Mining App Reviews to Support Software Engineering

Jacek Dąbrowski

A dissertation submitted in fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London

University College London

Department of Computer Science

Software Systems Engineering Group

6 June 2022

Declaration

I, Jacek Dąbrowski confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis. Parts of this thesis have been published in the following papers:

- Chapter 2 has been published as J. Dąbrowski, E. Letier, A. Perini and A. Susi, "Analysing App Reviews for Software Engineering: A Systematic Literature Review" in *Empirical Software Engineering (EMSE)*, 27(2):43, 2022.
- Chapter 3 has been published as J. Dąbrowski, E. Letier, A. Perini and A. Susi, "Mining User Feedback For Software Engineering: Use Cases and Reference Architecture", in *2022 IEEE 30th International Requirements Engineering Conference (RE'22)*, August 2022, (To Appear).
- Chapter 4 has been published as J. Dąbrowski, E. Letier, A. Perini and A. Susi, "Mining User Opinions to Support Requirement Engineering: An Empirical Study" in *32nd International Conference on Advanced Information Systems Engineering (CAiSE'20)*, pp. 401–416, June 2020, [Distinguished Paper].

The following manuscript has been submitted and is under review:

- Chapter 4 and Chapter 5 are submitted for publication as J. Dąbrowski, E. Letier, A. Perini and A. Susi, "Mining and Searching App Reviews for Requirements Engineering: Evaluation and Replication Studies" in *Information Systems (IS)*, 2022 [Special Issue: Distinguished Papers of CAiSE'20].

The following papers have been published during the course of this PhD programme, although they do not form a part of this thesis:

- J. Dąbrowski, E. Letier, A. Perini and A. Susi, "Finding and Analyzing App Reviews Related to Specific Features: A Research Preview" in *25th International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ'19)*, pp. 183-189, March 2019. The paper presents a preliminary empirical study that have been extended in Chapter 5.

- J. Dąbrowski, “Towards an adaptive framework for goal-oriented strategic decision-making” in *2017 IEEE 25th International Requirements Engineering Conference (RE’17)*, pp. 538–543, September 2017.
- J. Dąbrowski, F. M. Kifetew, D. Muñante, E. Letier, A. Siena, and A. Susi, “Discovering requirements through goal-driven process mining” in *2017 IEEE 2nd International Workshop on Crowd-Based Requirements Engineering (CrowdRE)*, pp. 199–203, September 2017.

Abstract

The thesis studies how mining app reviews can support software engineering.

App reviews —short user reviews of an app in app stores— provide a potentially rich source of information to help software development teams maintain and evolve their products. Exploiting this information is however difficult due to the large number of reviews and the difficulty in extracting useful actionable information from short informal texts.

A variety of app review mining techniques have been proposed to classify reviews and to extract information such as feature requests, bug descriptions, and user sentiments but the usefulness of these techniques in practice is still unknown. Research in this area has grown rapidly, resulting in a large number of scientific publications (at least 182 between 2010 and 2020) but nearly no independent evaluation and description of how diverse techniques fit together to support specific software engineering tasks have been performed so far.

The thesis presents a series of contributions to address these limitations. We first report the findings of a systematic literature review in app review mining exposing the breadth and limitations of research in this area. Using findings from the literature review, we then present a reference model that relates features of app review mining tools to specific software engineering tasks supporting requirements engineering, software maintenance and evolution.

We then present two additional contributions extending previous evaluations of app review mining techniques. We present a novel independent evaluation of opinion mining techniques using an annotated dataset created for our experiment. Our evaluation finds lower effectiveness than initially reported by the techniques authors. A final part of the thesis, evaluates approaches in searching for app reviews pertinent to a particular feature. The findings show a general purpose search technique is more effective than the state-of-the-art purpose-built app review mining techniques; and suggest their usefulness for requirements elicitation.

Overall, the thesis contributes to improving the empirical evaluation of app review mining techniques and their application in software engineering practice. Researchers and developers of future app mining tools will benefit from the novel reference model, detailed experiments designs, and publicly available datasets presented in the thesis.

Impact Statement

App reviews —short user reviews of an app in app stores— are rich source of information that can guide software engineering activities. Unfortunately, analyzing app reviews manually to extract this information is challenging due to their large number and noisy nature. Most research in this area has addressed the problem by studying the content of app reviews; and proposed automatic techniques to provide useful information from the feedback.

Despite this increased effort, yet little is known about "How mining app reviews can support software engineering?" This challenges the understanding of the usefulness of mining app reviews; and how it can be used in practice.

This thesis provides a new knowledge, software artifacts and datasets to address the problem. These contributions are beneficial to the improvement of both research and practice in software engineering; as well as the quality of software products, in particular mobile apps.

The thesis provides a systematic literature review communicating a new knowledge about app review analysis for software engineering. With this knowledge, researchers and practitioners can have a comprehensive understanding behind the motivation for app review analysis; and how useful information in app reviews can be facilitated. More importantly, the thesis can increase the awareness of why mining app reviews can be useful for software engineering. The knowledge can help scientists to better motivate and position their own research works; to identify gaps in the literature, and to develop future research ideas. It can also ease the onboarding process for newcomers in this research area (e.g., doctoral students).

The thesis presents a unified description of software engineering use cases for mining app reviews; and defines a reference architecture realizing these use cases through a combination of natural language processing and data mining techniques. The use cases can increase the awareness about the benefits of mining user feedback; and how it can be systematically integrated into software engineering to improve existing practices. The reference architecture can guide the development and the evolution of future tools to realize these benefits.

The thesis presents two empirical studies extending previous evaluations of techniques for opinion mining and finding feature-related reviews. Both studies show the techniques achieve lower effectiveness than reported originally and raise an important question about their practical use. The findings can raise awareness of scientists for more rigorous evaluations of their app review mining techniques. The thesis provides datasets and detailed descriptions of evaluation procedure that can improve the quality of these evaluations.

Acknowledgements

Thank You, Lord, for everything that You allowed to cross my path. Thank You for the decisions that You allowed me to make, and the lessons that came from the decisions.

I would like to express my gratitude to my supervisors Emmanuel Letier, Anna Perini and Angelo Susi for their guidance, advice throughout the completion of this doctorate as well as for giving me great freedom for pursuing my research.

Many thanks to Jens Krinke and David Clark, my examiners in the transfer viva, for meticulously reading my thesis and feedback helping me to substantially improve it.

I would also like to thank Alessio Ferrari and Earl T. Barr, the examiners of my final viva, for accepting the task of evaluating this thesis and the inspiring discussion during the exam.

No words are enough to express my gratitude to colleagues at FBK: Fitsum Kifetew, Matteo Biagiola and Biniam Demissie, whose advice and help were always available.

My appreciation goes to Accenture, who supported me to pursue my PhD. “High performance. Delivered” motto has become the true essence of my life.

I am forever indebted to my parents to whom I owe everything. My dear father and mother have always made me feel their continuing love and supported to pursue my dreams. I am also grateful for the continuous love and support of my sisters Ania and Basia.

