector based on moving average of the time series displayed in three colors on the lines

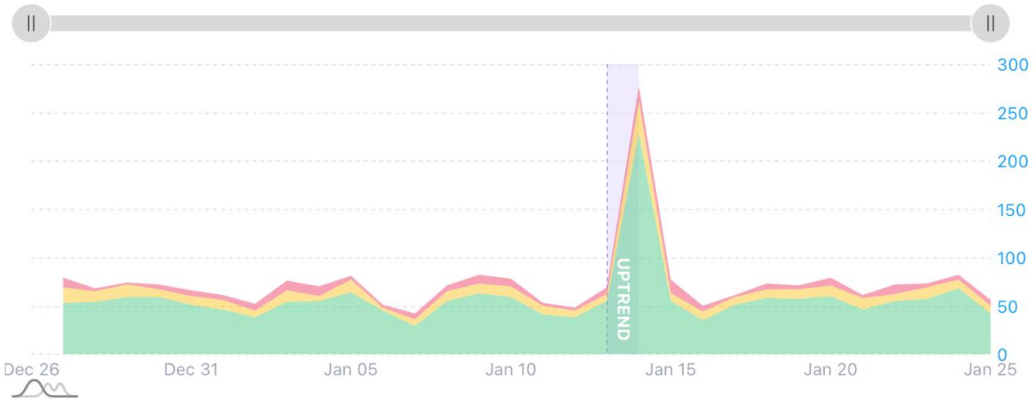

Figure 4.24: Stacked area chart with three priority levels of issues stacked on each other

the remainder of this section.

4.5.1 Definition of Research Questions

Our central research question is: *how do we prioritize and treat app reviews in time so that the app is competitive and guarantees the timely maintenance and evolution of the software?*

To answer this main question, we divided it into four specific research questions, as follows:

- **RQ1**: Can we detect issues using an unsupervised approach that is competitive compared to supervised methods and requires less effort?
- **RQ2**: Can we apply opinion mining to detect issues and monitor their evolution over time?

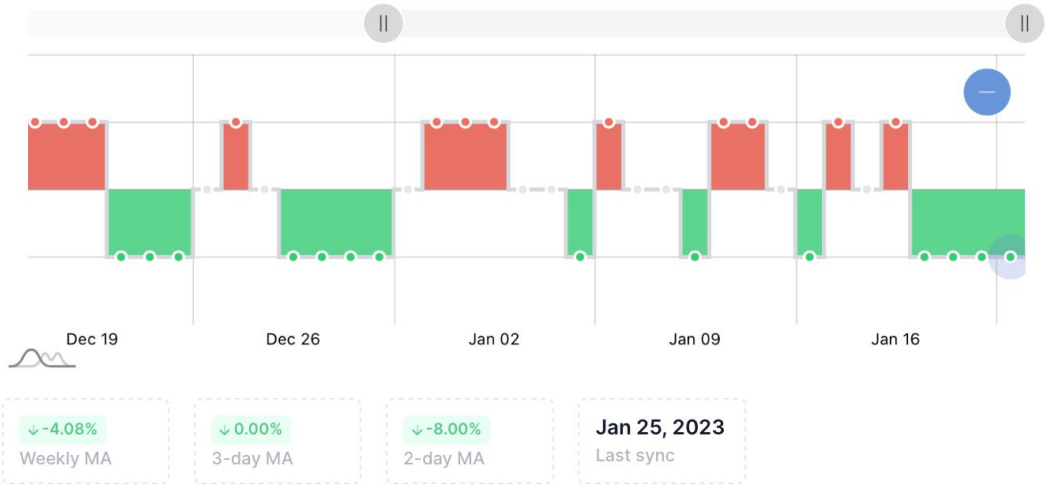

Figure 4.25: Trend detector to emphasize uptrends and downtrends in the time series

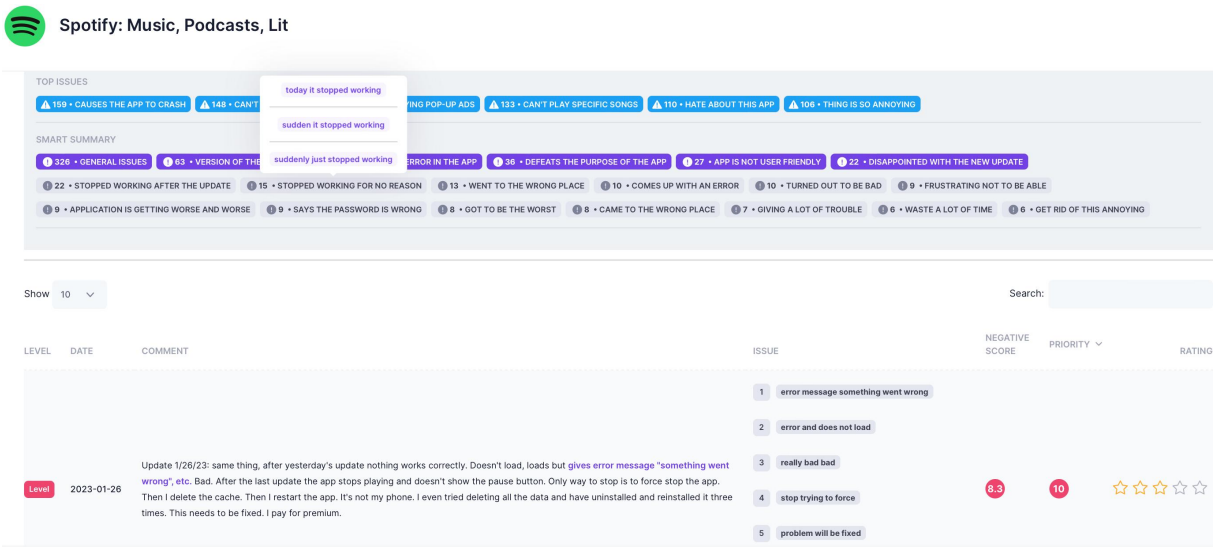

Figure 4.26: Details of a point in the Spotify app time series.

- **RQ3**: Can we prioritize issue candidates at different priority levels?
- **RQ4**: Can we monitor the relationship between release dates and issue uptrends?

The experiment was conducted to address the research questions presented in this section.

4.5.2 Experiment Definition

The experiment is defined as follows (Wohlin et al., 2012):

- analyze MApp-IDEA issue detector classifier and prioritization method,
- concerning effectiveness, efficiency, and perceived quality,

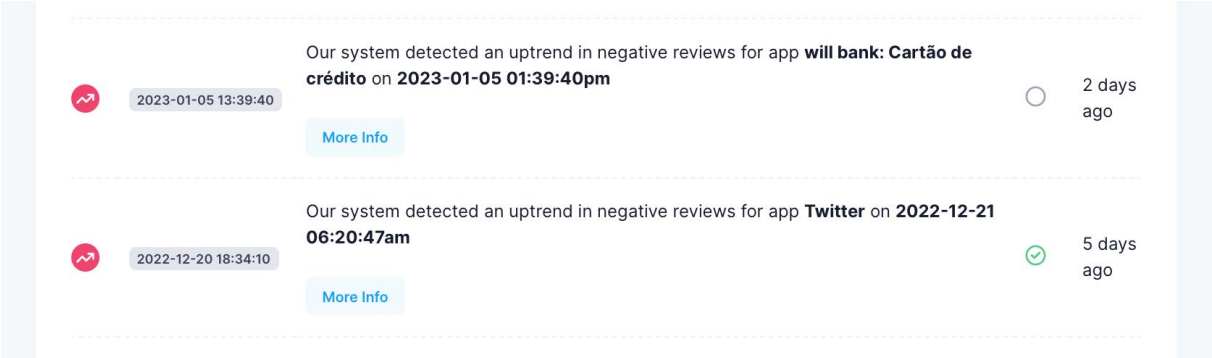

Figure 4.27: Notification system when there is an upward or downward trend of issues

Table 4.1: Stages of the experiment.

	MApp-IDEA Experimentation	
	Stage 1 (S1)	**Stage 2 (S2)**
Experimental Object	Issue detection classifier	Issue prioritization approach
Research questions	RQ1 RQ2	RQ3 RQ4

- from the point of view of the researcher,
- in the context of crowd feedback from app user reviews.

4.5.3 Preparation and Planning

The experiment plan comprises the sample selection, description of the experimental package, definition of variables, and description of employed design principles. To better understand the experiment plan, we separate the evaluation into two stages: (i) issue detection approach and (ii) issue prioritization approach.

Table 4.1 shows the stages related to research questions and experimental objects.

4.5.3.1 Stage 1 (S1) - Issue Detection

Our strategy uses an unsupervised approach. To evaluate the performance of the issue detection approach, we compare our strategy with well-known classifiers that use a supervised approach.

4.5.3.2 S1 Sample Selection

To evaluate the issue detector model, we used two annotated datasets, described by Table 4.2.

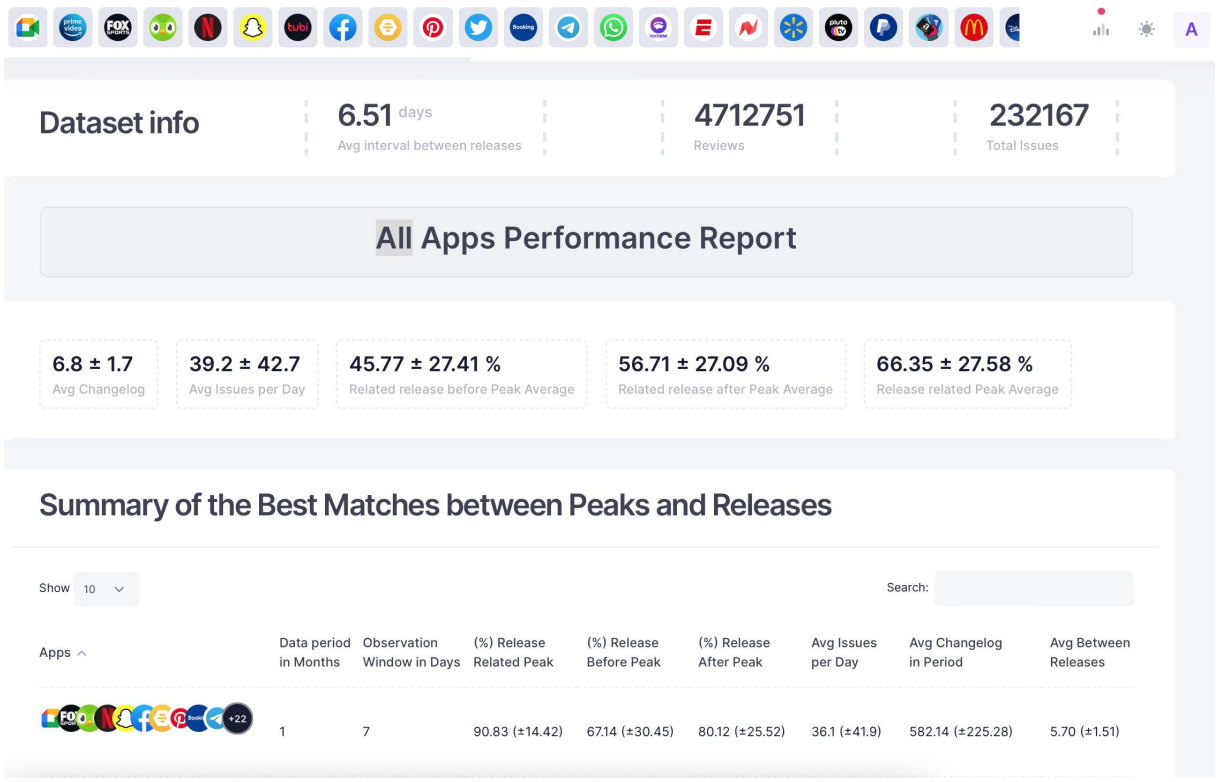

Figure 4.28: Notification system when there is an upward or downward trend of issues

4.5.3.3 S1 Experimental Package

The experimental package is publicly available on GitHub [1] for reproducibility. It has the following document:

- Description of the reference dataset: The experiment package describes all the information each object uses.
- Definition of objects: We use Python programming to define each object. The source code is available on GitHub.
- Description of the hyperparameters for each machine learning classification algorithm: We chose five supervised classification algorithms that are used in state of the art to support issue detection: k-Nearest Neighbor (kNN), Multi-layer Perceptron (MLP), Multinomial Naive Bayes (MNB), Random Forest (RF), and Support Vector Clustering (SVC). In addition to these classifiers, a Dummy Classifier (DC) makes predictions that ignore the input features. This dummy classifier is a simple baseline to compare against other more complex classifiers. The stratified strategy that generates predictions by respecting the class distribution of the training data was used.

[1] https://github.com/vitormesaque/mapp-idea

Table 4.2: Annotated reviews from tweets.

Dataset	Type	Reviews	Lang.	Classes	Filter by class
TEN	Tweets	10364	English	1) Problem found that must be fixed. 2) Wish or Request for a new feature. 3) Irrelevant noisy feedback.	Class (1)
BFRU	Tweets	3691	English	1) User explanation for a given score; 2) Experience of using a specific feature. 3) Problem found that must be fixed. 4) Wish or request for a new feature.	Class (1) if $starRating \leq 1$ Class (2) if $starRating \leq 1$ Class (3)

4.5.3.4 S1 Variables

The independent variable (factor) controlled in the experiment was the issue detector approach. The values assigned to this variable (treatments) are seven classifiers (kNN, MLP, MNB, RF, SVC, DC, and MApp-IDEA). The dependent variable, which is affected by the treatment, is the F1-Score, a measure of the accuracy of a test (RQ1 and RQ2).

We use the F1 evaluation measure that corresponds to the harmonic mean of Precision (5.1) and Recall (5.2), where TP (True Positive) refers to the number of issues that were both extracted and annotated; FP (False Positive) are issues that were extracted but not annotated, and FN (False Negative) refers to the issues annotated but not extracted. Equation 5.3 defines the F1 measure.

$$P = \frac{TP}{TP+FP} \tag{4.5}$$

$$R = \frac{TP}{TP+FN} \tag{4.6}$$

$$F1 = \frac{2*P*R}{P+R} = \frac{2*TP}{2*TP+FP+FN} \tag{4.7}$$

4.5.3.5 S1 Experimental Design

In this quasi-experiment, the experimental design has one factor (issue detection approach) and seven treatments (KNN, MLP, MNB, RF, SVC, DC, and MApp-IDEA) with crossover (Wohlin et al., 2012).

4.5.3.6 S1 Operation of the Experiment

We evaluated our issue detector classifier through cross-validation with five widely used supervised models, using 80% training and 20% data testing. Data are classified into two classes, issues and non-issues.

4.5.3.7 Stage 2 (S2) - Issue Prioritization

To evaluate our issue prioritization approach, we analyzed the relationship between update release dates and issue spikes detected by our prioritization method.

4.5.3.8 S2 Sample Selection

We used the 50 most popular apps in the Google Play store from 20 domains between February 6th and February 10th, 2023, as detailed in Table 4.3 to evaluate the issue detector model. Our crawler downloaded the last 100.000 reviews for each app. Reviews from the last six months were considered to generate enough data for the apps to be compared fairly. For example, newer apps would have fewer reviews and releases than older apps. For some apps with high reviews, we used 200.000 or 500.000 reviews to reach the minimum 6-month data period for the analysis. Almost 5 million reviews of these apps were processed. In the evaluation, we analyze the reviews of the last six months of each app.

4.5.3.9 S2 Experimental Package

The experimental package is publicly available at GitHub [2] for reproducibility purposes. It has the following document:

- Description of the reference dataset: The experiment package describes all the information each object uses.
- Definition of objects: We use Python and PHP programming to define each object. We also use PostgreSQL, JavaScript, and CSS (Cascading Style Sheets) to explore the data. The base source code is available on GitHub.
- Description of the parameters for each run: We run the issue prioritization model by varying the window (w) in days and the period (p) in months, where w is the range of days observed before and after an issue peak and p is the total period of observed data. We evaluated by varying w from 1 to 7 and p from 1 to 6.

[2] https://github.com/vitormesaque/mapp-idea

Table 4.3: Apps used in evaluation grouped by category.

Category	**Apps**
Business	Google Chat / Microsoft Teams
Communcation	WhatsApp Messenger / Discord: Talk / Chat & Hang Out / Skype / Telegram / TextNow: Call + Text Unlimited
Dating	Bumble - Dating. Friends. Bizz
Education	Duolingo: language lessons
Entertainment	Amazon Prime Video / Disney+ / Paramount+ / HBO Max: Stream TV & Movies / MISTPLAY: Play to earn rewards / Netflix / Peak Streaming / Peacock TV: Stream TV & Movies / Pluto TV - Live TV and Movies / Tubi - Movies & TV Shows Rewarded Play: Earn Gift Cards /
Finance	Cash App / PayPal - Send, Shop, Manage / Venmo
Food & Drink	DoorDash - Food Delivery / McDonald's / Uber Eats: Food Delivery
Graphic & Design	Canva: Design / Photo & Video
Lifestyle	Amazon Alexa / Pinterest
Maps & Navigation	Waze Navigation & Live Traffic / Uber - Request a ride
Music & Audio	Deezer: Music & Podcast Player / Spotify: Music, Podcasts, Lit
News & Magazines	NewsBreak: Local News & Alerts
Shopping	Temu: Shop Like a Billionaire / SHEIN-Fashion Shopping Online / Walmart Shopping & Grocery
Social Media	Facebook / Instagram / Instagram / Snapchat / TikTok / Twitter
Sports	ESPN / FOX Sports: Watch Live
Travel & Local	Booking.com: Hotels and more / Hopper: Hotels / Flights & Cars
Utility & Productivity	Firefox Fast & Private Browser / Microsoft Excel: Spreadsheets
Video Call	Google Meet / Zoom - One Platform to Connect
Video Players & Editors	CapCut - Video Editor

4.5.3.10 S2 Variables

The independent variable (factor) controlled in the experiment was the MApp-IDEA issue prioritization approach. The values assigned to this variable (treatments) are window (ranging from 1 to 7) and period (ranging from 1 to 6). The dependent variable, which is affected by the treatment, is the match-related peaks with release dates measured with mean and standard deviation (RQ3 and RQ4).

A mean is a number expressing a data set's central or typical value. Variance is the sum of squares of differences between all numbers and means. Standard Deviation (S) is the square root of variance. It measures the extent to which data varies from the mean, as follows equation 4.8.

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{4.8}$$

, where N is the total number of elements or frequency of distribution.

4.5.3.11 S2 Experimental Design

In this quasi-experiment, the experimental design has one factor (MApp-IDEA issue prioritization approach) and two treatments (window and period) without crossover (Wohlin et al., 2012).

4.5.3.12 S2 Operation of the Experiment

We analyze the relationship between the high volume of issues detected by MApp-IDEA and software updates to evaluate the issue prioritization approach. We assume that the spikes should be associated with releases of new features, general improvements, and bug fixes. So, if a peak of problems is detected one day after an update, we can say that this update triggered this peak of complaints. When launching a new release, the development team makes publicly available what was done, improved, or corrected in the application in that release. Thus, in addition to relating the high volume of increased issues to a defective release, we can analyze the change log of the next release after the peak to validate whether the correction made by the development team is related to the previous peak of issues. Not all developers report the changes in those releases and provide default text. However, we explore some cases where the developer has made updated details available in the data analysis section.