And last but not least, I would like to thank Sylwia my beautiful and charming wife who supported me over these years with all her love and patience. Sylwia has stuck with me through thick and thin; and she has never doubted me for a moment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

App reviews —short user reviews of an app in app stores— provide a rich source of on-line user feedback [1, 2, 3]; the feedback conveys information about user experience with apps, user opinions about app features (functional attributes) [4], qualities (non-functional attributes) [5], description of usage scenarios [2] as well as different types of user requests [6]. The information can help development teams to maintain and evolve their software products [7]. For instance, in requirements engineering, analyzing the feedback can help engineers to elicit new requirements about their features that users desire [8, 9]; in design, user-submitted content may provide valuable design recommendations on how to improve interface layout, boost readability and ease app navigation [10, 11]; in testing, in addition to finding bugs [12, 13], studying reviews can inform developers about the general users reactions to released beta versions of their apps [14, 15]; whereas, in maintenance, examining user comments may help to identify modification requests [16, 17] and prioritise expected enhancements [18].

Studies based on interviews highlight the importance that app developers give to information in app reviews to guide the evolution and maintenance of their apps (e.g., [15, 7, 19]). Practitioners however still use a manual approach for analysing the feedback [7, 19]. Unfortunately, analyzing app reviews manually to exploit the useful information is challenging due to their large number and the difficulty in extracting actionable information from short informal texts [7, 20]. Popular apps like WhatsApp Messenger can receive more than 300,000 reviews per month [21]; moreover, the review content can vary from informative and helpful one to content conveying hate and spam [1, 2].

A large and diverse research effort has been made to understand what useful information can be found in app reviews [1]; how the information can be facilitated using automatic approaches [22, 23]; and how it can help software engineers [7, 24]. Research in this area has grown rapidly, resulting in a large number of scientific publications (at least 182 between 2012 and 2020). A variety of review mining tools and techniques have been proposed to address the problem [24, 25]; for example, to classify reviews by topics [26], to extract information like bug descriptions [27], or to analyze user opinions about features [4].

1.1 Problem Statement

Despite the increased research effort, research in this area suffers from several limitations: no comprehensive and structured knowledge about the research field of analysing app reviews for software engineering; little attention paid to software engineering use cases of review mining techniques; and limited evidence supporting the usefulness of these techniques in practise.

No comprehensive and structured knowledge about the research field of analysing app reviews for software engineering. The information about app review mining approaches, their usage for software engineering as well as their empirical evaluations is currently scattered in the literature. Consequently, there is a lack of comprehensive understanding why app review analysis can be useful for software engineering; what information can app review mining techniques facilitate; and to what extent the techniques can be used in practice. Existing surveys identify app review analysis as an important topic within a broader area of app store analysis without a detailed investigation of this research direction [1]; or they focus on tools and techniques for a specific type of app review analysis e.g., for opinion mining [25, 28]. None of these studies, however, provides a through knowledge on how app review analysis can support software engineering.

Little attention paid to software engineering use cases. The existing literature has paid insufficient attention to software engineering use cases of app review mining approaches; how the techniques help software engineers for specific activities has not been their main focus [8, 29]. For example, studies on features extraction approaches [8, 30] claim they support requirements elicitation but do not explain how a requirements engineer would use their approaches during elicitation nor how using these approaches would address problems that developers currently face during elicitation. The lack of justifications challenges the understanding of how the approaches can be used in practice, and in general how app review mining can be exploited for software engineering purposes.

Limited evidence supporting the adequacy of mining approaches. Evidence supporting the validity of review mining approach do not come from independent empirical evaluation. Almost no replication study has been conducted to confirm the validity of existing results [31]. In fact, studies pay little attention to provide replication packages, including evaluation dataset and evaluation procedure. Furthermore, studies provide limited discussion of their evaluation results in the context of software engineering use cases. As a result, the strength of evidence supporting the adequacy of these approaches for software engineering purposes is limited.

1.2 Thesis Objectives and Contributions

The thesis overall objective is to study how mining app reviews can support software engineering; and specifically address the aforementioned limitations. To this end, the thesis presents four contributions:

1. **A systematic literature review on app review analysis for software engineering.** To provide a comprehensive understanding of app review analysis for software engineering, the thesis conducts a systematic literature review. The study organizes the knowledge into five high-level categories: a broad overview of app review analyses; the techniques facilitating the analyses; usage descriptions of the analyses for software engineering activities; a synthesis of empirical evaluations of app review mining approaches, including their procedures and results. The study also provides a critical discussion of the findings, limitations and recommendations for future research.

2. **Use cases and reference architecture for app review mining.** The thesis presents a unified description of software engineering use cases for app review mining; defines a novel reference architecture that realises these use cases through a combination of natural language processing and data mining techniques; and validates the architecture based on its partial implementations in commercial and research tools. The use cases illustrate the benefits of app review analysis for software engineers. The architecture provides a general framework to realise the benefits.

3. **Empirical study of opinion mining approaches.** The thesis presents a novel empirical evaluation of the application of opinion mining techniques for analysing app reviews. The study benchmarks three approaches in terms of their effectiveness in performing feature extraction and sentiment analysis tasks using a new dataset containing 1,000 app reviews annotated with 1,521 opinions and automated methods comparing the output of the approaches to this dataset. The study follows a more rigorous evaluation procedure than previous studies to address their limitations (e.g., subjectivity bias); it reveals the evaluated approaches achieve lower effectiveness than originally reported.

4. **Empirical study of approaches searching for feature-related reviews.** The thesis presents a novel independent evaluation of three approaches searching for feature-related reviews. The study compares the approaches in terms of their capability in find-

ing reviews referring to concrete features of interest. Unlike previous works, the study evaluates the approaches following the procedure for assessing the effectiveness of information retrieval systems; and it elaborates a new evaluation dataset containing 1,113 app reviews annotated with 24 queried features. The findings reveal a general-purpose searching tool achieves better accuracy than state-of-the-art techniques developed for app review analysis; and suggest it could be useful for requirement elicitation.

1.3 Structure of the Thesis

Chapter 2 provides a comprehensive systematic literature review on app review analysis for software engineering; it discusses the findings, limitations and recommendations for future research.

Chapter 3 demonstrates software engineering use cases for mining app reviews and a reference architecture realizing these use cases through a combination of natural language processing and data mining techniques.

Chapter 4 introduces the problem of opinion mining; then presents an empirical study evaluating opinion mining approaches in feature extraction and sentiment analysis tasks.

Chapter 5 presents a large scale empirical evaluation of approaches searching for feature-related reviews.

Chapter 6 concludes this thesis and discusses possible future research directions.

Chapter 2

Systematic Literature Review

This chapter was published in the journal of Empirical Software Engineering [32]. The first author‘s contribution to the paper was to formulate the idea, design and execute the experimentation, collect the results, analyse them, and write the manuscripts; other authors of the papers contributed to the research conceptualization and manuscript revision.

2.1 Introduction

App stores have become important platforms for the distribution of software products. In 2020, Google Play Store and Apple Store host over 5 million apps and are widely used for the discovery, purchase and updates of software products [33]. The emergence of these App Stores has had important effects on software engineering practices, notably by bridging the gap between developers and users, by increasing market transparency and by affecting release management [7]. In 2017, Martin et al. used the term ‘app store analysis’ to denote the emerging research using app store data for software engineering [1]. Their survey identified the richness and diversity of research using App Store data, notably for API analysis, feature analysis, release engineering, security and review analysis [1].

This study focuses on analysing app reviews for software engineering. App reviews are textual feedback associated with a star rating that app users can provide to other App Store users and app developers about their experience of an app [34]. Most reviews have length up to 675 characters [2]; and convey information on variety of topics such as feature requests, bug reports or user opinions [1, 26]. Analysing these reviews can benefit a range of software engineering activities. For example, for requirements engineering, analyzing app reviews can help software engineers to elicit new requirements about app features that users desire [9, 4]; for testing, app reviews can help in finding bugs [12, 35] and evaluating users’ reactions to released beta versions of their apps [7, 17]; during product evolution, analysing app reviews may help in identifying and prioritizing change requests [4, 18].