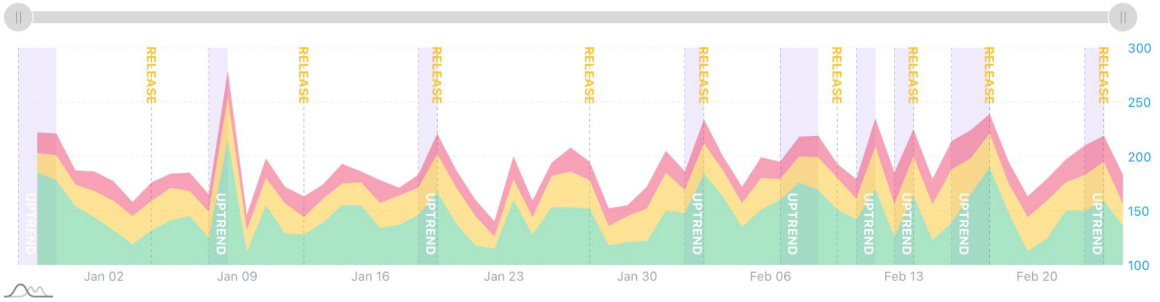

Figure 4.29: Time series of the Instagram app between January 1st and March 1st, 2023, stacked lines by criticality levels with the relationship between peak points and release dates

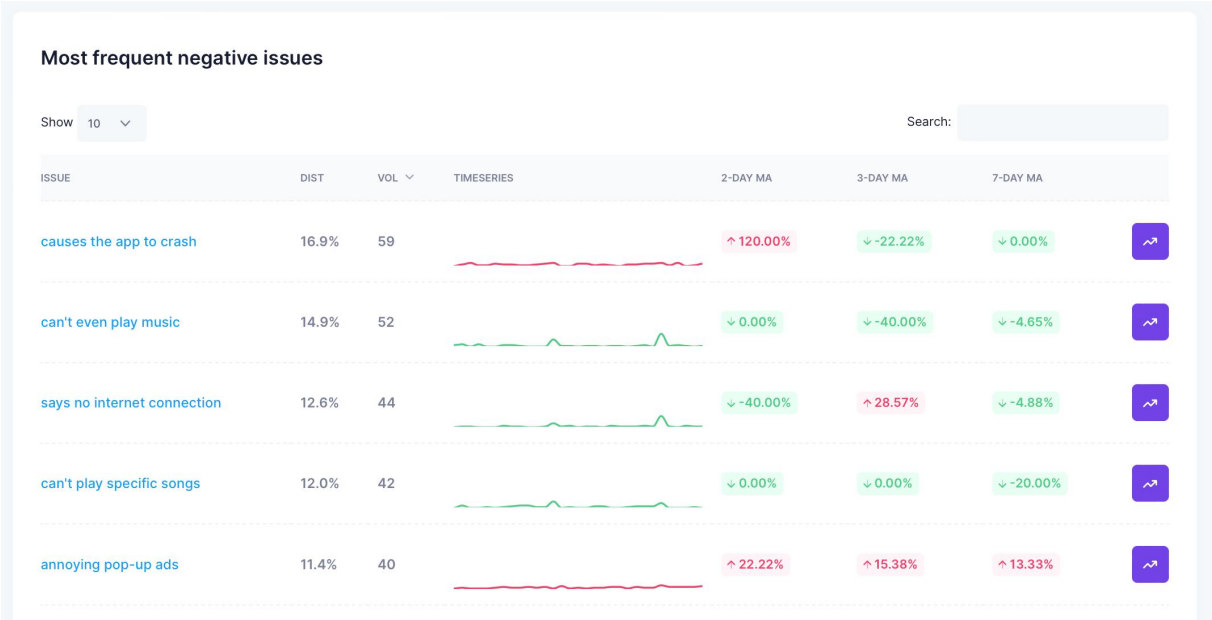

Figure 4.30: Ranking of top issues within a selected time range

4.6 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 Table 4.4. For the BRFU dataset, our classifier obtained an F1-Score of 0.8027±0.0350, only 0.9473±1.0453 less than the MPL classifier, as shown in Figure 4.31.

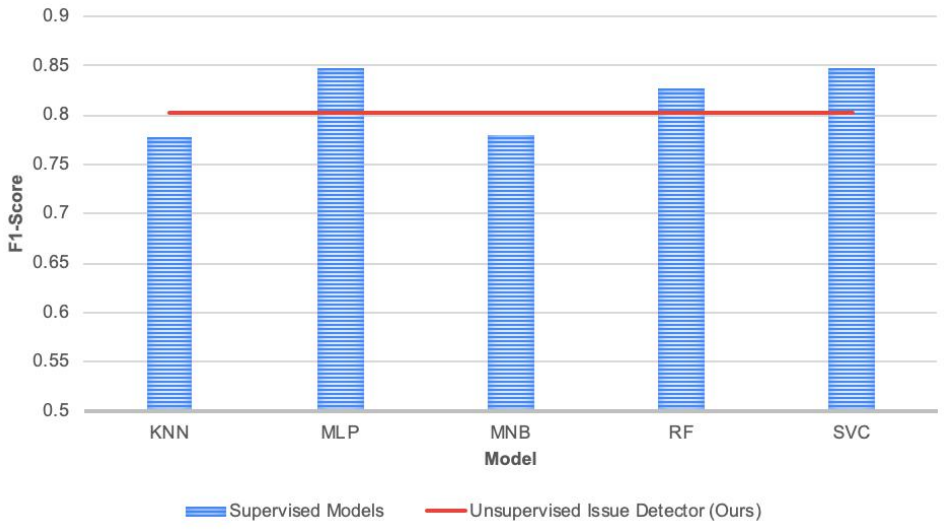

Figure 4.31: Comparison chart of our approach with supervised approaches

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.

Table 4.4: Average F1-Score of the issue detector compared to supervised models.

* Model	F1-Score Average and Standard Deviation				
	BRFU		TEN		Total
	AVG	SD	AVG	SD	AVG±SD
DC	0,7405230909	0,0039126259	0,513355459	0,00624702464	0,626939275 ± 0,119828659
k-NN	0,7784388303	0,0076876306	0,598531535	0,03378892565	0,688485182 ± 0,097593118
MLP	0,8474143289	0,0335658706	0,601225506	0,04136394154	0,724319917 ± 0,134525055
MNB	0,780618454	0,0036905403	0,622960909	0,07428282406	0,701789681 ± 0,096762013
MApp-IDEA	0,8027877657	0,0350887219	0,612294485	0,07823482048	0,707541125 ± 0,115531067
RF	0,8272610402	0,0156180859	0,663513493	0,07302492679	0,745387266 ± 0,099632327
SVC	0,8473141606	0,0295036641	0,665387544	0,08281196334	0,756350852 ± 0,112376541
Total	**0,803479667**	**0,042666978**	**0,61103841**	**0,073786739**	**0,70725904 ± 0,11389653**

Regarding our approach for prioritizing issues, we analyzed Table 4.5 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. The family-wise significance level of the tests is $alpha = 0.050$. We failed to reject the null hypothesis that the population is normal for all populations (minimal observed $p - value = 0.479$). Therefore, we assume that all populations are normal. We applied Bartlett's test (Bartlett, 1937) for homogeneity and failed to reject the null hypothesis ($p = 0.704$) that the data is homoscedastic. Thus, we assume that our data is homoscedastic. Because we have more than two populations and all populations are normal and homoscedastic, we use repeated measures Analysis of Variance (ANOVA) (Girden, 1992) as an overall test to determine if there are any significant differences between the mean values of the populations. If the results of the ANOVA test are significant, we use the post-hoc Tukey HSD test (Tukey, 1949) to infer which differences are significant. We report each population's mean value (M) and standard deviation (SD). Populations are significantly different if their confidence intervals are not overlapping. We reject the null hypothesis ($p = 0.000$) of the repeated measures ANOVA that there is a difference between the mean values of the populations Before (M=43.633±10.524, SD=16.116), After (M=54.607±10.524, SD=12.284), and All (M=64.156±10.524, SD=17.422). Therefore, we assume that there is a statistically significant difference between the mean values of the populations. Based on the post-hoc Tukey HSD test (Tukey, 1949), we assume no significant differences within the following groups: After and All. All other differences are significant.

Our statistical analysis reveals a strong correlation between peaks and up-

Table 4.5: Summary of results by window.

Window (days)	Release Related Peak (%)		
	All	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

Table 4.6: General results separated by groups.

Group	Metrics			Release Related Peak (%)		
	Number of reviews	Avg Changelog	Avg Issues per Day	Before	After	All
All	12035 - 370937	6.9 ± 1.7	37.7 ± 42.7	43.85 ± 26.02	54.69 ± 25.40	64.34 ± 27.16
Low	12035 - 26502	6.2 ± 2.3	6.3 ± 3.0	58.55 ± 34.09	71.24 ± 30.92	76.87 ± 27.33
Medium	46764 - 119486	7.0 ± 1.7	30.5 ± 26.4	42.05 ± 24.52	51.84 ± 23.53	61.58 ± 26.45
High	204694 - 370937	6.3 ± 0.6	43.6 ± 29.8	52.26 ± 30.73	55.76 ± 25.15	68.77 ± 29.54

ward trends in the time series and app release dates. This is illustrated in Figure 4.29, which presents a stacked line representation of the Instagram app's time series, categorized by criticality levels. The results indicate that app releases tend to occur more frequently after detecting issue spikes. However, it is noteworthy that only 50% of releases occur within three days or less after the issue peak, and approximately 66% occur within seven days.

Table 4.6 further supports this relationship, showing a significant association between release dates and detected issue spikes. Specifically, when considering a window of three days before and three days after the peak, an average of 39.19% ± 21.95% of releases occurred before the peak, while 50.88% ± 22.59% occurred after the peak. This pattern holds true across various observation window sizes. Notably, as the observation window decreases, a positive correlation becomes more pronounced between patch releases and the releases that caused issues. This implies that issues resulting from a release may take several days to manifest and impact the overall volume of reported issues.

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.

A commonly used statistical measure to assess risk is the standard deviation, which quantifies the dispersion of data in a distribution. A higher standard deviation indicates more significant variability around the mean. In Table 4.6, the standard deviation of issues per day is reported as 37.7 ± 42.7%. This indicates that there are periods of significant variation, suggesting the presence of issue peaks in the time series. Such variations are of interest to us as they help identify outliers.

An interesting finding in our results is that approximately 64% of updates are associated with issue spikes. This aligns with the findings of a previous study by Mcilroy et al. (2016a), which reported that 63% of updates are bug fixes.

We conducted an empirical analysis and observed the generated results regarding the prioritization of levels using our risk matrix. Figure 4.32 illustrates issues allocated in the lower-left position of the matrix (i_1, j_5), indicating a low priority level. In this context, a low priority level corresponds to bugs, enhancements, or requests that are perceived as optional and do not significantly impact the app's functionality or performance (Malgaonkar et al., 2022). On the other hand, at the top-right position of the matrix (i_5, j_1), we have a high priority level that represents bugs, enhancements, or requests considered critical and have a significant impact on the app's functionality or performance (Malgaonkar et al., 2022).

Level	Date	Priority	Impact	Detected Issues
1	2023-03-23	1	0	1 songs are a bit repetitive
1	2023-03-22	1	0	1 songs are not clear
1	2023-03-22	1	0	1 limited selection of songs
1	2023-03-21	1	0	1 poor selection of songs

Figure 4.32: Issues assigned a low priority level

4.7 Threats to Validity

In this section, we present threats to the validity of our experiment. The identified threats are discussed in four types of validity: construct validity, in-

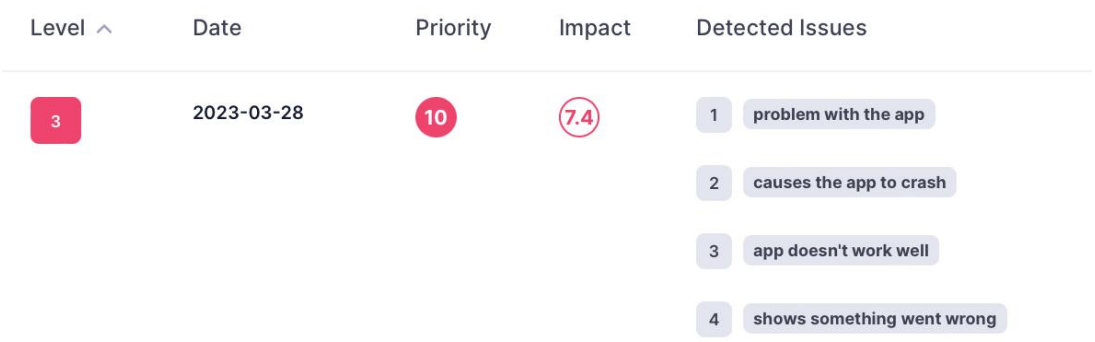

Figure 4.33: Issues assigned a high priority level

Table 4.7: Summary of results by the period in months.

Data Period	Release Related Peak (%)			Metrics		
	All	Before	After	Avg Issues per day (%)	Changelog in period	Avg between releases (%)
1	59.01 ±19.55	41.60 ±17.53	50.28 ±19.61	39.5 ±43.6	1804.00 ±665.42	7.22 ±1.64
2	59.93 ±18.38	44.06 ±17.53	49.16 ±17.78	37.4 ±43.0	2274.07 ±887.19	7.21 ±1.71
3	61.26 ±19.71	45.26 ±17.85	51.73 ±19.76	40.0 ±43.6	2800.00 ±1087.98	7.05 ±1.76
4	74.07 ±21.80	43.17 ±28.15	62.57 ±29.86	34.8 ±42.0	559.26 ±214.75	5.92 ±1.55
5	69.62 ±22.44	42.39 ±20.39	61.71 ±20.92	36.9 ±42.8	962.50 ±332.68	6.79 ±1.71
6	60.54 ±22.00	44.82 ±22.01	52.40 ±19.26	39.4 ±42.3	1360.87 ±498.47	7.19 ±1.67

ternal validity, external validity, and conclusion validity (Wohlin et al., 2012). Construction validity concerns the relationship between theory and observation, i.e., if the treatment reflects the cause well, the result reflects the effect well. Inferences about the causal relationship between treatments and outcomes are considered for internal validity. External validity is the generalizability of our experimental results. The validity of concluding refers to the correct inference about the relationship between the treatment and the experimental result (Wohlin et al., 2012).

4.7.1 Construct validity

Our experiment aims to evaluate the performance of our issue detection model compared to supervised models. We used a cross-validation scheme to compare our model directly with supervised models. To compare our proposal with supervised models that use 80% training and 20% testing, we compare the same 20%.

To assess issue prioritization ability, we correlated issue spikes and app releases. If there is an issue that becomes severe over some time, that issue was most likely introduced by a faulty app update. Going in, we need a volume of reviews for this to work. To mitigate this, we consider apps with a minimum review volume.

4.7.2 Internal validity

In the validation of the issue detection method, we depend on the quality of the dataset annotation to verify the classification. However, in our study, all classification methods run with the same data preprocessing and default configuration. No results were obtained through special preprocessing steps or adjustment settings according to the evaluated method.

When evaluating issue prioritization, we must ensure that app releases are properly reported in the app stores. However, some updates can be done without going through the app store, e.g., Web-View, remote database response latency, and server crashes. To mitigate this, we drop apps that do not have a minimum number of releases.

4.7.3 External validity

To use MApp-IDEA in a real scenario is necessary to have a sufficient number of revisions distributed over time, that is, a minimum number of available time series observations for the problem detection and prioritization model to work correctly. Therefore, our method is best suited in practical terms when large volumes of app reviews can be analyzed.

4.7.4 Conclusion validity

To reach the correct conclusion about the relationships between treatments and experimental results, we took the following precautions: (i) we followed the recommendations presented by (Wohlin et al., 2012); (ii) objective measures were adopted to increase the reliability of the results; (iii) treatments were uniformly applied to all datasets; and (iv) the study was carried out in a laboratory environment under the same circumstances.

To increase the reliability of our study analysis, we chose only objective measures with the independent variable. We used the statistical analysis proposed by (Herbold, 2020), a framework for comparing (multiple) paired populations.

4.8 Final Remarks

In this chapter, we introduce the MApp-IDEA method to detect issues, classify the issues in a risk matrix with prioritization levels, and monitor the temporal dynamics of issues and risks. Our method contributes an efficient solution to the research question of prioritizing and addressing user reviews in time so that the app is competitive and guarantees the timely maintenance and evolution of the software.

The findings demonstrate that the opinions extracted from user reviews provide information about app issues and risks. The results indicated that our unsupervised approach is competitive compared to supervised approaches for issue detection. Our method can prioritize issues in criticality levels and monitor the temporal evolution of issues and risks. MApp-IDEA aims to reduce the gap between issue incidence and identification by introducing an automated dashboard that enables developer access to automatic notifications, alerts, issue heat map networks, issue trees, issue time series, and issue detector trends. MApp-IDEA allows the prioritization of issues through a dynamic and user-friendly risk matrix while also monitoring temporal trends in identified risks.

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.

CHAPTER

5

Learning Risk Factors from App Reviews: A Large Language Model Approach for Risk Matrix Construction

5.1 Introduction

The analysis of mobile app reviews enables the identification of trends and issue patterns that can affect user experience and app reputation in app stores (Genc-Nayebi and Abran, 2017). Based on this analysis, developers can prioritize bug fixes, add requested features, and respond to user complaints to improve app quality and increase positive ratings. To achieve this, it is necessary to link user feedback extracted from reviews with app development and maintenance practices (Araujo et al., 2021).

A simple and intuitive way to organize and prioritize actions for software maintenance, aiming to reduce negative ratings, is through a risk matrix (Xiaosong et al., 2009; Pilliang et al., 2022). 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 5.1. 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 (Paltrinieri et al., 2019), 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.

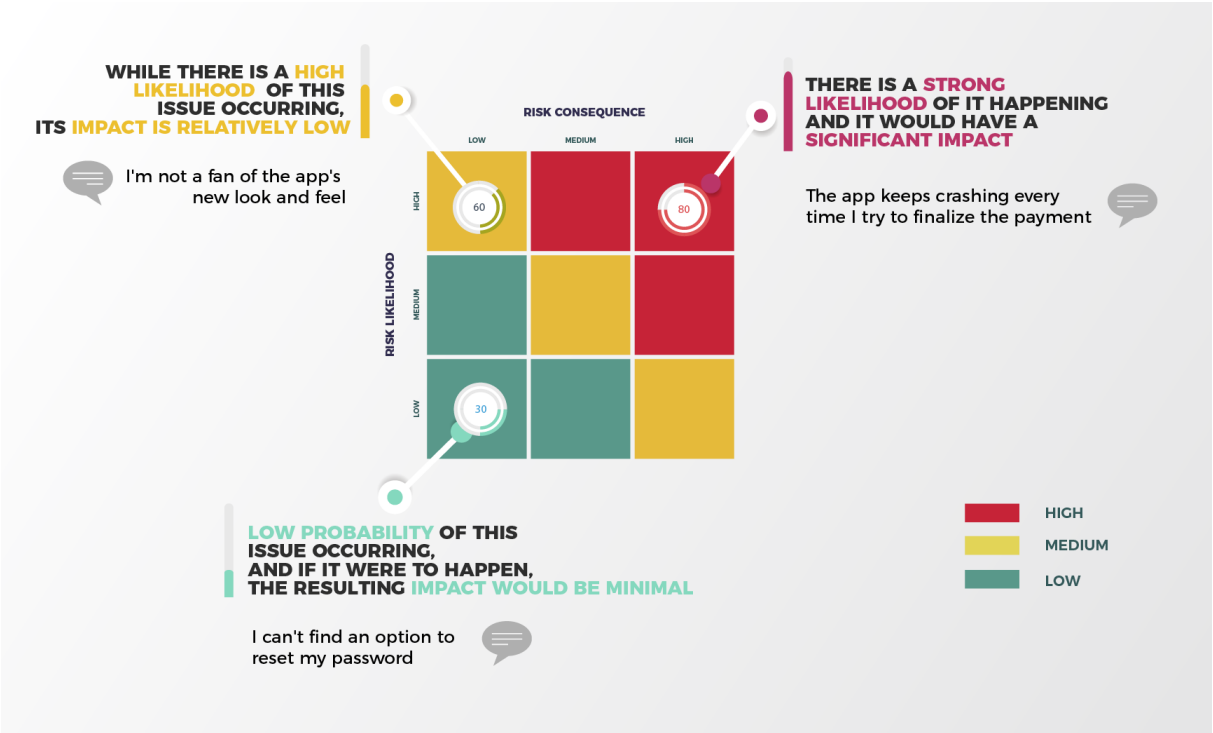

Figure 5.1: Risk matrix

Some initiatives in the literature already automate risk matrix generation using machine learning methods. For example, Chaouch et al. (2019) and Hammad and Inayat (2018) use Scrum with risk matrix, but with manual assessments. A recent study proposed using language models (e.g., BERT) and data clustering methods (e.g., K-Means) to automate risk matrix generation in software development projects (Pilliang et al., 2022). However, such studies have not yet explored the app review domain and rely on manually constructed resources, such as a vocabulary or lexicon to define risk priority. Another limitation is the lack of a step to extract app features from reviews, which is crucial for the development team.

To address this gap, we argue that recent Large Language Models (LLMs) are promising to automatically construct a risk matrix using information extracted from app reviews, such as features and bugs, organized according to the probability of occurrence and impact/severity to the app ratings. LLMs can analyze and understand complex contextual information present in review texts due to their extensive pre-training corpus (Ross et al., 2023).

This chapter 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) (Zhang et al., 2022). While large-

scale language models like GPT (Brown et al., 2020) 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. Our contributions 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.

We conducted experimental evaluations using a database of reviews from eight mobile apps. Through the proposed approach, we constructed risk matrix for each app and compared them with their respective reference risk matrix constructed with annotated data. The experimental results demonstrate that, with proper prompt optimization, OPT models are capable of generating a competitive risk matrix compared to GPT. While there is room for improvement compared to reference risk matrices, our results indicate a significant step toward the maintenance and evolution of software products, enabling feature prioritization that requires more attention from developers.

The rest of this chapter is structured as follows. Section 5.2 provides the background and discusses related works in the field. Section 5.3 presents the LLM-based risk matrix learning approach applied to app reviews with a focus on dynamic prompt construction for feature extraction (5.3.1), the estimation of review impact (5.3.2), and addresses the estimation of occurrence likelihood 5.3.3. Section **??** presents the experimental evaluation, describing the

datasets used (**??**), experimental setup (**??**), and discussing the results (5.4.9). Section 5.5 highlights the limitations of the approach, and finally, Section 5.6 offers the final remarks of the chapter, summarizing the main findings and contributions.

5.2 Background and Related Works

Early initiatives for analyzing reviews focusing on software maintenance explore named entity extraction techniques (such as software features) from reviews using predefined linguistic rules, such as Safe (Johann et al., 2017), GuMA (Guzman and Maalej, 2014), and ReUS (Dragoni et al., 2019). These approaches require sets of linguistic patterns, relying on experts to constantly update them. With the advancement of machine learning methods, models are trained to identify the entities of interest from a large set of labeled data. An example of such an approach is RE-BERT (Araujo and Marcacini, 2021), which uses annotated data to fine-tune a pre-trained BERT model to identify features or software requirement candidates mentioned by users in reviews.

With the recent emergence of Large Language Models (LLMs), such as GPT-3 and OPT (Zhang et al., 2022), opinion mining and sentiment analysis have also evolved to incorporate such models, although still with few applications in the context of app reviews. These models have architectures with billions of parameters and are pre-trained on large amounts of text, thereby providing capabilities for understanding and extracting knowledge from textual data. While the general aim of these models is text generation, their outputs can be conditioned through instructions or prompts (Strobelt et al., 2022). For example, the task of feature extraction or entity extraction from reviews can be performed using a paradigm called few-shot learning (Logan IV et al., 2022). In this case, the model receives a prompt that provides information about the type of feature to be extracted and a few samples related to that feature. It is worth noting that, in addition to feature extraction, the same LLM can also be used to identify the impact of reviews according to their severity level, where prompts can be used to guide the model in classifying the severity of the evaluations.

Another common strategy for app review analysis is the use of clustering methods or topic modeling based on review characteristics to organize them according to their similarities (Noei et al., 2021). This allows for identifying groups of reviews that mention similar problems, indicating the likelihood of bugs or complaints related to those issues. By combining feature extraction, severity classification, and the identification of the likelihood of app reviews, we can automate the construction of risk matrices (Pilliang et al., 2022). These

matrices can provide an overview of the risks associated with an app based on the information extracted from user reviews.

A review of the existing studies reveals various aspects related to risk management in software projects. In (Xiaosong et al., 2009; Chaouch et al., 2019; Hammad and Inayat, 2018), the authors discuss risk management in agile software development projects. Xiaosong et al. (2009) presents basic risk management concepts for software development projects, while Chaouch et al. (2019) proposes a framework for integrating risk management in agile projects using Scrum. Additionally, Hammad and Inayat (2018) also explores the integration of risk management in the Scrum framework, highlighting the importance of an iterative risk management process for project success.

In Hammad et al. (2019) and Ionita et al. (2019), the authors focus on identifying the risks faced by agile development practitioners and mitigation strategies. (Hammad et al., 2019) reveals that project deadlines and changing requirements are the most commonly encountered risks, while Ionita et al. (2019) proposes a framework that uses a risk assessment process to prioritize security requirements.

Pilliang et al. (2022) propose a risk matrix model for software development projects, using natural language processing and machine learning techniques to prioritize risks. The proposed model offers an approach for the automated construction of risk matrices by combining sentiment analysis based on lexicons or vocabularies to identify the impact of the risk and clustering methods to identify the likelihood of occurrence.

Although these studies address different aspects of risk management in software projects and the application of risk matrices, it is important to highlight some general limitations. Most studies focus on specific contexts, such as agile development or information security, which may limit the generalizability of their findings to other types of software projects. There is a gap in the literature on generating risk matrices from app reviews. Our focus is on generating a risk matrix based on reviews, especially in scenarios with little labeled data, which can be used in a wider range of applications at different stages of software development, from its conception to its maintenance.

5.3 LLM-based Risk Matrix Learning from App Reviews

The proposed approach leverages Large Language Models (LLMs) to extract relevant information from user reviews and utilize it in constructing a risk matrix. In general, the idea is to exploit the text generation capability of an LLM, but conditioned for a specific task.

Formally, the task of predicting the next word in an LLM can be formulated

as finding the most likely word $P(w_{i+1}|w_1, w_2, ..., w_i)$ given a sequence of words $w_1, w_2, ..., w_i$. This probability is estimated using the neural weights of the LLM, which is pre-trained on large textual corpora. The pretraining of the LLM is accomplished through a process called autoregressive language modeling in a Transformer neural architecture. During pretraining, the model learns to capture linguistic patterns and construct representations of words and phrases that can be used to predict the next word in a sequence.

Considering the context of software reviews as input, the proposed approach aims to condition the prediction of the next word through a prompt. Now, the conditioned probability $P(w_{i+1}|w_1, w_2, ..., w_i; \Theta)$, where Θ represents the prompt, is used to guide the prediction of the next word.

In the proposed method, we use the Open Pre-trained Transformers (OPT) as the Large Language Model (LLM). Most models available through APIs do not provide access to the full model weights obtained during pretraining, making it difficult to study them in detail and reproduce the experimental results. On the other hand, the OPT was developed to overcome this limitation by providing pre-trained models with different numbers of parameters. For example, models range from 125 million to 175 billion parameters. The authors of OPT conclude from their experiments that OPT-175B is comparable to GPT-3 (Zhang et al., 2022). However, some smaller models can achieve promising results through the appropriate use of prompts, as proposed in the next section.

Figure 5.2 provides an overview of the our method used for constructing the risk matrix.

5.3.1 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 (Reimers and Gurevych, 2019), 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 represent-

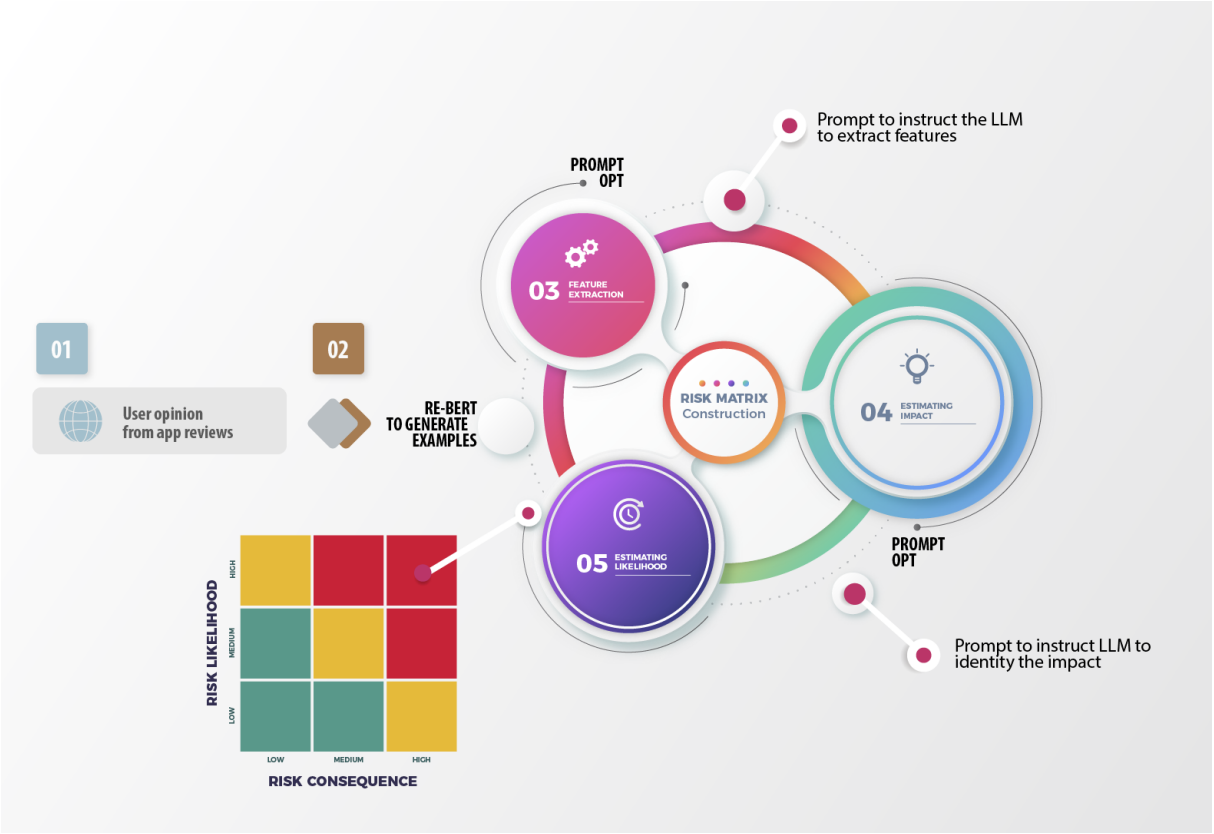

Figure 5.2: The overview of our Risk Matrix construction method

ing 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 similarity is commonly measured by the cosine of the angle between the vectors. Formally, to find the k-nearest neighbors of a target review R_i, we denote this list as $KNN(R_i)$ and define it as $KNN(R_i) = argmax_k(sim(e(R_i), e(R_k)))$, where $sim()$ represents the similarity function and $argmax_k$ returns the k indices corresponding to the most similar reviews to R_i.

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.

Figure 5.3 shows a prompt generated for the Instagram app. In blue are examples identified by similarity from the knowledge base generated through reviews and features of other applications. In red, it is the review to be processed. The model is induced to generate a list of features from the review after the "@" symbol.

PROMPT TO INSTRUCT THE LLM TO EXTRACT FEATURES

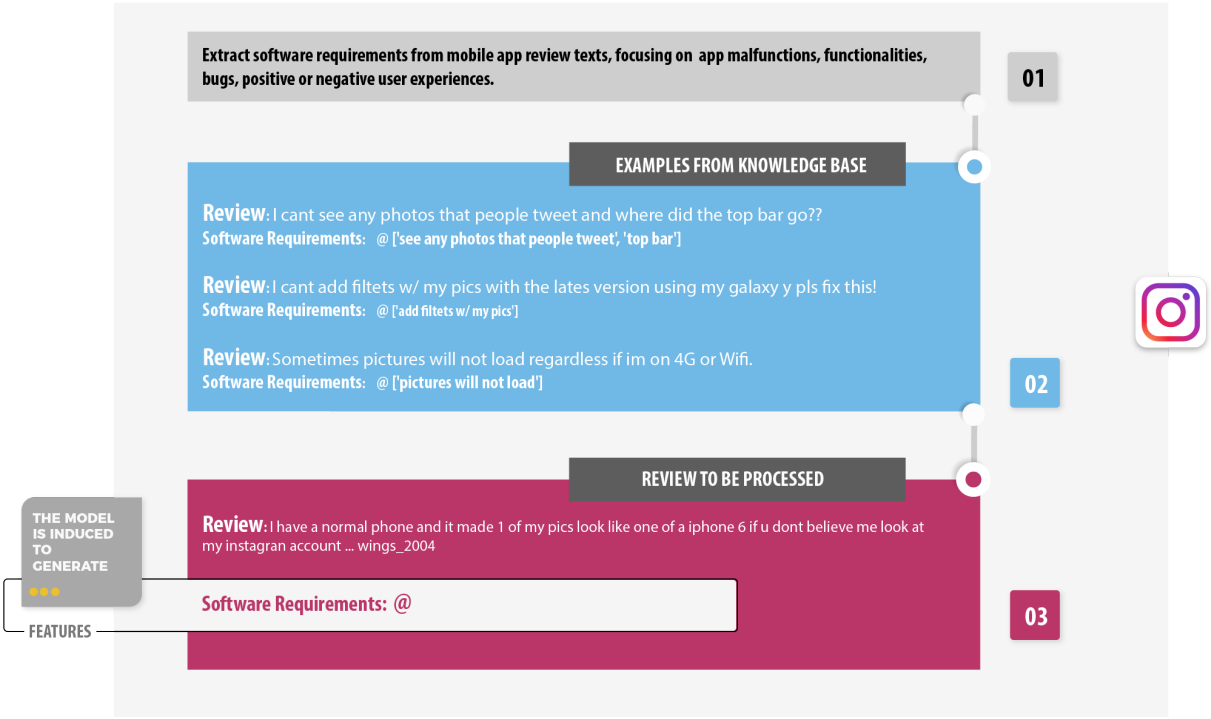

Figure 5.3: Example of a prompt generated for the Instagram app

5.3.2 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. Figure 5.4 presents an example of the prompt used. Note that we provide the prompt constructed along with the feature and let the model complete the severity classification. Unlike the previous step, we condition the model to offer an answer within a limited set of options.

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.

5.3.3 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

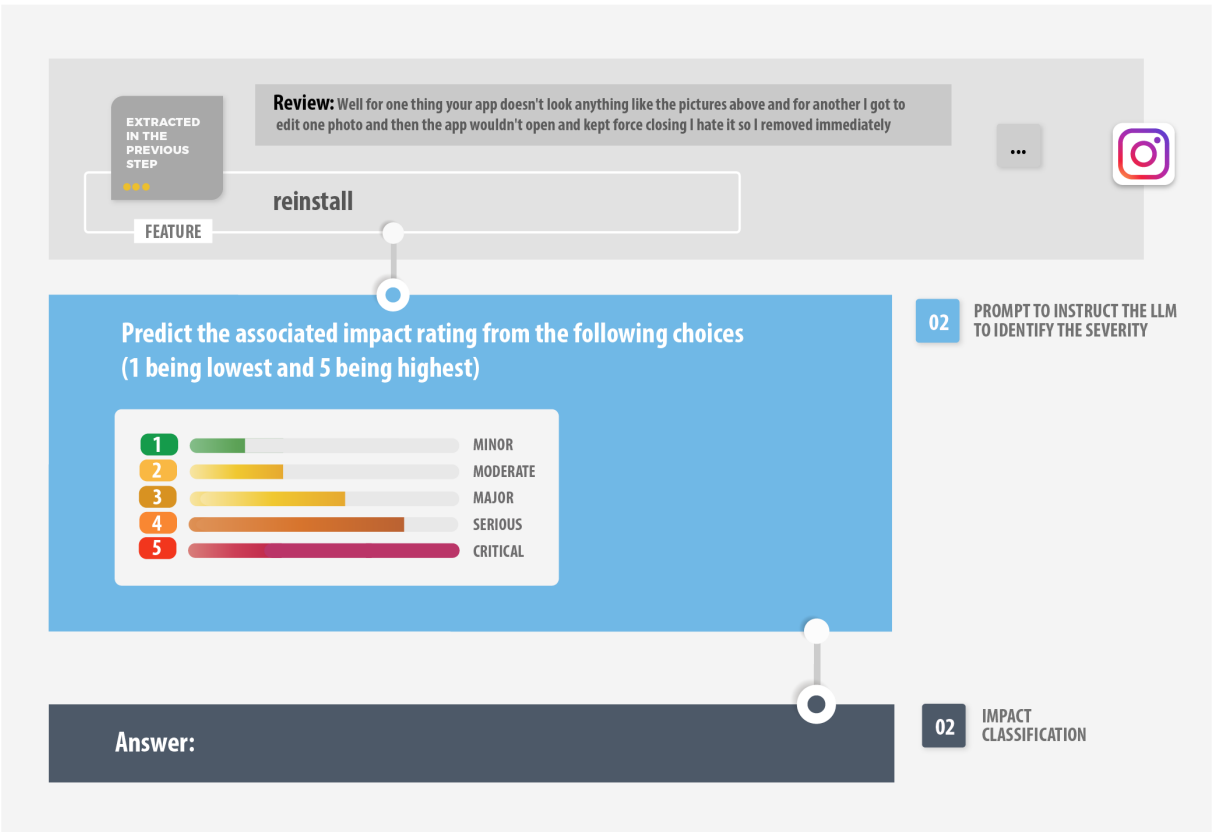

Figure 5.4: Prompt generated to obtain the severity classification for the Instagram app

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. Let $sim(e(t_i), e(t_j))$ be the function that computes the similarity between two expressions. If the similarity value exceeds a predefined threshold, an edge is created between the corresponding vertices. Based on these similarities, we can construct the graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, as illustrated in Figure 5.5.