In recent years, scholars have been also studying on-line user feedback from other digital sources such as microblogs e.g., Twitter [36], on-line forums e.g., Reddit [37], or issue tracking systems e.g., JIRA [38]. Most research effort, however, has been focused on analyzing app reviews [24]. Supposedly, the large number of this data, their availability and their usefulness make app reviews unique and thus the most frequently studied type of on-line user feedback [24].

Significant research has been devoted to study what relevant information can be found in app reviews; how the information can be analysed using manual and automatic approaches; and how the information can help software engineers. However, this knowledge is scattered in the literature, and consequently there is no clear view on how app review analysis can support software engineering. The previous survey on app store data analysis [1] identified app review analysis as one important topic within the broader area of app store analysis but does not present a detailed comprehensive analysis of app review analysis techniques. Other literature reviews focus on specific types of review analysis such as opinion mining [25] and information extraction [22, 39] but they do not cover the whole range of research on analysing app reviews. In contrast, this study provides a systematic literature review of the whole range of research on analysing app reviews from the first paper published in 2012 up to the end of 2020. The study objectives are to:

- identify and classify the range of app review analysis proposed in the literature;
- identify the range of natural language processing and data mining techniques that support such analysis;
- identify the range of software engineering activities that app review analysis can support;
- report the methods and results of the empirical evaluation of app review analysis approaches.

To accomplish these objectives, we have conducted a systematic literature review following a well-defined methodology that identifies, evaluates, and interprets the relevant studies with respect to specific research questions [40]. After a systematic selection and screening procedure, we ended up with a set of 182 papers, covering the period 2012 to 2020, that were carefully examined to answer the research questions.

The primary contributions of the study are: (i) a synthesis of approaches and techniques for mining app reviews, (ii) new knowledge on how software engineering scenarios can be supported by mining app reviews, (iii) a summary of empirical evaluation of review mining approaches, and finally (iv) a study of literature growth patterns, gaps, and directions for future research.

2.2 Research Method

To conduct our systematic literature review, we followed the methodology proposed by Kitchenham [40]. We first defined research questions and prepared a review protocol, which guided our conduct of the review and the collection of data. We then performed the literature search and selection based on agreed criteria. The selected studies were read thoroughly, and data items as in Table 2.3 were collected using a data extraction form. Finally, we synthesized the results for reporting.

2.2.1 Research Questions

The primary aim of the study is to understand how analysing app reviews can support software engineering. Based on the objective, the following research questions have been derived:

- **RQ1:** What are the different types of app review analyses?
- **RQ2:** What techniques are used to realize app review analyses?
- **RQ3:** What software engineering activities are claimed to be supported by analysing app reviews?
- **RQ4:** How are app review analysis approaches empirically evaluated?
- **RQ5:** How well do existing app review analysis approaches support software engineers?

The aim of RQ1 is to identify and classify the different types of app review analysis presented in primary literature; where app review analysis refers to a task of examining, transforming, or modeling data with the goal of discovering useful information. The aim of RQ2 is to identify the range of techniques used to realize the different types of app review analysis identified in RQ1; where a technique stands for a way for facilitating app review analysis. The aim of RQ3 is to identify the range of software engineering activities that have been claimed to be supported by analyzing app reviews; where a software engineering activity refers to a task performed along the software development life cycle [41]. The aim of RQ4 is to understand how primary studies obtain empirical evidence about effectiveness and the perceived-quality of their review analysis approaches. The aim of RQ5 is to summarize the results of empirical studies about effectiveness and user-perceived quality of different types of app review analysis.

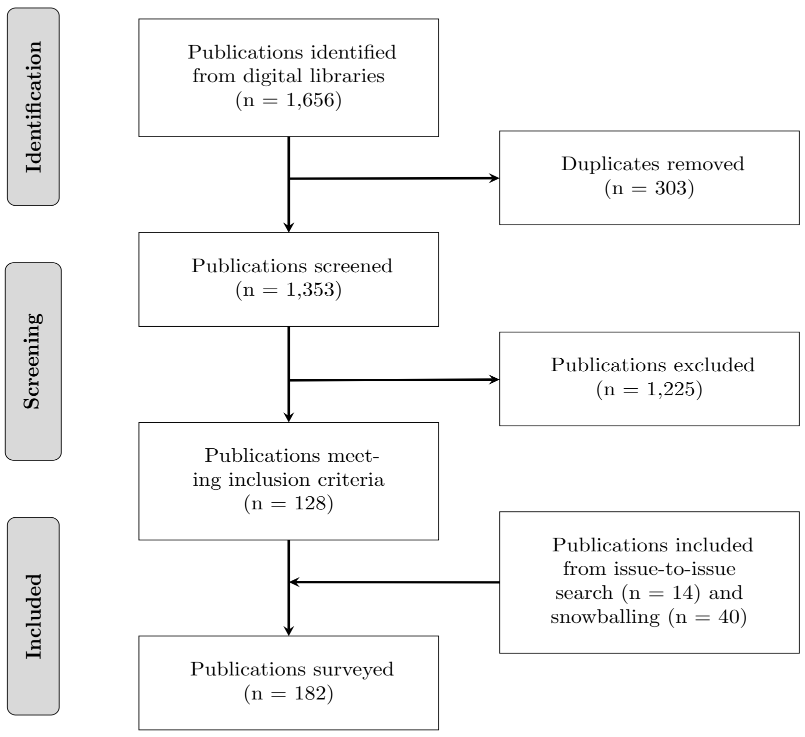

Identification
Screening
Included
Publications identified from digital libraries (n = 1,656)
Duplicates removed (n = 303)
Publications screened (n = 1,353)
Publications excluded (n = 1,225)
Publications meeting inclusion criteria (n = 128)
Publications included from issue-to-issue search (n = 14) and snowballing (n = 40)
Publications surveyed (n = 182)

Figure 2.1: PRISMA diagram showing study search and selection.

2.2.2 Literature Search and Selection

We followed a systematic search and selection process to collect relevant literature published between January 2010 and December 2020. We selected 2010 to be the initial period of our search as the earliest study of app store analysis had been reported that year [1]. Figure 2.1 outlines the process as a PRISMA diagram[1]; it illustrates the main steps of the process and their outcomes (the number of publications). The first author conducted the entire literature search and selection process.

The initial identification of publications was performed using keyword-based search on five major digital libraries: *ACM Digital Library*, *IEEE Xplore Digital Library*, *Springer Link Online Library*, *Wiley Online Library* and *Elsevier Science Direct*. The libraries cover titles from publishers in different subject fields like physical science, social science or computer science. We defined two search queries that we applied in both the meta-data (title + abstract) and full-text (when available) of the publications.

We applied the search queries in both the meta-data and full-text in *ACM Digital Library*, *IEEE Xplore Digital Library* and *Springer Link Online Library* as the libraries facilitated these options. As of *Wiley Online Library* and *Elsevier Science Direct*, we applied the queries on

[1]A description of the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) method can be found in [42].

meta-data only as the two libraries did not enable the full-text search.