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

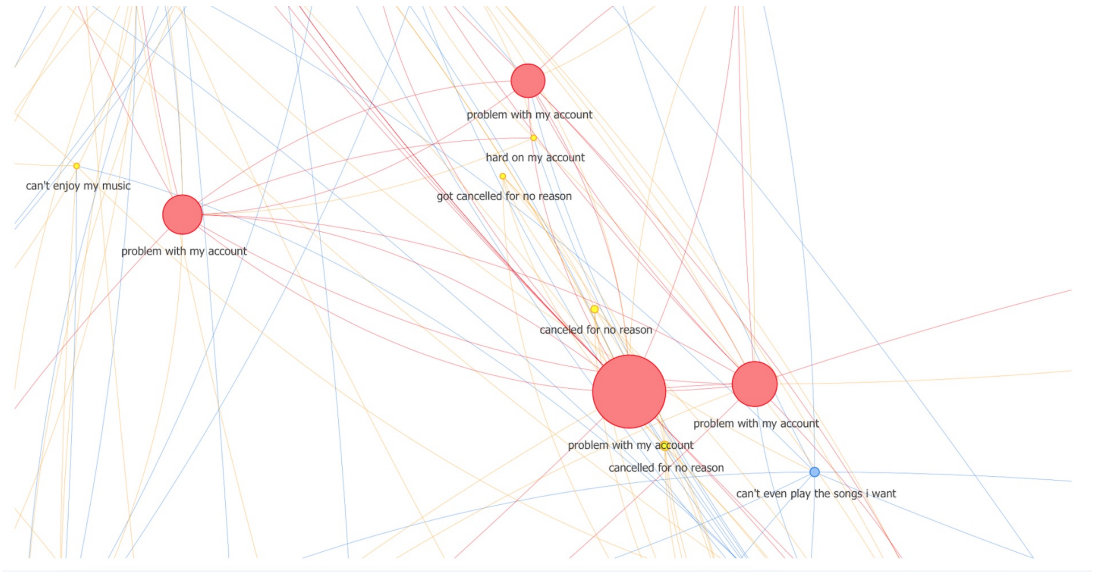

Figure 5.5: Edge is created between the corresponding vertices if the similarity value exceeds a predefined threshold

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.

Finally, the risk matrix is constructed considering the previous steps' impact and occurrence likelihood dimensions. The next section presents an experimental evaluation of the proposed approach.

5.4 Experiment Design

We conducted an experiment to evaluate the approach presented in this paper. To do so, we followed the guidelines proposed by Wohlin et al. (2012). The experimental design is detailed in the remainder of this section.

5.4.1 Definition of Research Questions

Our central research question is: *how do we learn risk factors from app reviews using Large Language Models and prioritize app reviews and anticipate to mitigate risks?*

To answer this main question, we divided it into two specific research questions, as follows:

- **RQ1**: How can we extract features using LLMs with limited labeled data?

- **RQ2**: How can we identify the severity or impact of risks mentioned in the reviews and automatically organize them into a risk matrix?

The experiment was conducted to address the research questions presented in this section.

5.4.2 Experiment Definition

The experiment is defined as follows (Wohlin et al., 2012):

- analyze risk matrix construction method using LLM,
- to evaluate feature extraction, impact estimation, and likelihood estimation,
- with respect to model performance,
- from the point of view of the researcher,
- in the context of crowd feedback from app user reviews.

5.4.3 Preparation and Planning

The experiment plan comprises the sample selection, description of the experimental package, definition of variables, and description of employed design principles.

To evaluate the performance of the risk matrix construction approach, we compare our strategy with other state-of-art approaches.

5.4.4 Sample Selection

To evaluate the risk matrix construction approach, we selected app review datasets used in previous studies of review mining (Dabrowski et al., 2020). In this context, we utilized human-labeled data consisting of reviews, app features, and corresponding sentiment to generate the impact of a risk matrix.

We used eight mobile apps from these datasets, as described in Table 5.1. We included apps from different categories to enhance the generalizability of our results. The ground truth consists of 1,000 reviews for the eight analyzed apps, with 1,255 distinct features, meaning they are mentioned only once, making extracting app features from reviews more challenging.

Table 5.1: The overview of the datasets used for automatic risk matrix construction.

App	Reviews	Labeled reviews	Sentences	Features	Distinct features
eBay	1,962	125	294	206	167
Evernote	4,832	125	367	295	259
Facebook	8,293	125	327	242	204
Netflix	14,310	125	341	262	201
Photo editor	7,690	125	154	96	80
Spotify	14,487	125	227	180	145
Twitter	63,628	125	183	122	99
WhatsApp	248,641	125	169	118	100

5.4.5 Experimental Package

In our experiment, we have three objects:

- feature extraction;
- impact estimation; and
- likelihood estimation.

The following components are part of the experiment package:

- Reference dataset description: Our experiment utilized human-labeled data containing reviews, app features, and sentiment. This dataset was obtained from Dabrowski et al. (2020).
- Object definition: We use Python programming to define each object.
- Machine learning classification methods: We compared our proposed OPT-based approach with three rule-based methods (GuMa, SAFE, ReUS), a fine-tuning method of language models (RE-BERT), and a large language model (GPT 3.5).

5.4.6 Variables

The independent variables (factors) controlled in the experiment were feature extraction and risk estimation, and their respective treatments are described below.

5.4.6.1 Feature Extraction Factor

The values assigned to this variable (treatments) are six classifiers (GuMa, SAFE, ReUS, RE-BERT, GPT, and our OPT-based Proposal). The dependent variable, which is affected by the treatment, is (i) the F1-Score, a measure of the accuracy of a test (RQ1), and (ii) MAPE / MAE employs typical measures from the regression field for prediction error (RQ2).

To evaluate the feature extraction step, we use the F1 measure for feature matching, as proposed by (Dabrowski et al., 2020), that corresponds to the harmonic mean of Precision (5.1) and Recall (5.2), where TP (True Positive) refers to the number of features that were both extracted and annotated; FP (False Positive) are features that were extracted but not annotated, and FN (False Negative) refers to the features annotated but not extracted. Equation 5.3 defines the F1 measure.

This measure allows us to assess the precision and recall of feature extraction in relation to annotated reference features. The parameter n of the Feature Matching allows for flexible matching, where $n = 0$ indicates exact matching, while $n > 0$ represents the difference between the sizes of the extracted and labeled sequences. We used $n = 2$ in the experimental evaluation.

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 = \frac{2 * P * R}{P + R} = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.3}$$

5.4.6.2 Risk Estimation Factor

The risk matrix is evaluated in the impact dimension by comparing the numerical level of the reference impact with the impact estimated by our method. For this evaluation, we employ typical measures from the regression field, such as the Mean Absolute Percentage Error (MAPE) and the Mean Absolute Error (MAE), as defined in Equation 5.4 and 5.5 respectively,

$$MAPE = \frac{1}{n} \sum_{t=1}^{n} \frac{|real_t - pred_t|}{real_t} \tag{5.4}$$

$$MAE = \frac{\sum_{t=1}^{n} |real_t - pred_t|}{n} \tag{5.5}$$

where $real_t$ is the real value and $pred_t$ is the predicted value by the method, and n is the sample size.

5.4.7 Experimental Design

In this experiment, the experimental design has two factors (feature extraction and risk estimation) with the following treatments (Wohlin et al., 2012):

- Feature extraction treatments: GuMa, SAFE, ReUS, RE-BERT, GPT, and OPT-based approach.
- Risk estimation treatments: GPT and OPT-based approach.

For the step of extracting features from reviews of applications, we compared the proposed OPT-based approach with three rule-based methods (GuMa, SAFE, ReUS), a fine-tuning method of language models (RE-BERT), and a large language model (GPT 3.5).

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.

5.4.8 Operation of the Experiment

Concerning the step of extracting features, GuMa performs feature extraction using a collocation search algorithm, which identifies commonly used expressions of two or more words that convey a specific meaning through co-occurrence-based measures. SAFE, relies on manually identified linguistic patterns, including patterns of parts of speech and sentences, to extract features from applications. ReUS utilizes linguistic rules composed of patterns of parts of speech and semantic dependency relations. These rules allow for simultaneous feature extraction and sentiment analysis. To determine sentiment, the method employs lexical dictionaries. In contrast, RE-BERT uses pre-trained language models to generate semantic textual representations, focusing on the local context of software requirement tokens. However, RE-BERT is a supervised learning method, i.e., it requires a labeled dataset for model training.

The proposed method employs the OPT-6.7b model, which contains 6.7 billion parameters, for the step of extracting features from application reviews. In this step, we employ the proposed strategy of dynamically generating prompts. We use a cross-domain approach, where reviews from other applications (source apps) are used as a knowledge base to generate specific prompts for each review of the target app. This cross-domain approach with dynamic prompt generation allows the model to be fed with information from related applications, expanding its ability to generalize without requiring prior knowledge about a specific target app. To perform this prompt generation, we

use the Sentence-BERT (Reimers and Gurevych, 2019) embedding model with $k = 10$ to compute the nearest neighbors based on cosine similarity.

For identifying the impact of the feature-review pair, our proposal utilizes the OPT-IML-1.3b model (Iyer et al., 2022). This model results from a fine-tuning process of large pre-trained language models on a collection of tasks described through instructions, also known as instruction-tuning. This process aims to improve the generalization capability of these models for previously unseen tasks. Our proposal employs the zero-shot learning strategy in this step. This means that the model is capable of learning to identify the impact of a feature-review pair, even without receiving specific examples of this relationship during training.

Finally, we also use GPT 3.5 as the reference model, which is another pre-trained language model, but in the category of Large Language Models. It is used for both the extraction of features from reviews and the identification of review impact. We employ a zero-shot learning strategy by providing instructions and only examples of how the extraction output should be formatted.

5.4.9 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. Table 5.2 presentes an overview of the experimental results in the task of extracting features from application reviews.

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.

Table 5.2: Comparison of approaches GuMa, SAFE, ReUS, RE-BERT, GPT (zero-shot learning) and OPT (Proposal with few-shot learning) for feature extraction from app reviews.

APP	F1 Matching Score (n = 2)					
	GuMa	**SAFE**	**ReUS**	**RE-BERT**	**GPT**	**Proposal**
eBay	0.22	0.36	0.21	0.53	0.22	0.39
Evernote	0.24	0.33	0.28	0.63	0.14	0.46
Facebook	0.19	0.24	0.19	0.61	0.20	0.40
Netflix	0.21	0.28	0.27	0.62	0.23	0.41
PhotoEditor	0.28	0.34	0.26	0.81	0.32	0.56
Spotify	0.28	0.35	0.27	0.60	0.17	0.48
Twitter	0.27	0.35	0.26	0.67	0.25	0.47
WhatsApp	0.26	0.39	0.24	0.61	0.18	0.47

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. As illustrated in Table 5.3, OPT-IML exhibits a lower error in constructing the risk matrix in the impact dimension, highlighting it as a promising alternative compared to the proprietary GPT model.

Figure 5.6 illustrates a risk matrix constructed automatically with the proposed approach for Netflix app. Note that reviews and app features categorized as critical impact and with a high likelihood of occurrence are frequent complaints with a strong negative sentiment. This occurs because the model identifies that such complaints have a greater severity on the application's reputation, meaning they directly affect the app's ratings in the store.

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.

Table 5.3: Error comparison in the Risk Matrix construction.

APP	GPT-3.5		OPT-IML	
	MAE	MAPE	MAE	MAPE
eBay	1.000	0.517	0.913	0.335
Evernote	1.053	0.532	1.160	0.388
Facebook	1.092	0.331	0.965	0.285
Netflix	1.255	0.446	1.061	0.359
PhotoEditor	0.957	0.443	0.929	0.301
Spotify	1.034	0.344	0.840	0.244
Twitter	0.943	0.267	0.971	0.253
Whatsapp	1.133	0.378	1.120	0.357

5.4.10 Findings to Research Questions

Returning to the initial primary research question: *"How do we learn risk factors from app reviews using Large Language Models and prioritize app reviews and anticipate to mitigate risks?"*. We addressed it by formulating two specific questions (see Section 5.4.1) and answering them based on the results obtained from the conducted experiment.

We present objective answers to each research question below (RQ1 and RQ2), highlighting the main findings:

- **(RQ1) Feature extraction with limited labeled data.** We introduced the analysis and classification of app reviews using Large Language Models (LLMs) with limited computational resources and data privacy constraints. We incorporated a dynamic and automatic prompt generation technique to extract specific app characteristics mentioned in the reviews. The experimental results showed that our OPT-based proposed approach is superior to rule-based and has advantages over supervised methods that require a significant amount of labeled data. The findings demonstrated that our proposal is competitive compared to large proprietary language models like GPT-3.5. Despite potential areas for improvement, the findings suggest significant progress in extracting features using LLMs with limited labeled data.

- **(RQ2) Automated risk matrix construction.** We developed and evaluated instructions to classify the severity or impact of risks mentioned in app reviews. These instructions served as prompts to guide our Large Language Model (LLM) in categorizing the risks into five levels: negligible, minor, moderate, major, and critical. The experimental results indicated

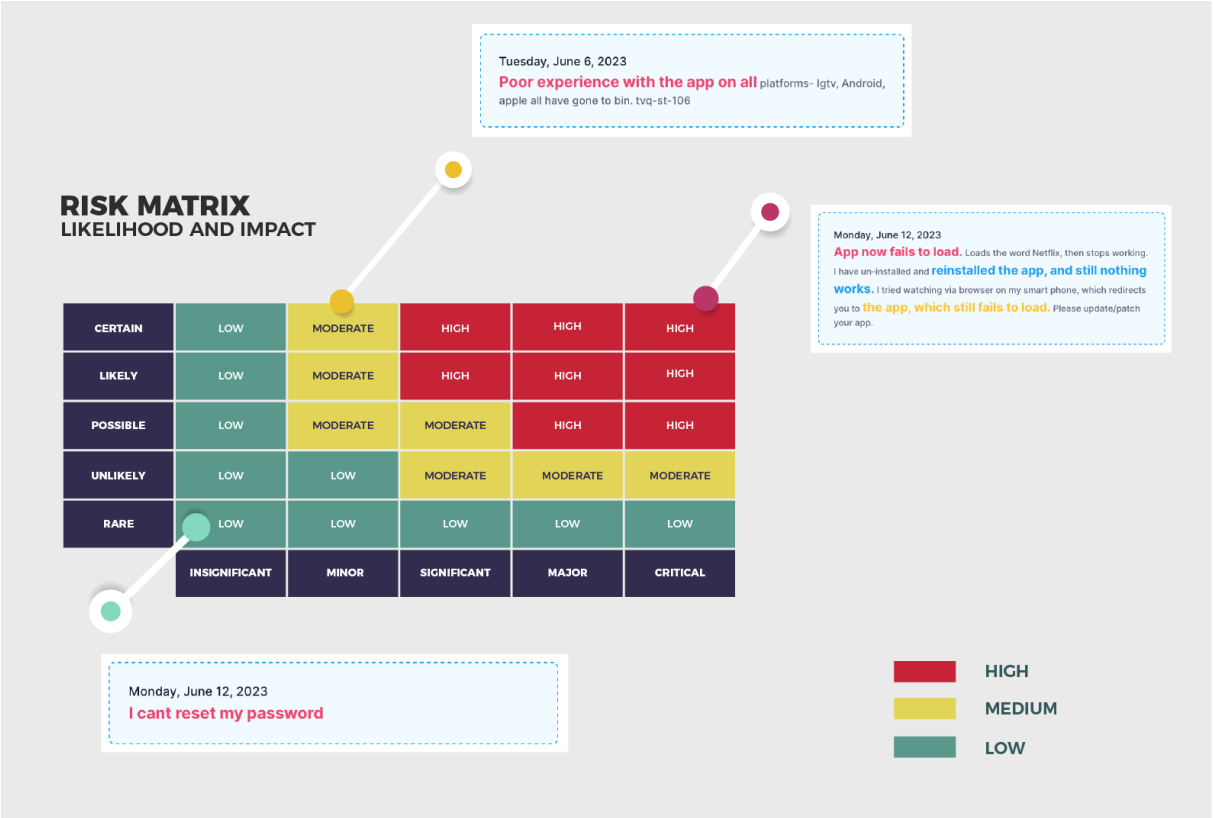

Figure 5.6: Risk matrix constructed automatically with the proposed approach for Netflix app

that our proposed approach, using LLMs, effectively identified the severity or impact of risks mentioned in the reviews and organized them into a risk matrix. Our research reveals a significant advancement in the field of software product maintenance and evolution using LLMs.

5.5 Threats to Validity

5.5.1 Internal Validity

Potential threats to internal validity include the quality and representativeness of the knowledge base of app reviews used for prompt construction. If the knowledge base is biased or lacks diversity, it may impact the effectiveness of the method. Additionally, the classification of risk severity based on the app rating may introduce subjectivity and potential errors. To mitigate these threats, we utilized review datasets from apps in different domains. We also employed a cross-domain validation strategy, where we used datasets to generate prompts from reviews of apps that are different from the target app being analyzed.

5.5.2 Construct Validity

Construct validity could be threatened if the classification of risk severity based on the app rating does not truly reflect the impact of the risks. In this study, our assumption is that one of the main risk factors for the app is actions that impact its reputation (average overall rating) in the app store and, therefore, increase the chances of being uninstalled or not even installed by users. To enhance construct validity, further research should focus on evaluating other types of risks associated with apps, such as security and malfunctions that affect users' smartphones.

5.5.3 External Validity

The external validity may be limited by the specific LLM used (OPT model) and the dataset of app reviews. Different LLM architectures or datasets from other domains may yield different results. The effectiveness of the approach may also vary depending on the characteristics of the target app. To improve external validity, conducting replication studies using different LLMs and diverse datasets from various app domains would be valuable. User studies or obtaining feedback from software engineering professionals would also help validate the practical usefulness and effectiveness of the risk matrices generated from app reviews.

5.6 Final Remarks

We introduced the analysis and classification of app reviews through a risk matrix using Large Language Models (LLMs), specifically open-access methods suitable for scenarios with limited computational resources and data privacy constraints, in contrast to proprietary models like GPT.

Our proposed approach incorporates a dynamic and automatic prompt generation technique, enabling the extraction of specific application characteristics mentioned by users in the reviews. We also developed and evaluated instructions to classify the severity or impact of the risks mentioned in the reviews. These instructions serve as prompts to guide the LLM in classifying the risks into five levels: negligible, minor, moderate, major, and critical. This standardizes the risk assessment and facilitates the automatic construction of the risk matrix. The experimental results provide evidence that the proposal, through Open Pre-trained Transformers (OPT), is competitive compared to large proprietary language models such as GPT-3.5. Although there is room for improvement regarding the risk matrices generated by supervised reference models, our results indicate a significant advancement in software

product maintenance and evolution.