To construct the first query, we looked at the content of several dozen publications analysing reviews for software engineering; we selected the publications that had been surveyed in the previous literature review on app store analysis [1]. We then identified the key terms that these papers share and used the terms to formulate the first query:

```
(‘app review mining’ OR ‘mining user review’ OR ‘review mining’ OR ‘review
analysis’ OR ‘analyzing user review’ OR ‘analyzing app review’) AND (‘app
store’)
```

To not omit other relevant papers not covered by this query, we formulated the second query based on phrases reflecting the key concepts of our research objective:

```
(‘app review’ OR ‘user review’ OR ‘app store review’ OR ‘user feedback’)
AND (‘software engineering’ OR ‘requirement engineering’ OR ‘software
requirement’ OR ‘software design’ OR ‘software construction’ OR ‘software
testing’ OR ‘software maintenance’ OR ‘software configuration’ OR ‘software
development’ OR ‘software quality’ OR ‘software coding’) AND (‘app store’)
```

We run the two queries sequentially. We first run the first query and then the second one. We then merged the search results of the two queries; the sequence of running the queries however did not have an impact on the merged results from the two queries.

The initial search via digital libraries resulted in 1,656 studies, where 303 of them were duplicated. We screened 1,353 studies obtained through the initial search and selected them in accordance with the inclusion and exclusion criteria (see Table 2.1). To ensure the reliability of our screening process, the four authors of this study independently classified a sample of 20 papers (each paper was assigned to two authors); we selected this number of the studies to satisfy the sample size requirements for Cohen’s Kappa calculation [43]. We then assessed their inter-rater agreement (Cohen‘s Kappa = 0.9) [44].

Due to the conservative searching, the majority of the studies were found to be unrelated to the scope of the survey. We excluded 1,225 publications that did not meet the inclusion criteria. Subsequently, we complemented our search process with two other strategies to find relevant papers that could have been missed in the initial search. We performed a manual *issue-by-issue* search of major conference proceedings and journals in software engineering in the period from January 2010 to December 2020. The searched journal and proceedings are listed in Table 2.2. That step produced another 14 unique publications. Finally,

Table 2.1: Inclusion and exclusion criteria.

No.	Inclusion Criteria
1	Primary studies related to software engineering and may have actionable consequences for engineers or researchers
2	Peer-reviewed studies published as conference, journal, or workshops papers or a book chapter
3	Studies related to the use of app reviews in support to at least one software engineering activity (directly or indirectly) [41]
No.	**Exclusion Criteria**
1	Papers not written in English
2	Papers analyzing app reviews without the purpose to support software engineering
3	Secondary or tertiary studies (e.g., systematic literature reviews, surveys, etc.) technical reports or manuals

Table 2.2: Selected conference proceedings and journals for manual search.

Venue	Abbr.
International Conference on Software Engineering	ICSE
European Software Engineering Conference and Symposium on the Foundations of Software Engineering	ESEC/FSE
International Conference on Automated Software Engineering	ASE
International Conference on Software Maintenance and Evolution	ICSM/ICSME
Conference on Advanced Information Systems Engineering	CAiSE
International Requirements Engineering Conference	RE
IEEE Transactions on Software Engineering	TSE
ACM Transactions on Software Engineering and Methodology	TOSEM
IEEE Software	IEEE SW
Empirical Software Engineering	EMSE
Information and Software Technology	IST
Requirements Engineering Journal	REJ

we completed the searching with a *snowballing* procedure following guidelines proposed by Wohlin [45]. We performed backward snowballing considering all the references from relevant studies found by previous searching strategies. Moreover, we conducted forward snowballing based on the 10 most cited papers. Using snowballing procedure, additional 40 relevant articles were found to match our inclusion criteria. We used these criteria to screen the papers based on the title, abstract and full-text (if needed). Accordingly, we ended up with 182 articles included in the survey.

The selected literature was narrowed to peer-reviewed studies only; in principle, in the peer-review process, one or more experts in the field perform rigorous evaluation a study to ensure the study maintains sufficient quality standards and is suitable for the publication [46]. We therefore did not conduct a further quality assessment of individual studies [40].

2.2.3 Data Extraction

The first author created a data extraction form to collect detailed contents for each of the selected studies. They used extracted data items to synthesize information from primary studies and answer research questions RQ1–RQ5. Table 2.3 presents the data items the first author extracted:

Table 2.3: Data Extraction Form.

Item ID	**Field**	**Use**
F1	Title	Documentation
F2	Author(s)	Documentation
F3	Year	Documentation
F4	Venue	Documentation
F5	Citation	Documentation
F6	Review Analysis	RQ1
F7	Mining Technique	RQ2
F8	Software Engineering Activity	RQ3
F9	Justification	RQ3
F10	Evaluation Objective	RQ4
F11	Evaluation Procedure	RQ4
F12	Evaluation Metrics and Criteria	RQ4
F13	Evaluation Result	RQ5
F14	Annotated Dataset	RQ4
F15	Annotation Task	RQ4
F16	Number of Annotators	RQ4
F17	Quality Measure	RQ4
F18	Replication Package	RQ4

- *Title, Author(s), Year, Venue, Citation* (F1–F5) are used to identify the paper and its bibliographic information. For F5, we record the citation count for each paper according to Google Scholar (as of the 4th of August 2021).

- *Review Analysis* (F6) records the type of app review analysis (F6.1) (e.g. review classification), mined information (F6.2) (e.g. bug report) and supplementary description (F6.3).

- *Technique* (F7) records what techniques are used to realize the analysis. We recorded the technique type (F7.1) e.g., machine learning and its name (7.2) e.g., Naïve Bayes.

- *Software Engineering Activity* (F8) records the specific software engineering activity (e.g. requirements elicitation) mentioned in the paper as being supported by the proposed app review analysis method. We used widely known taxonomy of software engineering phases and activities to identify and record these items [41].

- *Justification* (F9) records the paper's explanation for how the app review analysis support the software engineering activities. Some papers do not provide any justification.
- *Evaluation Objective* (F10) records the general objective of the paper's evaluation section (F10.1) (e.g. quantitative effectiveness, or user-perceived usefulness) and the type of evaluated app review analysis (F10.2).
- *Evaluation Procedure* (F11) records the paper's evaluation method and detailed evaluation steps.
- *Evaluation Metrics and Criteria* (F12) records the quantitative metrics (e.g. precision and recall) and criteria (e.g. usability) used in the evaluation.
- *Evaluation Result* (F13) records the result of an empirical evaluation with respect to the evaluation metrics and criteria.
- *Annotated Dataset* (F14) records information about the datasets used in the study. We stored information about App Store name from which reviews were collected (F14.1) e.g., Google Play, and the number of annotated reviews (F14.2).
- *Annotation Task* (F15) records the task that humans annotators performed when labeling a sample of app reviews e.g., classify reviews by discussed issue types.
- *Number of Annotators* (F16) records the number of human annotators labeling app reviews for an empirical evaluation.
- *Quality Measure* (F17) is the measure used for assessing reliability of the annotated dataset e.g., Cohen's Kappa.
- *Replication Package* (F18) records whether a replication package is available. When one is available, we also recorded details about its content such as the availability of an annotated dataset, analysis method implementation, and experiment's scripts. In addition to the reported information; we contacted the authors of primary studies to check the availability of the replication packages.

The reliability of data extraction was evaluated through inter- and intra-rater agreements [47]. The agreements were measured using percentage agreement on a recommended sample size [43, 48]. To evaluate intra-rater agreement, the first author re-extracted data items from a random sample of 20% of selected papers. An external assessor[2] then validated the

[2]The assessor has an engineering background and experience with manual annotation; they has no relationship with this research.

extraction results between the first and second rounds; and computed percentage agreement. To evaluate inter-rater agreement, the assessor cross-checked data extraction; the assessor independently extracted data items from a new random sample of 10% of selected papers. The first author and the assessor then compared their results and computed agreement. The intra-rater agreement was at the level of 93% whereas the inter-rater agreement was of 90%, indicating nearly the complete agreement [47].

2.2.4 Data Synthesis

Most data in our review is grounded in qualitative research. As found by other researchers, tabulating the data is useful for aggregation, comparison, and synthesis of information [40]. The data was thus stored in the spreadsheets, manually reviewed, and interpreted to answer research questions. Parts of the extracted data we synthesized using descriptive statistics.

We also used three classification schemas to group collected information on app review analysis (F6), mining techniques (F7) and SE activity (F8). We constructed each schema following the same general procedure based on the content analysis method [49]; the first author initially examined all the collected information of a specific data item type; then performed an iterative coding process. During the coding, each information was labeled with one of the categories identified in the literature or inferred from the collected data.

To create the schema of app review analyses, we adopted 5 categories proposed in the previous survey [1]. As the categories were not exhaustive for the coding; we extended them with 14 additional categories: 7 categories from the taxonomy of mining tasks [50], and 7 standard types of text analytics [51]; we referred to data and text mining areas as they have well defined terminology for text analysis. We then merged semantically-related categories; and removed those unrelated to the domain of app review analysis. The resulting list of 8 categories we then extended by adding the Recommendation category abstracted from the remaining unlabelled data. With 9 categories, the first author performed the final coding. Table 2.7, in the corresponding result section, presents the nine types of app review analyses.

The classification schema of mining techniques is informed by categories in previous survey on intelligent mining techniques [22] and text analytics [51, 52, 53]. We first identified 5 categories of mining techniques: 4 categories proposed in the previous survey [22]; and 1 category identified from text analytics i.e., statistical analysis [51, 52, 53]. While coding, we however excluded feature extraction category referring to an instance of general information extraction task rather than a type of technique [51]; and performed the final coding using the remaining 4 categories. The resulting mining techniques categories can be found in Table 2.9.

We derived the schema of SE activities based on the terminology from the software engineering body of knowledge [41]; we first identified 258 terms related to the main software engineering concepts; and then selected 58 terms describing candidate activities for the coding process. While coding, we excluded 44 terms as they did not match any data items; and performed the final coding using the remaining 14 terms (from now SE activities). Table 2.13 lists the the resulting software engineering activities in the corresponding result section.

We validated the coding reliability of each schema using inter- and intra- rater agreement. We measured the reliability using percentage agreement on a recommended sample size [48, 43]. To evaluate intra-rater agreement, the first authors re-coded a random sample of 20% of selected papers. The external assessor then checked the coding between the first and second coding. To evaluate inter-rater agreement, both the first author and the assessor coded a new random sample of 10% of the papers. They then cross-checked their results. The percentage intra- and inter-rater agreements were equal or above 90% and 80% for coding each schema, indicating their very good quality [47]; table 2.4 provides detail statistics for the reliability evaluation.

The spreadsheets resulting from our data extraction and data grouping can be found in the supplementary material of this survey [54].

The following section 2.3 provides answers to the research questions RQ1–RQ5. In answer to the questions, we give references to exemplary papers. The complete list of papers related to a specific research question and respective synthetized information is available in the supplementary spreadsheet [54].

Table 2.4: The intra- and inter-rater agreement for the classification schemas.

Classification Schema	**Intra-Rater Agreement**	**Inter-Rater Agreement**
App Review Analysis	93%	87%
Software Engineering Task	100%	87%
Mining Technique	90%	80%

2.3 Result Analysis

2.3.1 Demographics

Figure 2.2 shows the number of primary studies per year, including breakdown of a publication type (Journal, Conference, Workshop, and Book). The publication date of primary studies ranges from 2012 to 2020; no study was published in 2010 and 2011. We observed that 53%

of the primary studies were published in the last 3 years. The increase in the quantity of the publications however could not have been compared to the general increase in the quantity of scientific publications; the value is not publicly available, and it is difficult to estimate; it can depend on many factors like publication periods, research fields, or specific research topics.

The increased number of the publications analyzing app reviews to support software engineering however suggests a growing interest in this research area; the publication count quantifies the concrete amount of research works that have been conducted and published in peer-reviewed venues.

Figure 2.3 shows the distribution by a venue type: 65% of papers are published in conferences, 23% in journals, 10% in workshops and 2% as book chapters. Table 2.5 lists the top ten major venues in terms of the number of published papers[3]. The venues include the main conferences and journals in the software engineering community. Table 2.6 lists twenty most cited papers in the field of app review analysis for software engineering; and summarizes their contributions. These studies advanced the field in substantial ways, or introduced influential ideas.

Key insights from demographics

- The interest in the research on app review analysis has rose substantially since the first published papers in 2012. Papers published in the last 3 years account for over 50% of all the publications in this research area. The general increase of scientific publications however was not controlled to compare the two values.
- The main venues publishing research in app review analysis include the main general software engineering conferences and journals (ICSE, FSE, ASE, IEEE Software) as well as the main specialized venues in empirical software engineering (ESEM) and requirements engineering (RE, REFSQ).

2.3.2 RQ1: App Review Analysis

In this section, we answer RQ1 (what are the different types of app review analysis) based on data extracted in F6 (review analyses). To answer the question, we grouped data items into one of nine general categories, each representing a different review analysis type (F6.1). We performed the grouping following the classification schema we had constructed for this study (see Sect. 2.2.4); and categories previously proposed in the context of app store analysis [1] as well as data and text mining [50, 51]. Here, we focused on an abstract representation, because primary studies sometimes use slightly different terms to refer to the same type of

[3]The complete list of venues can be found in supplementary material [54].

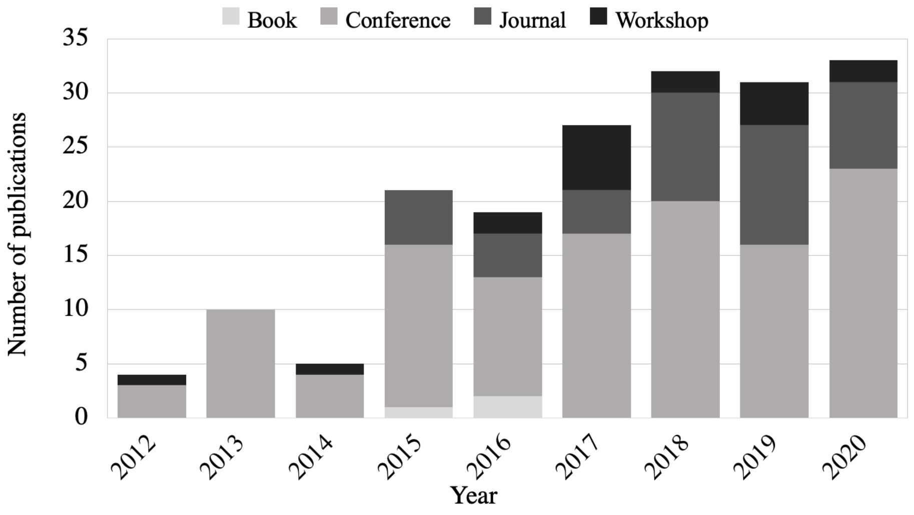


Figure 2.2: Number of publications per year. The first papers on app review analysis were published in 2012.