Direction for future work involves the development of tools to support decision-making, visualization, and monitoring associated with the risk matrices. These tools would provide actionable insights and facilitate the interpretation of the risk assessment results. Additionally, exploring techniques to enhance the accuracy and granularity of the risk classification levels could further improve the effectiveness of the risk matrix approach. Furthermore, investigating the integration of external data sources, such as social media and user forums, could provide additional context and insights into app-related risks. The development of part of these tools is addressed and better discussed in the next chapter.

CHAPTER

6

Design and Architecture of the Analytical Data Exploration Tool

6.1 Introduction

This chapter provides a comprehensive technical and documentary overview of the architecture and implementation of our analytical tool designed for data exploration, called the MApp-IDEA tool.

The rest of this chapter is structured as follows. In section 6.2, we will discuss various system design and implementation aspects. We will start by presenting the technologies schema 6.3, which provides an overview of the key technologies components separated in a stacked layer schema. Next, we will delve into patterns 6.4, specifically architectural patterns and design patterns. We will explore the characteristics of these patterns and how they are used in the project. Section 6.5 explores the system's graphical interface, discussing user interface design principles and frameworks employed. Additionally, we will explore the RESTful Bus (6.6) architecture for communication, explaining its concept, advantages, and implementation details in the system. Finally, we will provide some final remarks (6.7).

RESTful architecture for communication

6.2 Component-based Development

Software development is a complex process that can benefit significantly from adopting a component-based development approach. Component-based

development focuses on building software systems by assembling reusable and self-contained components, each responsible for a specific functionality or service Szyperski (2002).

The MApp-IDEA tool is based on components aimed at reusability and interoperability between systems and services. The components was designed to be independent and self-contained, making them highly reusable in different contexts and projects Clements et al. (2002). By reusing existing components, we can save time and effort, leading to increased productivity and improved software quality.

We highlight important keys about component-based development adopted by the MApp-IDEA tool:

- **Modularity**. Components are developed, tested, and maintained independently, allowing for better code organization and easier maintenance Budgen (2003). Changes made to one component are less likely to have an impact on other parts of the system, making it easier to update and evolve the software.

- **Scalability**. Components are designed to be loosely coupled and independent, so it becomes easier to scale specific parts of a software system without affecting the entire application Szyperski (2002). This enables efficient resource utilization and ensures optimal performance even under high loads.

- **Collaboration**. By providing well-defined interfaces and contracts, components enable developers to work together effectively Clements et al. (2002). Teams can focus on developing specific components (e.g., APIs) that can be seamlessly integrated into the larger system. This encourages specialization and facilitates code reuse and knowledge sharing within the development team.

- **Maintainability**. The modularity of components makes updating or replacing individual components easier without impacting the entire system Budgen (2003). This allows for efficient bug fixes, enhancements, and system evolution, making software maintenance less challenging and costly.

The component-based development approach adopted in the MApp-IDEA tool offers numerous advantages. It promotes reusability, modularity, scalability, collaboration, maintainability, and extensibility. By leveraging pre-built components and assembling them into a cohesive system, we can streamline development and improve productivity.

6.3 Technologies Schema

The development of the MApp-IDEA tool involved utilizing several key technologies, e.g., frameworks,libraries, NL models and APIs. Each of these technologies played a crucial role in different aspects of the tool, contributing to its functionality and user experience.

To facilitate comprehension of the technology interaction, we have organized it into a stacked layer scheme, where the technologies are grouped based on different scope levels. This division allows for a clearer understanding of the relationships and interactions between the various components involved, as fllowing:

- **Data collect**. Involves the process of collecting data, e.g., reviews and app metrics.
- **Opinion Mining**. Involves analyzing text to identify and classify issues from opinions expressed by app users.
- **Persistence**. Refers to the storage and retrieval of data in a durable and reliable manner.
- **Application**. Refers to the practical implementation and utilization of a system or software. It involves software components, integrating different technologies and user interfaces to provide functionalities and solutions to end users.
- **Synchronization**. Involves data replication, conflict resolution, and communication protocols to achieve consistency and maintain coherence between entities and data.

Figure 6.1 shows an overview of the technologies separated by the stacked layer scheme. The following is a summary list of technologies. Additional information can be found in the project's public repository on Github [1].

- **Programming languages:** PHP, Python, Javascript, CSS (style), and HTML (markup);
- **Frameworks:** Laminas, Bootstrap;
- **NL Model:** DestilBERT;
- **Database:** Postgres; and
- **Libraries:** Faiss, Nltk, NetworkX, Sklearn, VADER, Transformers, Pandas, Numpy, SciPy, ONNX Runtime, Vijs, Am5Chart, and others.

[1] https://github.com/vitormesaque/mapp-idea

These technologies stack facilitated seamless server-side and client-side functionality integration, enabling an efficient and user-friendly tool for data analysis.

Several of these technologies intermingle and depend on each other to handle specific functionalities, e.g., crawling, issue detection, and synchronization. For example, PHP code made background calls to Python code, which utilized its rich ecosystem of libraries and tools for these tasks.

Through the RESTFul bus, with log systems, notifications, and listeners, it is possible to integrate all technologies transparently for the user. For instance, after completing each step, the Python module responsible for issue detection sends a message to the communication bus (Figure 6.4). Subsequently, the PHP modules can access the bus as a listener to check for new communication messages.

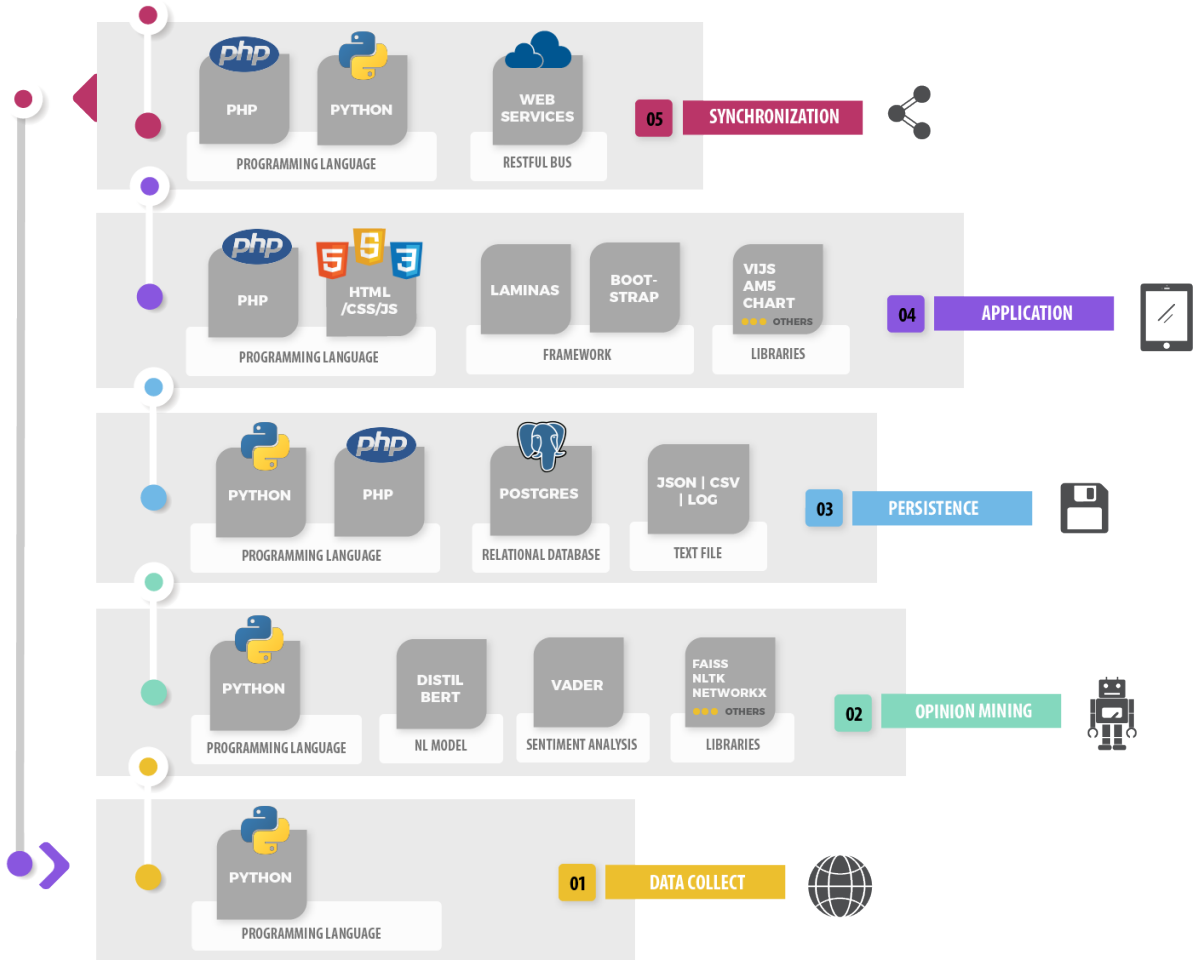

Figure 6.1: Scheme of technologies used separated by stacked layers

6.4 Patterns

Design patterns and architectural patterns are both important concepts in software engineering, but they differ in their scope and level of abstraction. The following sections will discuss the patterns incorporated into the MApp-IDEA tool.

6.4.1 Architectural Patterns

Architectural patterns deal with higher-level structures and the organization of entire software systems. They provide a framework for designing the overall structure and interaction between major components and subsystems. Architectural patterns define the fundamental principles and guidelines for system organization, distribution of responsibilities, and communication between different parts of the system (Sommerville, 2013).

6.4.1.1 Model-View-Controller Pattern

MApp-IDEA tool uses the MVC (Model-View-Controller) architectural pattern widely used in software development. It provides a structured approach for designing and organizing applications by separating the concerns of data, presentation, and user interaction Gamma et al. (1995).

In the MVC pattern, the application is divided into three main components Gamma et al. (1995); Reenskaug (1979):

- **Model** represents the application's data and business logic. It encapsulates data access, data manipulation, and business rules. The Model component is responsible for managing the state and behavior of the application.
- **View** is responsible for the application's presentation layer. It displays the data from the Model to the user and handles the user interface elements. The View is focused on providing a visually appealing and intuitive user interface.
- **Controller** acts as an intermediary between the Model and the View. It receives input from the user or external systems, performs the necessary actions, and updates the Model and View accordingly. The Controller handles user interactions, interprets requests, and coordinates data flow between the Model and the View.

The MVC pattern promotes modularity, maintainability, and code reusability by separating data, presentation, and user interaction concerns. It allows

developers to change one component without affecting the others, facilitating code organization and collaboration in larger projects Gamma et al. (1995).

Overall, the MVC pattern provides a clear separation of responsibilities, making developing, testing, and maintaining software applications easier. It is widely used in web development frameworks, desktop applications, and mobile app development.

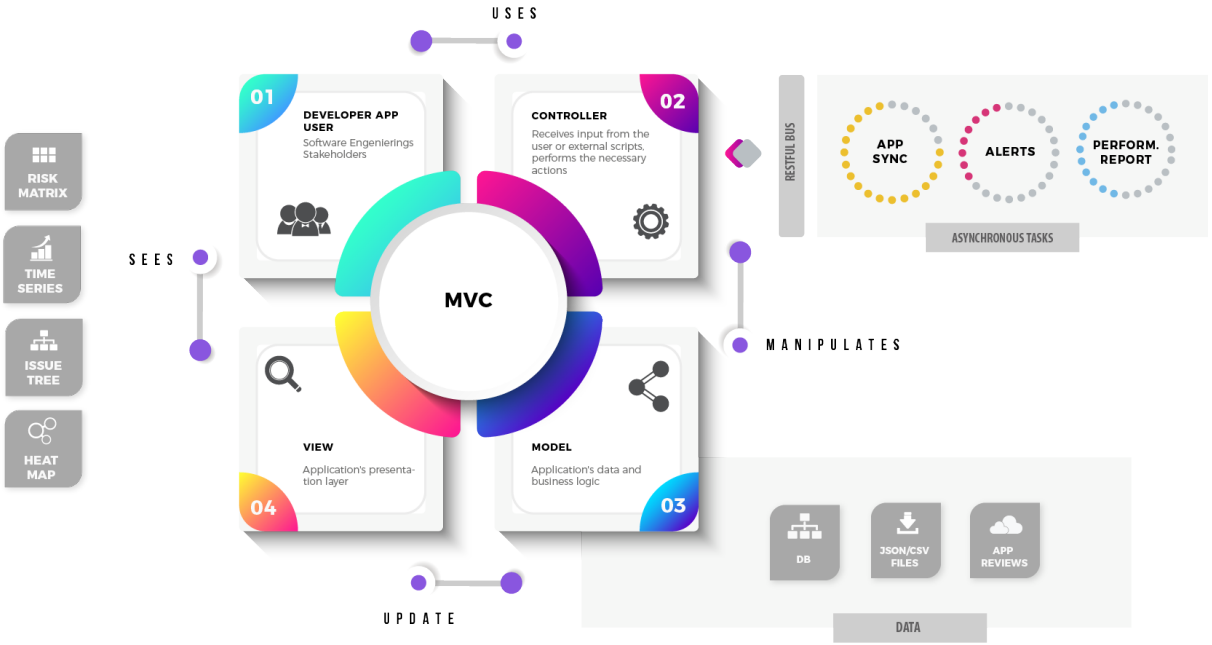

Figure 6.2: MVC architecture of the MApp-IDEA tool

6.4.1.2 MVC Framework

In our project we use the Laminas framework [2] to support the implementation of the MVC architectural pattern. The Laminas is a PHP framework for building robust web applications.

One of the key advantages of the Laminas is its modular architecture. The framework is designed as a collection of independent and reusable modules, allowing developers to selectively use and integrate the necessary components for their specific application requirements. This modular approach promotes code organization, reduces development time, and facilitates code reuse and integration across multiple projects.

In addition to its modular architecture, the Laminas offers seamless integration with various databases. It provides convenient abstractions and utilities for working with different database systems, making it easier for developers to handle database operations such as querying, data manipulation, and transaction management. This integration simplifies the development process and ensures compatibility with different database backends.

[2] https://getlaminas.org

Furthermore, the Laminas Framework emphasizes the importance of community-driven development. It has an active and supportive community that provides regular updates, extensive documentation, and a wide range of resources. This community-driven approach fosters collaboration, knowledge sharing, and continuous improvement, making it easier for developers to learn and leverage the capabilities of the Laminas Framework.

6.4.2 Design Patterns

As described by Gamma et al. (1995), design patterns focus on solving specific design problems at the class or object level. They provide reusable solutions to recurring design challenges and encapsulate best practices for designing individual components or interactions within a system. Design patterns address object creation, structuring relationships, and behavior delegation.

Several popular design patterns have been widely adopted and studied in the field. We highlight the characteristics and applications of the main patterns (Factory, Singleton, Observer, Adapter, Strategy, and Table Data Gateway) (Gamma et al., 1995; Fowler, 2002) used in the MApp-IDEA tool.

- **Factory**. Focuses on creating objects without specifying their concrete classes. It encapsulates object creation logic within a separate class or method, providing a centralized and flexible approach to object instantiation (Gamma et al., 1995). The Factory pattern promotes easy modification and extension of the object creation process.

- **Hydrator**. Focuses on separating the process of object creation from the process of populating that object with data. It acts as a bridge between data sources (e.g., databases or APIs), and object-oriented programming models.

- **Singleton**. Restricts the instantiation of a class to a single object. It ensures that only one instance of a class exists throughout the application, providing global access to that instance. The Singleton pattern is useful in scenarios where a single shared resource or state must be accessed from multiple system parts. It simplifies coordination and avoids unnecessary resource duplication .

- **Observer**. Establishes a one-to-many dependency between objects, where the dependent objects (observers) are notified and updated automatically when the state of the subject object changes (Gamma et al., 1995). This pattern decouples the subjects and observers, enabling them to evolve independently and reducing their interdependencies.

- **Adapter**. Allows incompatible interfaces of different classes to work together. It acts as a bridge between two incompatible interfaces, converting the interface of one class into another that clients expect (Gamma et al., 1995). The Adapter pattern enables the integration of legacy code or third-party libraries into new systems without modifying their existing interfaces, promoting component interoperability.

- **Strategy**. Focuses on encapsulating interchangeable algorithms or behaviors within a family of classes. It allows clients to select different strategies dynamically, depending on their requirements, without modifying the client code (Gamma et al., 1995). The Strategy pattern promotes flexible design and enables easy extension and modification of the system's behavior.

- **Table Data Gateway**. Provides a centralized gateway for accessing a database or data source. It encapsulates the data access logic within a single class, which handles all interactions with the underlying data store. The Table Data Gateway pattern abstracts the database operations and provides a convenient interface for clients to query and manipulate data (Fowler, 2002).

6.5 Graphical Interface

To build a robust and standardized graphical interface, we use a well-known framework called Bootstrap [3] to build. It provides pre-designed and responsive components, saving time and effort. Bootstrap ensures visual consistency and a professional standard. It has extensive documentation and a supportive community. The framework is mobile-first, enhancing responsiveness and usability.

The MApp-IDEA aims to provide an efficient and user-friendly experience for users accessing it through different devices, including smartphones, tablets, and desktop computers. By implementing a responsive design, the MApp-IDEA tool ensures that users can easily navigate and interact with the tool's features, regardless of the screen size or device they use. This adaptability enhances usability and accessibility, allowing users to access the tool's functionality on their preferred devices conveniently. Additionally, a responsive graphical interface reduces development and maintenance costs by eliminating the need to create separate versions for different devices.

We developed dark mode navigation to improve the navigation experience, as Figure 6.3 illustrates. The availability of a dark mode option in apps en-

[3] https://getbootstrap.com

hances visual comfort, saves battery life, and improves accessibility. Dark mode provides a high-contrast interface that can benefit users with visual impairments or sensitivities. The darker background and lighter text make it easier for individuals with low vision or color blindness to read and navigate the app's content.

Figure 6.3: Dark mode home screen

6.6 RESTful Bus

The RESTful bus refers to a messaging infrastructure that follows the principles of REST architectural style. The RESTful bus allows communication and coordination between different software components or services using HTTP as the underlying protocol. It relies on the principles of REST, such as the use of uniform resource identifiers (URIs) to identify resources, stateless communication, and standard HTTP methods (e.g., GET and POST) for performing operations on these resources.

For better understanding, we define communication messages into two distinct levels: i) RESTFull bus messages, and ii) alert trigger messages, as following:

- **RESTFull bus messages**. Refer to the general communication exchanged between different components or nodes within the bus service.
  - **Notification**. Notifications are messages that provide general information about certain events, system states, or updates.

- **Alert system messages**. Messages sent about monitoring issues and risks associated with trends detected by the tool.
  - **Notice**. Notices are informational messages that communicate general information without requiring immediate attention or action.
  - **Alert**. Alerts are specific messages that are used to indicate urgent or critical situations that require immediate attention.

6.6.1 Synchronization

The MApp-IDEA tool offers a remarkable capability of real-time issue analysis and prioritization. It is essential to download reviews and detect issues in the background to enable this analysis, considering the substantial amount of data being processed and the associated computational demands. Given that many operations are conducted in the background, a communication bus becomes crucial for facilitating the exchange of status messages among the involved components. Also, background services are constantly running with synchronization tasks and alerts.

Once an app is registered in the tool, it is periodically synchronized. The sync period can be customized, but by default it is five minutes. In the scheduled time interval for synchronization, the process is called in the background, making a series of checks to find the last updated review and processing the newly collected reviews. To optimize the review download and processing time, the system estimates how many reviews will be downloaded according to the time of the last update and the average of reviews the app receives per day.

The system performs the following steps in the background on synchronization illustrated in Figure 6.4.

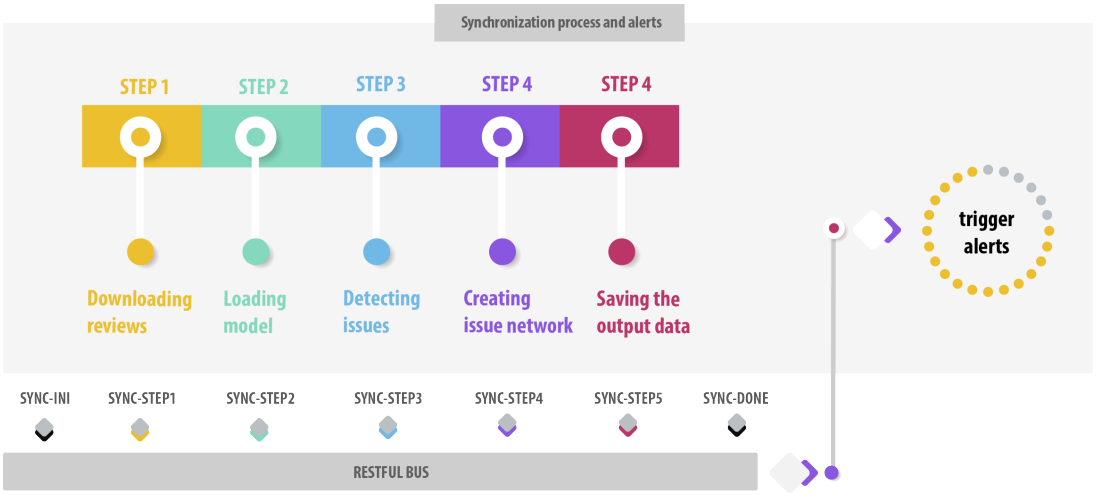

Figure 6.4: Synchronization process steps and triggering alerts

Depending on the volume of reviews processed, the synchronization may take a few minutes, but the user can follow the processing progress.

During the synchronization process, whenever there is a status update, a message is sent to the RESTful bus to notify about the new status. This enables the view layer to query the bus and retrieve the updated status information.

6.6.2 Alert Trigger

Once the synchronization process is completed, the system notifies the RESTful bus about the synchronization status. Subsequently, the RESTful bus forwards this information to the alert trigger system, which initiates the scanning process to identify and trigger alerts and notices related to app issues and risks. Similar to the synchronization process, the alert trigger system operates in real-time.

By utilizing the RESTful bus, we ensure that potential app issues and risks are promptly detected, and appropriate actions are taken through efficient communication and real-time availability of status updates, enhancing the overall responsiveness and functionality of the application.

As an illustration, Figure 6.5 displays an Alert message dated May 26, generated by the Netflix app to inform users about an app update that revised the policy regarding simultaneous screens in the family plan. Figure 6.6 exhibits a notice message from the Facebook application on May 25, 2023.

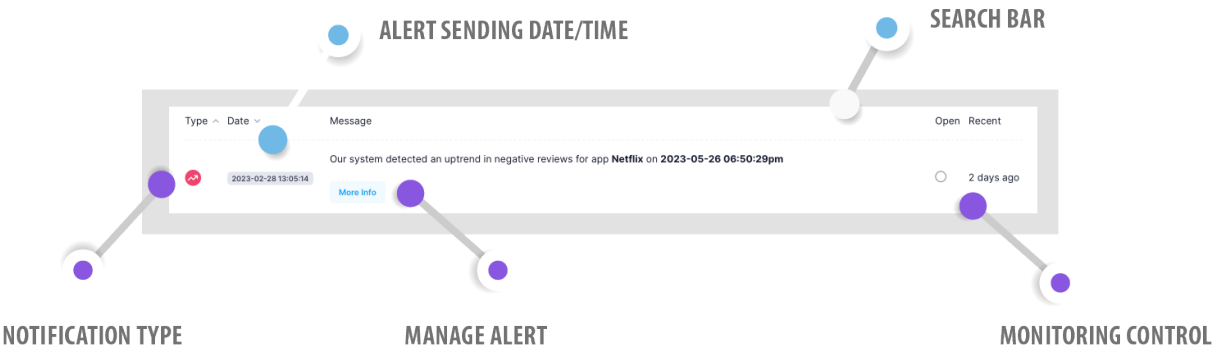

Figure 6.5: Alert message from the Netflix app on May 26, when there was an app update changing the policy for simultaneous screens in the family plan.

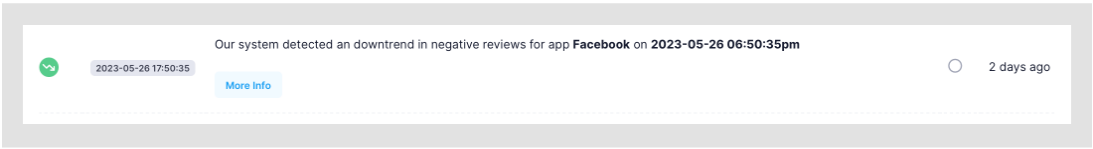

Figure 6.6: Facebook application notice message on May 25, 2023

The MApp-IDEA tool includes a dedicated component for message management, ensuring that all sent messages are stored, as shown in Figure 6.7. When a message is opened, the system marks it as read to provide users with

a clear indication of their message status. Even if a user chooses to delete a message (whether it's an alert or a notice), the system retains a record of it in the database for traceability purposes. Therefore, while the message may appear unavailable to the user, it is not permanently deleted from the system. This approach ensures data integrity and allows tracking and auditing of message-related activities.

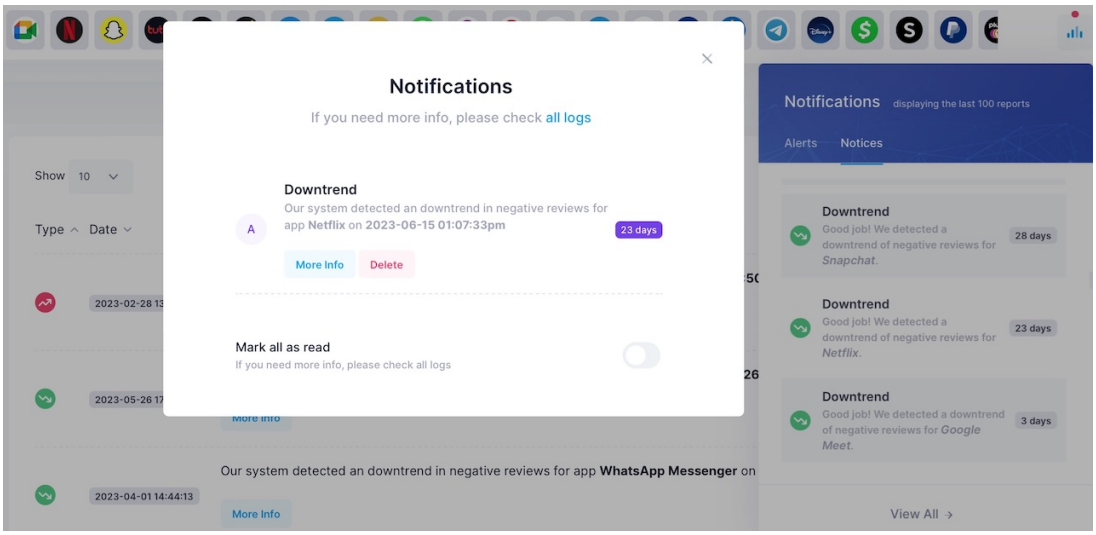

Figure 6.7: Component for managing triggered messages related to the detection and prioritization of issues.

6.7 Final Remarks

In this chapter, we discussed the design and architecture of the MApp-IDEA tool, focusing on key aspects such as component-based development, technologies schema, patterns, and the graphical interface.

The design and architecture of the MApp-IDEA tool were carefully crafted to deliver a powerful and user-friendly data analysis solution. Reusable components were leveraged through component-based development, enabling modularity, scalability, collaboration, and maintainability. Various technologies, such as frameworks, models, and libraries, were seamlessly integrated to create a cohesive system. Architectural patterns (e.g., MVC), and design patterns (e.g., Factory, Singleton, Adapter, Strategy, and Table Data Gateway), enhanced code structure, flexibility, and extensibility. The graphical interface, built with Bootstrap, ensured a responsive and visually appealing user experience. The synchronization process and real-time alerts optimized the processing of app reviews. In summary, the MApp-IDEA tool embodies industry best practices, resulting in a powerful and user-friendly solution for analytical data exploration.

CHAPTER

7

Conclusions

This chapter presents the conclusions of this thesis. In Section 7.1, the final remarks, the main findings, and the research implications are summarized. In Section 7.2, we answered the research questions raised in the introductory chapter (1.2) of this thesis. Section 7.3 highlights the original contributions made by the thesis to the field of study. The Publications section (7.4) provides a list of relevant publications resulting from the research. Section 7.5 specifies the availability of the data and codes used in the research, ensuring transparency and reproducibility. Finally, in Section 7.6, Limitations and Future works, the limitations of the current research are discussed, and potential directions for future studies are suggested.

7.1 Final Remarks

Opinion mining for app reviews can provide helpful user feedback to support software engineering activities. The automated issue detection, analysis, prioritization, and monitoring process from app reviews combines machine learning and software engineering techniques to offer a new software maintenance and evolution field.

This doctoral project 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.

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.

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.

7.2 Answer to Research Questions

In this thesis, we investigate two primary research questions and present objective answers to each.

- **(RQ1) Predict negative app rating trends.** We show that the prediction of initial trends on defective requirements from users' opinions can be accomplished by a method capable of to incorporates contextual word embeddings for software requirements representation and clustering, enabling to quantify accurately negative user mentions over time and identify potential issues and trends related to defective requirements using predictive models, before they negatively impact the overall evaluation of the app. This enables developers to proactively address these issues and make improvements to ensure user satisfaction. Chapter 3 addressed this issue by introducing the MApp-Reviews method.

- **(RQ2) Prioritize and address app issues.** We show that the prioritization and timely resolution of app issues from reviews can be achieved by exploring word embeddings to build acyclic graphs representing app issues, combining sentiment analysis and graph techniques to generate a risk matrix for identifying and prioritizing app issues mentioned in user reviews. This allows developers to allocate resources efficiently and promptly address critical issues, ensuring the app's competitiveness and facilitating timely maintenance and evolution of the software. Chapters 4 and 5 addressed this issue by introducing the MApp-IDEA method and an LLM approach for risk matrix construction.

7.3 Thesis Contributions

The main contributions of this doctoral project are as follows:

- Presents a two-fold approach, MApp-Reviews and MApp-IDEA, to explore emerging issues from user feedback to proactively detect, predict, prioritize, and monitor risks and issues over time.
- MApp-Reviews method identifies possible defective software requirements and trains predictive models for anticipating requirements with a higher probability of negative evaluation.
- MApp-IDEA method detects issues in reviews, classifies them in a risk matrix with prioritization levels, and monitors their evolution over time.
- Proposes an unsupervised approach that leverages user reviews to detect and prioritize app issues. This approach is competitive compared to supervised methods, highlighting its effectiveness in identifying and managing app issues.
- Presents a dynamic risk matrix that allows prioritizing issues based on their criticality levels. This feature enables software engineers to focus their attention and resources on the most impactful and urgent issues, leading to more efficient issue resolution.
- Introduces a risk matrix construction approach using Large Language Models (LLMs), exploring open-access methods suitable for scenarios with limited computational resources and data privacy constraints, in contrast to proprietary models like GPT.
- Introduces a comprehensive platform that facilitates app issue management. The platform includes features such as an automated dashboard and various visualizations (e.g., risk matrices, issue heat maps, issue trees, and issue time series) to enable developers to monitor and address issues efficiently.
- Bridgings the gap between issue incidence and identification aims to reduce the time lag between the occurrence of app issues and their detection by introducing automated notifications and alerts. This feature helps developers stay informed about emerging issues and take timely actions to mitigate their impact.

These contributions have the potential to enhance app development processes and improve user satisfaction by enabling software professionals to manage and resolve app issues effectively.

The findings showed that this doctoral project contributes an efficient solution to the research question of prioritizing and addressing user reviews in time so that the app is competitive and guarantees the timely maintenance and evolution of the software.

7.4 Publications and Other Intellectual Contributions

Throughout the development of this doctoral project, contributions from the point of view of research and technological innovation were disseminated through journals, conferences, technical reports and software. The results directly related to this doctoral thesis are presented below.

- **Journals**
  - **Paper 1 (Chapter 3)** Lima, V. M. A., Araújo, A. F., and Marcacini, R. M. (2022). Temporal dynamics of requirements engineering from mobile app reviews. PeerJ Computer Science, 8, e874.
  - **Paper 2 (Chapter 4)** Lima, V. M. A., Barbosa, J. R., and Marcacini, R. M. (2023). Issue detection and prioritization based on app reviews. 10.21203/rs.3.rs-2838568/v1.
  - **Paper 3 (Chapter 5)** Lima, V. M. A., Barbosa, J. R., Brandao, L. S., and Marcacini, R. M. (2023). Learning Risk Factors from App Reviews: A Large Language Model Approach for Risk Matrix Construction. 10.21203/rs.3.rs-3182322/v1
- **Conference**
  - **Paper 4 (Chapters 4 and 6)** Lima, V. M. A., Barbosa, J. R., and Marcacini, R. M. (2023). MApp-IDEA: Monitoring App for Issue Detection and Prioritization. 37th Brazilian Symposium on Software Engineering (submitted).
- **Technical Reports**
  - **Paper 5 (Chapter 6)** Lima, V. M. A., Barbosa, J. R., and Marcacini, R. M. (2023). MApp-IDEA: Design and Architecture of the Analytical Data Exploration Tool. Technical Reports of the Institute of Mathematics and Computer Sciences, University of São Paulo.
- **Software**
  - **Software Registration (Chapters 4 and 6)** Lima, V. M. A., Barbosa, J. R., and Marcacini, R. M. (2023). Analytical Data Exploration Tool for Opinion Mining from App Reviews. Software Registration. National Institute of Industrial Property.

7.5 Data and Codes Availability Statement

Data and source code files that support the findings and contributions of this doctoral thesis are available at:

- https://github.com/vitormesaque/mapp-reviews
- https://github.com/vitormesaque/mapp-idea

7.6 Limitations and Future Works

Future work directions encompass evaluating MAPP-Reviews and MApp-IDEA in various scenarios to incorporate and compare different types of domain knowledge into the predictive models. This includes considering factors like new app releases, marketing campaigns, server failures, competing apps, and other relevant information that may impact app evaluations. Furthermore, developing an accessible API for the models would enable third parties to utilize and benefit from them easily.

Moreover, additional future work involves evaluating MApp-IDEA in diverse software company settings to incorporate various types of domain knowledge and establishing an experimental design to assess the method by software engineers. Another direction is to incorporate predictive models into the temporal dynamics of the risk matrix, building upon the proven efficiency demonstrated in Chapter 3. A promissory research direction is an improvement of the risk matrix construction of MApp-IDEA (Chapter 4) to incorporate the proposed LLM-based approach presented in Chapter 5.

To gain deeper insights, we intend to explore our method further to determine the input variables that contribute most to the output behavior and the non-influential inputs or to determine some interaction effects within the model. In addition, sensitivity analysis can help us reduce the uncertainties found more effectively and calibrate the model.

Bibliography

Aggarwal, C. C. e Zhai, C. (2012). A survey of text clustering algorithms. In *Mining text data*, páginas 77–128. Springer. Cited on pages 17 e 18.

Ahmed, N. K., Atiya, A. F., Gayar, N. E., e El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621. Cited on page 2 20.

Akhtar, M. S., Kumar, A., Ekbal, A., e Bhattacharyya, P. (2016). A hybrid deep learning architecture for sentiment analysis. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, páginas 482–493. Cited on page 2 17.

Al Kilani, N., Tailakh, R., e Hanani, A. (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)*, páginas 541–548. Cited on pages 3, 27, e 30.

Al-Subaihin, A. A., Sarro, F., Black, S., Capra, L., Harman, M., Jia, Y., e Zhang, Y. (2016). Clustering mobile apps based on mined textual features. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, páginas 1–10. Cited on page 2 24.

AlSubaihin, A., Sarro, F., Black, S., Capra, L., e Harman, M. (2019). App store effects on software engineering practices. *IEEE Transactions on Software Engineering*. Cited on page 2 1.

Araujo, A., Golo, M., Viana, B., Sanches, F., Romero, R., e Marcacini, R. (2020). From bag-of-words to pre-trained neural language models: Improving automatic classification of app reviews for requirements engineering. In *Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional*, páginas 378–389. SBC. Cited on page 2 25.

Araujo, A. e Marcacini, R. M. (2021). Re-bert: Automatic extraction of software requirements from app reviews using bert language model. In *The 36th ACM/SIGAPP Symposium On Applied Computing*. Cited on pages 1, 24, 25, 29, 34, 36, 58, e 92.

Araujo, A. F., Gôlo, M. P., e Marcacini, R. M. (2022). Opinion mining for app reviews: an analysis of textual representation and predictive models. *Automated Software Engineering*, 29(1):1–30. Cited on pages 2, 3, 23, 24, 27, e 30.

Araujo, A. F., Gôlo, M. P. S., e Marcacini, R. M. (2021). Opinion mining for app reviews: an analysis of textual representation and predictive models. *Automated Software Engineering*, 29(1):5. Cited on page 2 89.

Aurum, A. e Wohlin, C. (2003). The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, 45(14):945–954. Cited on page 2 22.

Baccianella, S., Esuli, A., e Sebastiani, F. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. Cited on page 2 16.

Bartlett, M. S. (1937). Properties of sufficiency and statistical tests. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 160(901):268–282. Cited on page 2 82.

Baskar, A. (2013). Sentiment classification approaches – a review. *International Journal of Innovations in Engineering and Technology*, Vol. 3:22–31. Cited on page 2 16.

Bennett, K. H. e Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, páginas 73–87. Cited on page 2 2.

Blair-Goldensohn, S., Hannan, K., McDonald, R., Neylon, T., Reis, G., e Reynar, J. (2008). Building a sentiment summarizer for local service reviews. Cited on page 2 12.

Blei, D. M., Ng, A. Y., e Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022. Cited on pages 14, 19, e 27.

Bojanowski, P., Grave, E., Joulin, A., e Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146. Cited on page 2 36.

Box, G. E., Jenkins, G. M., Reinsel, G. C., e Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons. Cited on page 2 20.

Brocklebank, J. C. e Dickey, D. A. (2003). *SAS for forecasting time series*. John Wiley & Sons. Cited on page 2 21.