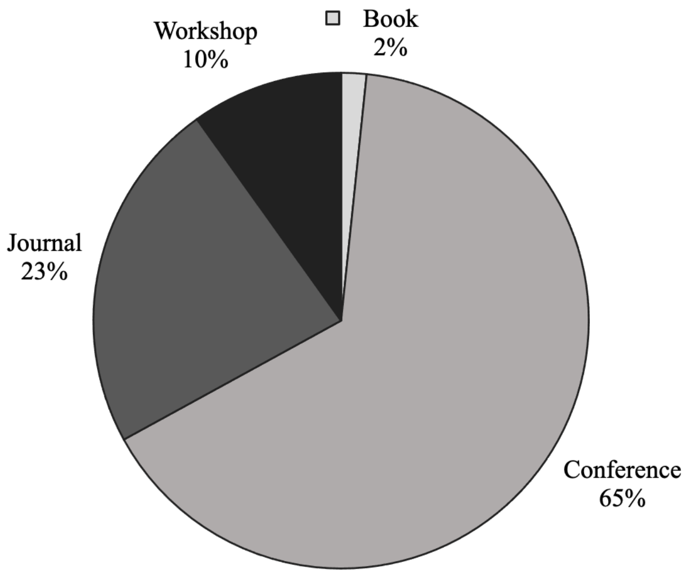


Figure 2.3: Pie chart showing the distribution of research papers per a venue type in the period from 2010 to December 31, 2020.

analysis. Table 2.7 lists the different types of app review analyses and their prevalence in the literature.

2.3.2.1 Information Extraction

App reviews is unstructured text. Manually extracting relevant information from large volume of reviews is not cost-effective [64]. To address the problem, 56 of the primary studies (31%) proposed approaches facilitating information extraction. Formally, information extraction is the task of extracting specific (pre-specified) information from the content of a review; this information may concern app features (functional attributes) [4, 8, 9], qualities (non-functional attributes) [5, 68], problem reports and/or new feature requests (e.g., [17, 35, 58, 69]), opinions

Table 2.5: Top ten venues publishing papers on app review analysis between 2010 and 2020.

Venues	No. Studies
International Requirements Engineering Conference (RE)	11
Empirical Software Engineering Journal (EMSE)	10
International Working Conference on Requirements Engineering (REFSQ)	7
International Conference on Software Engineering (ICSE)	7
IEEE Software (IEEE Softw)	6
International Symposium on Foundations of Software Engineering (FSE)	6
International Conference on Automated Software Engineering (ASE)	6
International Workshop on App Market Analytics (WAMA)	5
Intl. Conference on Mobile Software Engineering and Systems (MOBILESoft)	5
Intl. Conference on Evaluation and Assessment in Software Engineering (EASE)	5

Table 2.6: Twenty most influential papers in the field of app reviews analysis for software engineering, ordered by year of publication.

Reference	Contribution	Citat.
Vasa et al. [55]	Preformed the first preliminary analysis of mobile app reviews.	123
Galvis-Carreño et al. [56]	Proposed an approach extracting requirements from feedback.	329
Fu et al. [57]	Proposed WisCom system for analyzing millions of reviews.	415
Iacob et al. [58]	Developed a tool extracting and summarizing user requests.	334
Pagano and Maalej [2]	Studied the content and the usefulness of app reviews for RE.	514
Chen et al. [29]	Developed AR-Miner tool for filtering and prioritizing reviews.	480
Guzman and Maalej [8]	Proposed an approach for feature-based sentiment analysis.	531
Guzman et al. [59]	Proposed ensemble methods for app review classification.	101
Khalid et al. [60]	Studied user complains in reviews and their impact on ratings.	415
Maalej and Nabil [12]	Benchmarked techniques for automatically classifying reviews.	381
Martin et al. [61]	Studied the app sampling problem for app store mining.	121
Panichella et al. [62]	Taxonomy and an approach for identyfing users intentions.	352
Palomba et al. [63]	CRISTALS approach facilitating reviews-to-code traceability.	156
Gu and Kim [14]	Developed and evaluated SUR-Miner tool for opinion mining.	110
Vu et al. [64]	MARK framework searching and analyzing user opinions.	140
Di Sorbo et al. [65]	SURF tool summarizing users' needs and topics from reviews.	197
Maalej et al. [15]	Large-scale empirical study on classification techniques.	166
Maalej et al. [66]	Proposal for utilizing on-line user feedback to support RE.	209
McIlroy et al. [67]	Automatically analyzed the types of user issues in reviews.	126
Villarroel et al. [18]	Automatic approach for release planning by review analysis.	205

about favored or unfavored features (e.g., [8, 14, 64, 70]) as well as user stories [71]. Relevant information can be found at any location in the reviews. For instance, a problematic feature can be discussed in a middle of a sentence [8, 72], or a requested improvement can be expressed anywhere in a review [71, 73].

Table 2.7: App review analysis types and their prevalence in the literature.

App Review Analysis	No. Studies	Percentage
Information Extraction	56	31%
Classification	105	58%
Clustering	44	24%
Search and Information Retrieval	24	13%
Sentiment Analysis	40	22%
Content Analysis	54	30%
Recommendation	30	16%
Summarization	25	14%
Visualization	20	11%

2.3.2.2 Classification

Classification consists of assigning predefined categories to reviews or textual snippets (e.g., sentences or phrases). Classification is by far the most common type of app review analysis found in the literature: 58% of publications describe techniques for classifying reviews. Classification can be used to separate informative reviews from those that are uninformative (e.g., [29, 65, 74, 75]), spam [76] or fake [77]. Informative reviews can be subsequently classified to detect user intentions (e.g., [15, 78]) and discussion topics (e.g., [79, 80]). User intentions include reporting an issue or requesting a new feature (e.g., [62, 81, 82]).

Discussion topics include a variety of concerns such as installation problems, user interface, or price [72, 83, 84]; topics concerning user perception e.g., rating, user experience or praise [2, 85]; or topics reporting different types of issues [67, 86, 87]. For instance, review classification has been proposed to detect different types of usability and user experience issues [10, 88], quality concerns [89, 90] or different types of security and privacy issues [86, 91]. Similarly, app store feedback can be classified by their reported requirements type [90, 92]; this could help distinguish reviews reporting functional requirements from those reporting non-functional requirements [68, 93, 94]; distilling non-functional requirements into fine-grained quality categories such as reliability, performance, or efficiency [95, 96]. Another key use of the classification task is rationale mining; it involves detecting types of argumentations and justification users describe in reviews when making certain decisions, e.g. about upgrading, installing, or switching apps [97, 98, 99].

2.3.2.3 Clustering

Clustering consists of organizing reviews, sentences, and/or snippets into groups (called a cluster) whose members share some similarity. Members in the same group are more similar (in some sense) to each other than to those in other groups. Unlike classification, clustering does not have predefined categories. Clustering is thus widely used as an exploratory analysis technique to infer topics commonly discussed by users (e.g., [8, 100, 101]) and aggregate reviews containing semantically related information (e.g., [78, 102, 103]). Clustering can be used for grouping reviews that request the same feature [65, 104], report similar problems [16, 61, 72], or discuss a similar characteristic of the app [105, 106, 107]. The generated clusters might help software engineers synthesize information from a group of reviews referring to the same topics rather than examining each review individually (e.g., [57, 69, 73, 108]).

2.3.2.4 Search and Information Retrieval

Search and information retrieval concerns finding and tracing reviews (or their textual snippets) that match needed information. The task can be used to find reviews discussing a queried app feature [64, 109, 110], to obtain the most diverse user opinions in reviews [102], or to trace what features described in the app description are discussed by users [9, 111]. Information retrieval is also used to establish traceability links between app reviews and other software engineering artefacts [63, 112], such as source code [35, 78, 103], stack tracers [113], issues from tracking systems [63, 114], and warnings from static analysis tools [115] in order to locate problems in source code [103, 116, 117], suggest potential changes [63, 103], or to flag errors and bugs in an application under test [115]. Such traceability links can be also detected between reviews and feedback from other source like Twitter to study if the same issues are discussed in both digital channels [118, 119, 120]; or between reviews and goals in goal-model to understand the extent to which app satisfies the users' goals [121, 122].

Table 2.8 summarizes types of data that have been combined with app reviews using search and information retrieval; indicates the purpose of the analysis; and provides references to primary studies.

2.3.2.5 Sentiment Analysis

Sentiment analysis (also known as opinion mining) refers to the task of interpreting user emotions in app reviews. The task consists in detecting the sentiment polarity (i.e., positive, neutral, or negative) in a full review [124, 125, 126], in a sentence [8, 62, 81], or in a phrase [4, 14]. App reviews are a rich source of user opinions [8, 125, 127]. Mining these opinions involves identifying user sentiment about discussed topics [4, 14], features [8, 128] or software qualities [11, 88, 129]. These opinions can help software engineers to understand how users perceive their app [11, 14, 130], to discover users requirements [110, 131], preferences [88, 127, 132], and factors influencing sales and downloads of the app [133]. Not surprisingly, knowing user opinions is an important information need developers seek to

Table 2.8: Types of data that have been combined with app reviews using search and information retrieval.

Type of Data	Purpose
App Description	Use features from app descriptions to filter informative reviews [9]; to discover 'hot' features [9]; to understand users' preferences [111]; and to identify domain features [123].
Git Commit	Detect links between reviews and source code changes to analyze the impact of user feedback on the development process; to keep track on requests that have (not) been implemented [63, 112].
Goal Model	Detect links between reviews and goals in goal model; to identify users' satisfaction w.r.t. these goals; or to recommend new goals that need to be satisfied by the app [121, 122].
Issue Report	Detect links between reviews and issues to understand what reports have (not) be addressed [63, 112]; to identify issue report duplications; and to prioritize the issues [114].
Lint Warning	Recover the links between warnings from static analysis tools and app user reviews to support warning prioritization [115].
Source Code	Link reviews to source-code to locate components related to requested changes; to recommend software changes [78, 103]; to estimate the impact of the changes [116].
Stack Trace	Link reviews to stack trace to integrate user feedback into app testing [117]; to augment testing report with contextual information that can ease the understanding a failure [113].
Tweet	Link reviews to user feedback from Twitter [120]; to integrate the feedback from both channel; and to understand what different issues are discussed by app users [118, 119].

satisfy [134, 135].

2.3.2.6 Content Analysis

Content analysis studies the presence of given words, themes, or concepts within app reviews. For example, studies have analysed the relation between user ratings and the vocabulary and length of their reviews [55, 136]. Studies have shown that users discuss diverse topics in reviews [2, 35], such as app features, qualities [11, 137], requirements [92, 96] or issues [72, 87, 138]. For example, using content analysis, researchers analysed recurring types of issues reported by users [35, 67, 139], their distribution in reviews as well as relations between an app issue type and other information such as price and rating [140, 141] or between an issue type and code quality indicators [75]. Interestingly, studies have pointed out that users' perception for the same apps can vary per country [142], user gender [143], development framework [144], and app store [145]. Content analysis can be also beneficial for software engineers to understand whether cross-platform apps achieve consistency of users' perceptions across different app stores [146, 147], or whether hybrid development tools achieve their main purpose: delivering an app that is perceived similarly by users across platforms [147]. Finally, studying the dialogue between users and developers has shown evidence that the chances of users to update their rating for an app increase as a result of developer's response to reviews [148, 149].

2.3.2.7 Recommendation

Recommendation task aims to suggest course of action that software engineers should follow. Several mining approaches, for instance [29, 122], have been proposed to recommend reviews that software engineers should investigate. These approaches typically assign priorities to a group of comments reporting the same bug [16, 73, 150], requesting the same modification or improvement [78, 151, 152]. Such assigned priorities indicate relative importance of the information that these reviews convey from the users' perspective. Factors affecting the importance vary from e.g., the number of reviews in these groups [29, 78], to the influence of this feedback on app download [153], and the overall sentiment these comments convey [128, 154]. In line with this direction, mining approaches have been elaborated to recommend feature refinement plans for the next release [154, 155], to highlight static analysis warnings that developers should check [115], to recommend test cases triggering bugs [35], to indicate mobile devices that should be tested [156], and to suggest reviews that developers should reply [126, 157, 158]; the approaches can analogously recommend responses for these reviews [157, 158], stimulating users to upgrade their ratings or to revise feedback to be more positive [148, 159].

2.3.2.8 Summarization

Review summarization aims to provide a concise and precise summary of one or more reviews. Review summarisation can be performed based on common topics, user intentions, and user sentiment for each topic (e.g., [8, 84, 121]). For example, Di Sorbo et al. proposed summarizing thousands of app reviews by an interactive report that suggests to software engineers what maintenance tasks need to be performed (e.g., bug fixing or feature enhancement) with respect to specific topics discussed in reviews (e.g., UI improvements) [65, 79]. Other review summarization techniques give developers a quick overview about users' perception specific to core features of their apps [8, 58, 107], software qualities [84], and/or main users' concerns [13, 116, 160]. With the addition of statistics e.g., the number of reviews discussing each topic or requesting specific changes, such a summary can help developers to prioritize their work by focusing on the most important modifications [116]. In addition, such a summary can be exported to other software management tools e.g., GitHub, JIRA [13] to generate new issue tickets and help in problem resolution [161].

2.3.2.9 Visualization

Visualization can aid developers in identifying patterns, trends and outliers, making it easier to interpret information mined from reviews [100, 121]. To communicate information clearly and efficiently, review visualization uses tables, charts, and other graphical representations [15, 100], accompanied by numerical data [15, 88]. For example, Maalej et al. demonstrated that trend analysis of a review type (e.g., bug report, feature request, user experience) over time can be used by software engineers as an overall indicator of the project's health [15]. Other studies proposed visualizing dynamics of main themes discussed in reviews to identify emerging issues (e.g., [16, 17, 73, 162]), or to show the issue distribution for an app across different app stores [150]. Simple statistics about these issue (e.g., 'How many reviews reported specific issues?') can give an overall idea about the main problems, in particular if compared against other apps (e.g., 'Do users complain more about security of my app compared to similar apps?). Similarly, analyzing the evolution of user opinions and bug reports about specific features can help software engineers to monitor the health of these features and to prioritize maintenance tasks (e.g., [64, 88, 105, 163]). For instance, software engineers can analyse how often negative opinions emerge, for how long these opinions have been reported, and whether their frequency is rising or declining [14, 64, 86]. This information could provide developers with evidence of the relative importance of these opinions from the users' perspective [88, 110].

RQ1: App Review Analysis

- 9 broad types of review analyses have been identified in the literature: (1) information extraction; (2) classification; (3) clustering; (4) search and information retrieval; (5) sentiment analysis; (6) content analysis; (7) recommendation; (8) summarization and (9) visualization.
- Reviews classification, clustering, and information extraction are the mostly frequently applied automatic tasks; they help to group reviews, discover hidden patterns and to focus on relevant parts of reviews.
- Content analysis is used to characterize reviews, to identify discussed topics, and to explore information needs that can be satisfied by the feedback.
- Searching and information retrieval aids software engineers to query reviews with information of their interest, and to trace it over other software artefacts (e.g., stack traces, issue tracking system or goal-models) or other sources of on-line

user feedback (e.g. tweets).