Brody, S. e Elhadad, N. (2010). An unsupervised aspect-sentiment model for online reviews. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, páginas 804–812. Cited on page 2 14.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, páginas 1877–1901. Cited on pages 24 e 91.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2021). Language models are unsupervised multitask learners. *Journal of Machine Learning Research*, 22(79):1–21. Cited on page 2 24.

Budgen, D. (2003). *Software Design*. Addison-Wesley Professional. Cited on page 2 110.

Carenini, G., Ng, R. T., e Zwart, E. (2005). Extracting knowledge from evaluative text. In *Proceedings of the 3rd international conference on Knowledge capture*, páginas 11–18. Cited on page 2 19.

Chaouch, S., Mejri, A., e Ghannouchi, S. A. (2019). A framework for risk management in scrum development process. *Procedia Computer Science*, 164:187–192. Cited on pages 90 e 93.

Chatfield, C. (2003). *The analysis of time series: an introduction*. Chapman and hall/CRC. Cited on pages 19, 21, e 63.

Chatfield, C. e Xing, H. (2019). *The analysis of time series: an introduction with R*. CRC press. Cited on pages 20 e 39.

Chen, M. e Liu, X. (2011). Predicting popularity of online distributed applications: Itunes app store case analysis. In *Proceedings of the 2011 IConference*, iConference '11, pagina 661–663, New York, NY, USA. Association for Computing Machinery. Cited on page 2 25.

Chen, N., Lin, J., Hoi, S. C., Xiao, X., e Zhang, B. (2014). Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th international conference on software engineering*, páginas 767–778. Cited on pages 3, 26, 27, e 29.

Chen, P., Sun, Z., Bing, L., e Yang, W. (2017). Recurrent attention network on memory for aspect sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, páginas 452–461, Copenhagen, Denmark. Association for Computational Linguistics. Cited on page 2 15.

Cheng, C., Sa-Ngasoongsong, A., Beyca, O., Le, T., Yang, H., Kong, Z. J., e Bukkapatnam, S. T. (2015). Time series forecasting for nonlinear and non-stationary processes: a review and comparative study. *IIE Transactions*, 47(10):1053–1071. Cited on page 2 19.

Cheng, J., Zhao, S., Zhang, J., King, I., Zhang, X., e Wang, H. (2017). Aspect-level sentiment classification with heat (hierarchical attention) network. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pagina 97–106, New York, NY, USA. Association for Computing Machinery. Cited on page 2 15.

Cho, K., Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., e Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. Cited on page 2 15.

Ciurumelea, A., Schaufelbühl, A., Panichella, S., e Gall, H. C. (2017). Analyzing reviews and code of mobile apps for better release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, páginas 91–102. Cited on page 2 26.

Clements, P., Bass, L., e Kazman, R. (2002). *Software Architecture in Practice*. Addison-Wesley Professional. Cited on page 2 110.

Cowpertwait, P. S. e Metcalfe, A. V. (2009). *Introductory time series with R*. Springer Science & Business Media. Cited on pages 21 e 22.

Dabrowski, J., Letier, E., Perini, A., e Susi, A. (2020). Mining user opinions to support requirement engineering: An empirical study. In Dustdar, S., Yu, E., Salinesi, C., Rieu, D., e Pant, V., editors, *Advanced Information Systems Engineering*, páginas 401–416, Cham. Springer International Publishing. Cited on pages 1, 2, 25, 34, 99, 100, e 101.

Dang, N. C., García, M. N. M., e de la Prieta, F. (2020). Sentiment analysis based on deep learning: A comparative study. *CoRR*, abs/2006.03541. Cited on pages 12 e 15.

Davis, A. M. (2003). The art of requirements triage. *Computer*, 36(3):42–49. Cited on page 2 22.

De Clercq, O., Van de Kauter, M., Lefever, E., e Hoste, V. (2015). Lt3: Applying hybrid terminology extraction to aspect-based sentiment analysis. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, páginas 719–724. Cited on page 2 13.

De Mulder, W., Bethard, S., e Moens, M.-F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech and Language*, 30(1):61–98. Cited on page 2 23.

Devlin, J., Chang, M., Lee, K., e Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. Cited on pages 23, 36, e 57.

Di Sorbo, A., Panichella, S., Alexandru, C. V., Shimagaki, J., Visaggio, C. A., Canfora, G., e Gall, H. C. (2016). What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pagina 499–510, New York, NY, USA. Association for Computing Machinery. Cited on page 2 26.

Do, H. H., Prasad, P., Maag, A., e Alsadoon, A. (2019). Deep learning for aspect-based sentiment analysis: a comparative review. *Expert systems with applications*, 118:272–299. Cited on page 2 15.

Dong, L., Wei, F., Tan, C., Tang, D., Zhou, M., e Xu, K. (2014). Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)*, páginas 49–54. Cited on page 2 17.

Dragoni, M., Federici, M., e Rexha, A. (2019). An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Information Processing and Management*, 56(3):1103–1118. Cited on pages 25, 29, 34, e 92.

Du, H., Xu, X., Cheng, X., Wu, D., Liu, Y., e Yu, Z. (2016). Aspect-specific sentimental word embedding for sentiment analysis of online reviews. In *Proceedings of the 25th International Conference Companion on World Wide Web*, páginas 29–30. Cited on page 2 17.

Esuli, A. e Sebastiani, F. (2006). Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the fifth international conference on language resources and evaluation (LREC'06)*. Cited on page 2 16.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. The MIT Press. Cited on pages 16 e 19.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional. Cited on pages 115 e 116.

Galvis Carreño, L. V. e Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pagina 582–591. IEEE Press. Cited on pages 3, 26, 27, e 29.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. Cited on pages 113, 114, 115, e 116.

Gao, C., Zeng, J., Lyu, M. R., e King, I. (2018). Online app review analysis for identifying emerging issues. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pagina 48–58, New York, NY, USA. Association for Computing Machinery. Cited on pages 27 e 29.

Gao, C., Zeng, J., Wen, Z., Lo, D., Xia, X., King, I., e Lyu, M. R. (2021). Emerging app issue identification via online joint sentiment-topic tracing. *IEEE Transactions on Software Engineering*. Cited on page 2 27.

Gao, C., Zeng, J., Wen, Z., Lo, D., Xia, X., King, I., e Lyu, M. R. (2022). Emerging app issue identification via online joint sentiment-topic tracing. *IEEE Transactions on Software Engineering*, 48(08):3025–3043. Cited on pages 3, 27, 29, e 30.

Gao, C., Zheng, W., Deng, Y., Lo, D., Zeng, J., Lyu, M. R., e King, I. (2019). Emerging app issue identification from user feedback: Experience on wechat. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, páginas 279–288. Cited on pages 27 e 29.

García-Pablos, A., Cuadros, M., e Rigau, G. (2018). W2vlda: almost unsupervised system for aspect based sentiment analysis. *Expert Systems with Applications*, 91:127–137. Cited on page 2 14.

Genc-Nayebi, N. e Abran, A. (2017). A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software*, 125:207–219. Cited on page 2 89.

Ghani, R., Probst, K., Liu, Y., Krema, M., e Fano, A. (2006). Text mining for product attribute extraction. *SIGKDD Explor. Newsl.*, 8(1):41–48. Cited on page 2 10.

Girden, E. R. (1992). *ANOVA: Repeated measures*. Number 84. sage. Cited on page 2 82.

Gómez, M., Rouvoy, R., Monperrus, M., e Seinturier, L. (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, pagina 1–11. IEEE Press. Cited on pages 3, 26, e 27.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, páginas 857–871. Cited on page 2 18.

Groen, E. C., Doerr, J., e Adam, S. (2015). Towards crowd-based requirements engineering a research preview. In *Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23-26, 2015. Proceedings 21*, páginas 247–253. Springer. Cited on page 2 3.

Gu, X., Gu, Y., e Wu, H. (2017). Cascaded convolutional neural networks for aspect-based opinion summary. *Neural Processing Letters*, 46. Cited on page 2 17.

Gu, X. e Kim, S. (2015). "what parts of your apps are loved by users?" (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, páginas 760–770. Cited on pages 3, 26, 27, e 29.

Guha, S., Joshi, A., e Varma, V. (2015). Siel: aspect based sentiment analysis in reviews. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, páginas 759–766. Cited on page 2 13.

Guo, H., Zhu, H., Guo, Z., Zhang, X., e Su, Z. (2009). Product feature categorization with multilevel latent semantic association. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pagina 1087–1096, New York, NY, USA. Association for Computing Machinery. Cited on page 2 19.

Guzman, E., Alkadhi, R., e Seyff, N. (2016). A needle in a haystack: What do twitter users say about software? In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, páginas 96–105. Cited on page 2 26.

Guzman, E., Ibrahim, M., e Glinz, M. (2017). Mining twitter messages for software evolution. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pagina 283–284. IEEE Press. Cited on page 2 26.

Guzman, E. e Maalej, W. (2014). How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international require-*

ments engineering conference (RE), páginas 153–162. IEEE. Cited on pages 25, 27, 29, 34, e 92.

Hammad, M. e Inayat, I. (2018). Integrating risk management in scrum framework. In *2018 International Conference on Frontiers of Information Technology (FIT)*, páginas 158–163. IEEE. Cited on pages 90 e 93.

Hammad, M., Inayat, I., e Zahid, M. (2019). Risk management in agile software development: A survey. In *2019 International Conference on Frontiers of Information Technology (FIT)*, páginas 162–1624. IEEE. Cited on page 2 93.

Harman, M., Jia, Y., e Zhang, Y. (2012). App store mining and analysis: Msr for app stores. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, páginas 108–111. Cited on pages 24, 26, e 29.

Hastie, T. e Tibshirani, R. (1987). Generalized additive models: some applications. *Journal of the American Statistical Association*, 82(398):371–386. Cited on page 2 41.

Herbold, S. (2020). Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173. Cited on page 2 86.

Herbold, S., Trautsch, A., e Trautsch, F. (2020). On the feasibility of automated prediction of bug and non-bug issues. *Empirical Software Engineering*, 25(6):5333–5369. Cited on pages 3, 27, e 30.

Hochreiter, S. e Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. Cited on page 2 14.

Hofmann, T. (2013). Probabilistic latent semantic analysis. Cited on page 2 14.

Hoon, L., Vasa, R., Schneider, J.-G., e Mouzakis, K. (2012). A preliminary analysis of vocabulary in mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pagina 245–248, New York, NY, USA. Association for Computing Machinery. Cited on page 2 25.

Howard, J. e Ruder, S. (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, páginas 328–339, Melbourne, Australia. Association for Computational Linguistics. Cited on page 2 24.

Hu, M. e Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge*

Discovery and Data Mining, KDD '04, pagina 168–177, New York, NY, USA. Association for Computing Machinery. Cited on pages 10, 12, e 16.

Huang, S., Liu, X., Peng, X., e Niu, Z. (2012). Fine-grained product features extraction and categorization in reviews opinion mining. In *2012 IEEE 12th International Conference on Data Mining Workshops*, páginas 680–686. IEEE. Cited on page 2 13.

Hutto, C. e Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, páginas 216–225. Cited on page 2 60.

Iacob, C. e Harrison, R. (2013). Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pagina 41–44. IEEE Press. Cited on pages 3, 26, 27, e 29.

Ionita, D., van der Velden, C., Ikkink, H.-J. K., Neven, E., Daneva, M., e Kuipers, M. (2019). Towards risk-driven security requirements management in agile software development. In *Information Systems Engineering in Responsible Information Systems: CAiSE Forum 2019, Rome, Italy, June 3–7, 2019, Proceedings 31*, páginas 133–144. Springer. Cited on page 2 93.

Ishaq, A., Asghar, S., e Gillani, S. A. (2020). Aspect-based sentiment analysis using a hybridized approach based on cnn and ga. *IEEE Access*, 8:135499–135512. Cited on page 2 12.

Iyer, S., Lin, X. V., Pasunuru, R., Mihaylov, T., Simig, D., Yu, P., Shuster, K., Wang, T., Liu, Q., Koura, P. S., et al. (2022). Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*. Cited on page 2 103.

Jakob, N. e Gurevych, I. (2010). Extracting opinion targets in a single- and cross-domain setting with conditional random fields. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pagina 1035–1045, USA. Association for Computational Linguistics. Cited on page 2 13.

Jangid, H., Singhal, S., Shah, R. R., e Zimmermann, R. (2018). Aspect-based financial sentiment analysis using deep learning. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pagina 1961–1966, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee. Cited on page 2 15.

Jebbara, S. e Cimiano, P. (2017). Aspect-based sentiment analysis using a two-step neural network architecture. *CoRR*, abs/1709.06311. Cited on page 2 15.

Jeyapriya, A. e Selvi, C. S. K. (2015). Extracting aspects and mining opinions in product reviews using supervised learning algorithm. In *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, páginas 548–552. Cited on page 2 13.

Jin, W. e Ho, H. H. (2009). A novel lexicalized hmm-based learning framework for web opinion mining. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pagina 465–472, New York, NY, USA. Association for Computing Machinery. Cited on page 2 13.

Jo, Y. e Oh, A. H. (2011). Aspect and sentiment unification model for online review analysis. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pagina 815–824, New York, NY, USA. Association for Computing Machinery. Cited on page 2 14.

Johann, T., Stanik, C., Maalej, W., 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)*, páginas 21–30. IEEE. Cited on pages 25, 29, 34, e 92.

Johanssen, J. O., Kleebaum, A., Bruegge, B., e Paech, B. (2019). How do practitioners capture and utilize user feedback during continuous software engineering? In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, páginas 153–164. IEEE. Cited on page 2 1.

Johnson, J., Douze, M., e Jégou, H. (2017). Billion-scale similarity search with gpus. Cited on page 2 59.

Karimi, A., Rossi, L., e Prati, A. (2020). Improving bert performance for aspect-based sentiment analysis. Cited on page 2 15.

Khalid, H., Shihab, E., Nagappan, M., e Hassan, A. E. (2015). What do mobile app users complain about? *IEEE Software*, 32(3):70–77. Cited on pages 3, 26, e 27.

Khan, K., Baharudin, B., Khan, A., e Ullah, A. (2014). Mining opinion components from unstructured reviews: A review. *Journal of King Saud University - Computer and Information Sciences*, 26(3):258–275. Cited on page 2 11.

Kirchgässner, G., Wolters, J., e Hassler, U. (2013). *Introduction to Modern Time Series Analysis*. Number 978-3-642-33436-8 in Springer Texts in Business and Economics. Springer. Cited on page 2 22.

Ku, L.-W., Liang, Y.-T., e Chen, H.-H. (2006). Opinion extraction, summarization and tracking in news and blog corpora. In *Proceedings of AAAI*, páginas 100–107. Cited on page 2 12.

Lafferty, J. D., McCallum, A., e Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pagina 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. Cited on page 2 13.

Li, B., Zhou, L., Feng, S., e Wong, K.-F. (2010). A unified graph model for sentence-based opinion retrieval. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, páginas 1367–1375, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 2 13.

Li, X., Bing, L., Li, P., Lam, W., e Yang, Z. (2018). Aspect term extraction with history attention and selective transformation. páginas 4194–4200. Cited on page 2 15.

Li, X. e Lam, W. (2017). Deep multi-task learning for aspect term extraction with memory interaction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, páginas 2886–2892, Copenhagen, Denmark. Association for Computational Linguistics. Cited on page 2 15.

Licorish, S. A., Savarimuthu, B. T. R., e Keertipati, S. (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, pagina 108–117, New York, NY, USA. Association for Computing Machinery. Cited on pages 2, 3, 4, e 26.

Lientz, B. P. e Swanson, E. B. (1980). *Software maintenance management*. Addison-Wesley Longman Publishing Co., Inc. Cited on page 2 2.

Lim, H. S. (2004). Improving knn based text classification with well estimated parameters. In *International Conference on Neural Information Processing*, páginas 516–523. Springer. Cited on page 2 16.

Lima, V., Araújo, A., e Marcacini, R. (2022). Temporal dynamics of requirements engineering from mobile app reviews. *PeerJ Computer Science*, 8:e874. Cited on pages 2 e 3.

Lin, B., Cassee, N., Serebrenik, A., Bavota, G., Novielli, N., e Lanza, M. (2022). Opinion mining for software development: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 31(3):Article 38. Cited on page 2 28.

Lin, C. e He, Y. (2009). Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pagina 375–384, New York, NY, USA. Association for Computing Machinery. Cited on page 2 14.

Liu, B. (2012a). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167. Cited on page 2 1.

Liu, B. (2012b). *Sentiment Analysis and Opinion Mining*. Morgan amp; Claypool Publishers. Cited on pages 10, 11, 12, 15, 16, e 17.

Liu, B. e Zhang, L. (2012). A survey of opinion mining and sentiment analysis. In *Mining text data*, páginas 415–463. Springer. Cited on page 2 16.

Liu, H., Chatterjee, I., Zhou, M., Lu, X. S., e Abusorrah, A. (2020). Aspect-based sentiment analysis: A survey of deep learning methods. *IEEE Transactions on Computational Social Systems*, 7(6):1358–1375. Cited on page 2 16.

Liu, P., Joty, S., e Meng, H. (2015). Fine-grained opinion mining with recurrent neural networks and word embeddings. páginas 1433–1443. Cited on page 2 15.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., e Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. Cited on page 2 23.

Logan IV, R., Balažević, I., Wallace, E., Petroni, F., Singh, S., e Riedel, S. (2022). Cutting down on prompts and parameters: Simple few-shot learning with language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, páginas 2824–2835. Cited on page 2 92.

Maalej, W. e Nabil, H. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, páginas 116–125. Cited on pages 3, 26, e 27.

Maalej, W., Nayebi, M., Johann, T., e Ruhe, G. (2016). Toward data-driven requirements engineering. *IEEE Software*, 33:48–54. Cited on page 2 22.

Maalej, W. e Pagano, D. (2011). On the socialness of software. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, páginas 864–871. Cited on page 2 22.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on*

mathematical statistics and probability, volume 1, páginas 281–297. Oakland, CA, USA. Cited on pages 18 e 61.

Mai, L. e Le, B. (2018). Aspect-based sentiment analysis of vietnamese texts with deep learning. In Nguyen, N. T., Hoang, D. H., Hong, T.-P., Pham, H., e Trawiński, B., editors, *Intelligent Information and Database Systems*, páginas 149–158, Cham. Springer International Publishing. Cited on page 2 15.

Makridakis, S. (1993). Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529. Cited on page 2 46.

Malgaonkar, S., Licorish, S. A., e Savarimuthu, B. T. R. (2022). Prioritizing user concerns in app reviews – a study of requests for new features, enhancements and bug fixes. *Information and Software Technology*, 144:106798. Cited on pages 1, 3, 4, 30, e 84.

Malik, H., Shakshuki, E. M., e Yoo, W.-S. (2020). Comparing mobile apps by identifying 'hot' features. *Future Generation Computer Systems*, 107:659–669. Cited on pages 25 e 29.

Marcacini, R. M. (2014). *Aprendizado de máquina com informação privilegiada: abordagens para agrupamento hierárquico de textos*. PhD thesis, Universidade de São Paulo. Cited on page 2 18.

Martin, W., Sarro, F., Jia, Y., Zhang, Y., e Harman, M. (2016). A survey of app store analysis for software engineering. *IEEE transactions on software engineering*, 43(9):817–847. Cited on page 2 1.

Mcilroy, S., Ali, N., e Hassan, A. E. (2016a). Fresh apps: An empirical study of frequently-updated mobile apps in the google play store. *Empirical Softw. Engg.*, 21(3):1346–1370. Cited on pages 2 e 84.

Mcilroy, S., Ali, N., Khalid, H., e E. Hassan, A. (2016b). Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Softw. Engg.*, 21(3):1067–1106. Cited on pages 3, 26, e 27.

Messaoud, M. B., Jenhani, I., Jemaa, N. B., e Mkaouer, M. W. (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*, pagina 805–816, Athens, Greece. Springer-Verlag. Cited on pages 3, 27, e 30.

Mihalcea, R. e Radev, D. (2011). *Graph-based natural language processing and information retrieval*. Cambridge university press. Cited on page 2 57.

Mikolov, T., Chen, K., Corrado, G., e Dean, J. (2013). Efficient estimation of word representations in vector space. Cited on pages 23 e 36.

Moghaddam, S. e Ester, M. (2010). Opinion digger: An unsupervised opinion miner from unstructured product reviews. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pagina 1825–1828, New York, NY, USA. Association for Computing Machinery. Cited on page 2 12.

Montgomery, D., Jennings, C., e Kulahci, M. (2015). *Introduction to Time Series Analysis and Forecasting*. Wiley Series in Probability and Statistics. Wiley. Cited on page 2 20.

More, P. e Ghotkar, A. (2016). A study of different approaches to aspect-based opinion mining. *International Journal of Computer Applications*, 145:11–15. Cited on pages 12, 13, e 14.

Mukherjee, A. e Liu, B. (2012). Aspect extraction through semi-supervised modeling. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, páginas 339–348, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 2 19.

Naing, W. Y. N. e Htike, Z. Z. (2015). State of the art machine learning techniques for time series forecasting: A survey. *Advanced Science Letters*, 21:3574–3576. Cited on page 2 20.

Narayanan, V., Arora, I., e Bhatia, A. (2013). Fast and accurate sentiment classification using an enhanced naive bayes model. In *International Conference on Intelligent Data Engineering and Automated Learning*, páginas 194–201. Springer. Cited on page 2 16.

Nayebi, M., Adams, B., e Ruhe, G. (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)*, volume 1, páginas 552–562. Cited on page 2 2.

Nayebi, M., Cho, H., e Ruhe, G. (2018a). App store mining is not enough for app improvement. *Empirical Softw. Engg.*, 23(5):2764–2794. Cited on page 2 26.

Nayebi, M., Kuznetsov, K., Chen, P., Zeller, A., e Ruhe, G. (2018b). Anatomy of functionality deletion: An exploratory study on mobile apps. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pagina 243–253, New York, NY, USA. Association for Computing Machinery. Cited on pages 2 e 26.

Nayebi, M., Marbouti, M., Quapp, R., Maurer, F., e Ruhe, G. (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)*, páginas 57–66. Cited on pages 3, 26, e 27.

Nguyen, T. H. e Shirai, K. (2015). Phrasernn: Phrase recursive neural network for aspect-based sentiment analysis. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, páginas 2509–2514. Cited on page 2 17.

Noei, E., Zhang, F., e Zou, Y. (2021). Too many user-reviews! what should app developers look at first? *IEEE Transactions on Software Engineering*, 47(2):367–378. Cited on pages 3, 26, 27, e 92.

Otter, D. W., Medina, J. R., e Kalita, J. K. (2021). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624. Cited on page 2 23.

Pagano, D. e Maalej, W. (2013). User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, páginas 125–134. Cited on pages 3, 25, 26, e 27.

Palomba, F., Linares-Vásquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., e De Lucia, A. (2015). User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, páginas 291–300. Cited on page 2 28.

Palomba, F., Linares-Vásquez, M., Bavota, G., Oliveto, R., Penta, M. D., Poshyvanyk, D., e Lucia, A. D. (2018). Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software*, 137:143–162. Cited on pages 2, 26, e 29.

Paltrinieri, N., Comfort, L., e Reniers, G. (2019). Learning about risk: Machine learning for risk assessment. *Safety science*, 118:475–486. Cited on page 2 89.

Pang, B., Lee, L., e Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*. Cited on page 2 16.

Parmezan, A. e Batista, G. (2016). Descrição de modelos estatísticos e de aprendizado de máquina para predição de séries temporais. Cited on page 2 21.

Pavlopoulos, J. e Androutsopoulos, I. (2014). Multi-granular aspect aggregation in aspect-based sentiment analysis. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, páginas 78–87. Cited on page 2 19.

Pekar, V., Afzal, N., e Bohnet, B. (2014). Ubham: Lexical resources and dependency parsing for aspect-based sentiment analysis. Cited on page 2 13.

Pennington, J., Socher, R., e Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, páginas 1532–1543, Doha, Qatar. Association for Computational Linguistics. Cited on page 2 36.

Phong, M. V., Nguyen, T. T., Pham, H. V., e Nguyen, T. T. (2015). Mining user opinions in mobile app reviews: A keyword-based approach (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, páginas 749–759. Cited on pages 27 e 29.

Pilliang, M., Munawar, Hadi, M. A., Firmansyah, G., e Tjahjono, B. (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)*, páginas 137–142. Cited on pages 28, 30, 89, 90, 92, e 93.

Pontiki, M., Galanis, D., Papageorgiou, H., Androutsopoulos, I., Manandhar, S., Al-Smadi, M., Al-Ayyoub, M., Zhao, Y., Qin, B., De Clercq, O., et al. (2016). Semeval-2016 task 5: Aspect based sentiment analysis. In *International workshop on semantic evaluation*, páginas 19–30. Cited on page 2 11.

Popescu, A.-M. e Etzioni, O. (2005). Extracting product features and opinions from reviews. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pagina 339–346, USA. Association for Computational Linguistics. Cited on page 2 12.

Poria, S., Cambria, E., e Gelbukh, A. (2016a). Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108:42–49. New Avenues in Knowledge Bases for Natural Language Processing. Cited on page 2 14.

Poria, S., Cambria, E., e Gelbukh, A. (2016b). Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108:42–49. New Avenues in Knowledge Bases for Natural Language Processing. Cited on page 2 15.

Poria, S., Cambria, E., Ku, L.-W., Gui, C., e Gelbukh, A. (2014). A rule-based approach to aspect extraction from product reviews. In *Proceedings of the second workshop on natural language processing for social media (SocialNLP)*, páginas 28–37. Cited on page 2 13.

Poria, S., Chaturvedi, I., Cambria, E., e Bisio, F. (2016c). Sentic lda: Improving on lda with semantic similarity for aspect-based sentiment analysis. In *2016 international joint conference on neural networks (IJCNN)*, páginas 4465–4473. IEEE. Cited on page 2 14.

Pozzi, F. A., Fersini, E., Messina, E., e Liu, B. (2017). Challenges of sentiment analysis in social networks: an overview. *Sentiment analysis in social networks*, páginas 1–11. Cited on page 2 10.

Qiu, G., Liu, B., Bu, J., e Chen, C. (2011). Opinion word expansion and target extraction through double propagation. *Comput. Linguist.*, 37(1):9–27. Cited on page 2 13.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. Cited on page 2 13.

Radford, A., Narasimhan, K., Salimans, T., e Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI Blog*. Retrieved from https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language$_u$nderstanding$_p$aper.pdf.Citedonpage2 24.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., e Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(9). Cited on page 2 24.

Rana, T. A. e Cheah, Y.-N. (2016). Aspect extraction in sentiment analysis: comparative analysis and survey. *Artificial Intelligence Review*, 46(4):459–483. Cited on page 2 12.

Rana, T. A. e Cheah, Y.-N. (2017). A two-fold rule-based model for aspect extraction. *Expert Systems with Applications*, 89:273–285. Cited on page 2 13.

Reenskaug, T. (1979). Models-views-controllers. Retrieved from https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. Cited on page 2 113.

Reimers, N. e Gurevych, I. (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)*, páginas 3973–3983. Cited on pages 36, 94, e 103.

Rietzler, A., Stabinger, S., Opitz, P., e Engl, S. (2019). Adapt or get left behind: Domain adaptation through bert language model finetuning for aspect-target sentiment classification. Cited on page 2 15.

Rodrigues, L. S., Rezende, S. O., Moura, M. F., e Marcacini, R. M. (2018). Agribusiness time series forecasting using perceptually important events. In *Latin American Computing Conference - CLEI*. IEEE. Cited on page 2 20.

Rokach, L. (2010). *A survey of Clustering Algorithms*, páginas 269–298. Cited on pages 17 e 18.

Ross, S. I., Martinez, F., Houde, S., Muller, M., e Weisz, J. D. (2023). The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, páginas 491–514. Cited on page 2 90.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65. Cited on page 2 37.

Ruder, S., Ghaffari, P., e Breslin, J. (2016). Nsight-1 at semeval-2016 task 5: Deep learning for multilingual aspect-based sentiment analysis. páginas 330–336. Cited on pages 14 e 17.

Saaty, T. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill International Book Company. Cited on page 2 25.

Sanh, V., Debut, L., Chaumond, J., e Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. Cited on pages 23 e 59.

Santos, B. N. D., Marcacini, R. M., e Rezende, S. O. (2021). Multi-domain aspect extraction using bidirectional encoder representations from transformers. *IEEE Access*, 9:91604–91613. Cited on pages 10 e 15.

Sarro, F., Al-Subaihin, A. A., Harman, M., Jia, Y., Martin, W., e Zhang, Y. (2015). Feature lifecycles as they spread, migrate, remain, and die in app stores. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, páginas 76–85. Cited on page 2 26.

Shams, M. e Baraani-Dastjerdi, A. (2017). Enriched lda (elda): Combination of latent dirichlet allocation with word co-occurrence analysis for aspect extraction. *Expert Systems with Applications*, 80:136–146. Cited on page 2 14.

Sommerville, I. (2013). *Engenharia de Software*. Pearson Education, [S.l.], 9th edition. Cited on page 2 113.

Song, Y., Wang, J., Jiang, T., Liu, Z., e Rao, Y. (2019). Targeted sentiment classification with attentional encoder network. In Tetko, I. V., Kůrková, V., Karpov, P., e Theis, F., editors, *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, páginas 93–103, Cham. Springer International Publishing. Cited on page 2 15.

Strobelt, H., Webson, A., Sanh, V., Hoover, B., Beyer, J., Pfister, H., e Rush, A. M. (2022). Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE transactions on visualization and computer graphics*, 29(1):1146–1156. Cited on page 2 92.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional. Cited on page 2 110.

Taboada, M., Brooke, J., Tofiloski, M., Voll, K., e Stede, M. (2011). Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307. Cited on page 2 16.

Tang, D., Qin, B., e Liu, T. (2016). Aspect level sentiment classification with deep memory network. *arXiv preprint arXiv:1605.08900*. Cited on page 2 17.

Tay, Y., Tuan, L. A., e Hui, S. C. (2018). Learning to attend via word-aspect associative fusion for aspect-based sentiment analysis. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32. Cited on page 2 17.

Taylor, S. J. e Letham, B. (2018). Forecasting at Scale. *The American Statistician*, 72(1):37–45. Cited on page 2 41.

Toh, Z. e Jian, S. (2016). Nlangp at semeval-2016 task 5: Improving aspect based sentiment analysis using neural network features. páginas 282–288. Cited on page 2 14.

Toh, Z. e Su, J. (2016). Nlangp at semeval-2016 task 5: Improving aspect based sentiment analysis using neural network features. In *Proceedings of the 10th*

international workshop on semantic evaluation (SemEval-2016), páginas 282–288. Cited on page 2 19.

Trisna, K. W. e Jie, H. J. (2022). Deep learning approach for aspect-based sentiment classification: A comparative review. *Applied Artificial Intelligence*, 0(0):1–37. Cited on page 2 11.

Tsytsarau, M. e Palpanas, T. (2011). Survey on mining subjective data on the web. *KDD*, 24:478–514. Cited on pages 10 e 15.

Tudor, D. e Walter, G. (2006). Using an agile approach in a large, traditional organization. In *AGILE 2006 (AGILE'06)*, páginas 7 pp.–373. Cited on page 2 25.

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, páginas 99–114. Cited on page 2 82.

Vargas, F. A. e Pardo, T. A. S. (2018). Aspect clustering methods for sentiment analysis. In *International conference on computational processing of the Portuguese language*, páginas 365–374. Springer. Cited on page 2 19.

Vasa, R., Hoon, L., Mouzakis, K., e Noguchi, A. (2012). A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pagina 241–244, New York, NY, USA. Association for Computing Machinery. Cited on page 2 25.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., e Polosukhin, I. (2017a). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pagina 6000–6010, Red Hook, NY, USA. Curran Associates Inc. Cited on page 2 23.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., e Polosukhin, I. (2017b). Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008. Cited on page 2 24.

Vendramin, L., Campello, R. J., e Hruschka, E. R. (2010). Relative clustering validity criteria: A comparative overview. *Statistical analysis and data mining: the ASA data science journal*, 3(4):209–235. Cited on page 2 38.

Villarroel, L., Bavota, G., Russo, B., Oliveto, R., e Di Penta, M. (2016). Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, páginas 14–24. Cited on pages 3, 26, 27, e 29.

Vu, P. M., Pham, H. V., Nguyen, T. T., e Nguyen, T. T. (2016). Phrase-based extraction of user opinions in mobile app reviews. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, páginas 726–731. Cited on pages 27 e 29.

Wang, B., Liakata, M., Zubiaga, A., e Procter, R. (2017). TDParse: Multi-target-specific sentiment recognition on Twitter. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, páginas 483–493, Valencia, Spain. Association for Computational Linguistics. Cited on page 2 15.

Wang, H. e Ester, M. (2014). A sentiment-aligned topic model for product aspect rating prediction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, páginas 1192–1202, Doha, Qatar. Association for Computational Linguistics. Cited on page 2 14.

Wang, J., Xu, B., e Zu, Y. (2021). Deep learning for aspect-based sentiment analysis. In *2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, páginas 267–271. Cited on page 2 15.

Wang, W., Pan, S. J., Dahlmeier, D., e Xiao, X. (2016a). Recursive neural conditional random fields for aspect-based sentiment analysis. *arXiv preprint arXiv:1603.06679*. Cited on page 2 17.

Wang, Y., Huang, M., Zhu, X., e Zhao, L. (2016b). Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, páginas 606–615. Cited on page 2 17.

Williams, G., Tushev, M., Ebrahimi, F., e Mahmoud, A. (2020). Modeling user concerns in sharing economy: the case of food delivery apps. *Automated Software Engineering*, 27. Cited on page 2 44.

Williamson, E. (2010). *Lists, Decisions and Graphs*. S. Gill Williamson. Cited on page 2 57.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., e Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Norwell, MA, USA. Cited on pages 72, 74, 77, 80, 85, 86, 98, 99, e 102.

Wu, H., Gu, Y., Sun, S., e Gu, X. (2016). Aspect-based opinion summarization with convolutional neural networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, páginas 3157–3163. IEEE. Cited on page 2 17.

Wu, Y., Zhang, Q., Huang, X., e Wu, L. (2009). Phrase dependency parsing for opinion mining. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pagina 1533–1541, USA. Association for Computational Linguistics. Cited on page 2 12.

Wunsch, D. e Xu, R. (2008). *Clustering*. IEEE Press Series on Computational Intelligence. Wiley. Cited on pages 17 e 18.

Xiaosong, L., Shushi, L., Wenjun, C., e Songjiang, F. (2009). The application of risk matrix to software project risk management. In *2009 International Forum on Information Technology and Applications*, volume 2, páginas 480–483. IEEE. Cited on pages 89 e 93.

Xu, J., Chen, D., Qiu, X., e Huang, X. (2016). Cached long short-term memory neural networks for document-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, páginas 1660–1669, Austin, Texas. Association for Computational Linguistics. Cited on pages 14 e 15.

Xu, J., Tang, B., Liu, X., e Xiong, F. (2020). Automatic aspect-based sentiment analysis (aabsa) from customer reviews. In *AffCon@ AAAI*, páginas 47–66. Cited on page 2 19.

Xu, L., Lin, J., Wang, L., Yin, C., e Wang, J. (2017). Deep convolutional neural network based approach for aspect-based sentiment analysis. páginas 199–204. Cited on pages 14 e 17.

Xueke, X., Xueqi, C., Songbo, T., Yue, L., e Shen, H. (2013). Aspect-level opinion mining of online customer reviews. *Communications, China*, 10:25–41. Cited on page 2 14.

Yang, B. e Cardie, C. (2013). Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, páginas 1640–1649. Cited on page 2 13.

Zeng, B., Yang, H., Xu, R., Zhou, W., e Han, X. (2019a). Lcf: A local context focus mechanism for aspect-based sentiment classification. *Applied Sciences*, 9:3389. Cited on page 2 15.

Zeng, Y., Lan, T., Wu, Z., e Liu, Q. (2019b). Bi-memory based attention model for aspect level sentiment classification. *Chinese Journal of Computers*, 42(8):1845–1857. Cited on page 2 15.

Zhai, Z., Liu, B., Xu, H., e Jia, P. (2011a). Clustering product features for opinion mining. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pagina 347–354, New York, NY, USA. Association for Computing Machinery. Cited on page 2 19.

Zhai, Z., Liu, B., Xu, H., e Jia, P. (2011b). Constrained lda for grouping product features in opinion mining. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, páginas 448–459. Springer. Cited on page 2 19.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. (2022). Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*. Cited on pages 90, 92, e 94.

Zhang, S. e. a. (2022). Opt: Open pre-trained transformer language models. *ArXiv*. Cited on page 2 24.

Zhang, Y. e Lin, Z. (2018). Predicting the helpfulness of online product reviews: A multilingual approach. *Electronic Commerce Research and Applications*, 27:1–10. Cited on page 2 25.

Zhang, Z., Nie, J.-Y., e Wang, H. (2015). Tjudem: a combination classifier for aspect category detection and sentiment polarity classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, páginas 772–777. Cited on page 2 11.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., e Batista-Navarro, R. T. (2020). Natural language processing (nlp) for requirements engineering: A systematic mapping study. *arXiv preprint arXiv:2004.01099*. Cited on page 2 3.

Zhao, W. X., Jiang, J., Yan, H., e Li, X. (2010). Jointly modeling aspects and opinions with a maxent-lda hybrid. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pagina 56–65, USA. Association for Computational Linguistics. Cited on page 2 14.

Zhao, Y., Qin, B., e Liu, T. (2014). Clustering product aspects using two effective aspect relations for opinion mining. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, páginas 120–130. Springer. Cited on page 2 19.

Zhu, Y., Kiros, R., Zemel, R. S., Salakhutdinov, R., Urtasun, R., Torralba, A., e Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE*

International Conference on Computer Vision (ICCV), Santiago, Chile. Cited on page 2 24.

Zhuang, L., Jing, F., e Zhu, X.-Y. (2006). Movie review mining and summarization. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pagina 43–50, New York, NY, USA. Association for Computing Machinery. Cited on page 2 12.