- Summarizing and visualizing information scattered across a large number of reviews can aid developers in interpreting the information that could be costly and time-consuming to undertake if done manually.
- Mined information is commonly used to recommend engineers a course of their maintenances actions e.g., bugs in need of urgent intervention, or localizing the problem in the source code.

2.3.3 RQ2: Mining Techniques

App review analyses (see Section 2.3.2) are realized using different text mining techniques. In this section, we address RQ2 (what techniques are used to realize app review analysis) based on extracted data in F7 (mining technique) that we grouped following the classification schema we had constructed for this study (see Sect. 2.2.4). The categories of this schema come from the survey on intelligent mining techniques and tools [22] and text analytics area [51, 52, 53].

In answer to this question, we identified 4 broad categories of mining techniques: manual analysis (MA), natural language processing (NLP), machine learning (ML) and statistical analysis (SA). Table 2.9 lists the techniques and their prevalence in the literature. It can be observed more than a half of studies employed NLP or ML; whereas MA and SA were present in 25% and 29% of the studies. Table 2.10 reports how many studies used a certain technique to realize a given type of app review analysis. We observe that the NLP or ML are dominant for realizing app review analyses, except for Content Analysis that is mostly performed using MA or SA technique. A single study frequently used the same type of technique for realizing several app review analyses (e.g., Clustering, Classification); the number of studies, in the furthest right column, is thus less or equal than the sum of a row. On the other hand, we also recorded studies frequently combined the techniques together to perform a single app review analysis. Table 2.11 reports what combinations of techniques were used in the literature and how many studies used each combination for realizing a specific app review analysis. A single study could use a certain combination of techniques to facilitate multiple review analyses; the total number of studies, on the right hand side, is thus less than the sum of a row. The results indicate NLP and ML were mostly combined for Classification; MA and SA were used together for Content Analysis; whereas NLP and SA was adopted for Information Extraction. The following sections discuss each type of technique.

Table 2.9: Mining techniques and their prevalence in the literature.

Mining Techniques	No. Studies	Percentage
Manual Analysis	45	25%
Natural Language Processing	113	62%
Machine Learning	108	59%
Statistical Analysis	53	29%

Table 2.10: How often primary studies used certain mining techniques to realise a type of app review analysis.

		App Review Analysis								Studies	
		Information Extraction	Classification	Clustering	Search and Info. Retrieval	Sentiment Analysis	Content Analysis	Recommendation	Summarization	**Number**	**Percentage**
Technique	Manual Analysis	1	11	1	0	2	37	0	0	45	25%
	Natural Language Processing	36	55	13	24	19	5	9	10	113	62%
	Machine Learning	10	73	36	2	7	3	13	2	108	59%
	Statistical Analysis	9	2	1	1	0	23	11	12	53	29%

2.3.3.1 Manual Analysis

Scholars have shown an interest in manual analysis of app reviews [80, 98]. The technique is used to facilitate Content Analysis e.g., to understand topics users discuss [2, 11, 72] and to develop a ground truth dataset for training and evaluating mining techniques [4, 97]. Manual analysis typically takes a form of tagging a group of sample reviews with one or more meaningful tags (representing certain concepts). For example, tags might indicate types of user complaint [60, 139], feature discussed in reviews [4, 12], or sentiment users expresses [164]. To make replicable and valid inferences upon manual analysis, studies perform it in a systematic manner. Figure 2.4 illustrates the overall procedure of manual analysis. Scholars first formulate the analysis objective corresponding to the exploration of review content (e.g., understanding types of user complaints) or the development of ground truth (e.g., labelling types of user feedback). They then select the reviews to be analysed, and specify the unit of analysis (e.g., a review or a sentence). Next, one or more humans (called 'coders') follow a coding process to systematically annotate the reviews. A coder examines a sample of reviews and tags them with specific concepts. Unless these concepts are known in advance or coders

Table 2.11: How often primary studies used certain combination of techniques to realise a type of app review analysis; MA stands for manual analysis; NLP denotes natural language processing; ML marks machine learning; and SA signifies statistical analysis.

		App Review Analysis								Studies	
		Information Extraction	Classification	Clustering	Search and Info. Retrieval	Sentiment Analysis	Content Analysis	Recommendation	Summarization	**Number**	**Percentage**
Combinations of Techniques	MA	1	11	1	0	2	25	0	0	31	17%
	NLP	32	15	4	22	13	1	5	7	67	37%
	ML	2	32	28	0	1	2	8	2	62	34%
	SA	0	0	0	0	0	11	10	9	30	16%
	MA + ML	0	0	0	0	0	1	0	0	1	1%
	MA + NLP	0	0	0	0	0	1	0	0	1	1%
	MA + SA	0	1	0	0	0	9	0	0	9	5%
	ML + SA	0	0	0	0	0	0	1	0	2	1%
	NLP + ML	8	39	8	2	6	0	4	0	53	29%
	NLP + SA	9	0	1	0	0	2	0	3	15	8%
	MA + NLP + SA	0	0	0	0	0	1	0	0	1	1%
	NLP + ML + SA	0	1	0	1	0	0	0	0	2	1%

agree about the tagging, the step is iterative; When, for example, new concepts are identified, coders examine once again all the previously tagged reviews and check if they should be also tagged with the new concepts. Such iterations minimize the threat of human error when tagging the reviews. Once all the reviews are tagged, authors either analyse findings or use the dataset to evaluate other mining techniques [4, 72, 165].

Manual analysis is time-consuming and require vast human effort [2, 8, 80]; a pilot study typically proceeds an actual analysis [4, 97, 164]; subsequently the actual tagging, focusing on a statistically representative sample of reviews, takes places [60]. For example, Guzman and Maalej involved seven coders who independently tagged 2800 randomly sampled user reviews [8]. For each review, two coders independently tagged the type of user feedback, features mentioned in the review and sentiments associated to these features. The study reports that coders spent between 8 and 12.5 hours for coding around 900 reviews.

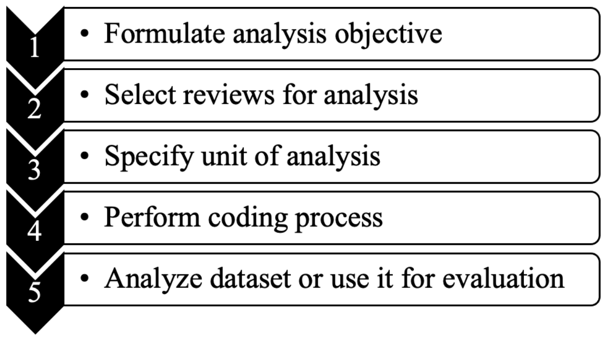

1
• Formulate analysis objective
2
• Select reviews for analysis
3
• Specify unit of analysis
4
• Perform coding process
5
• Analyze dataset or use it for evaluation

Figure 2.4: Figure showing the overall process of manual analysis.

2.3.3.2 Natural Language Processing

User-generated content of app reviews takes the form of text [55, 136]. Such text has plenty of linguistic structure intended for human consumption rather than for computers [166]. The content must, therefore, undergo a good amount of natural language processing (NLP) before it can be used [166, 167]. Given this fact, it is not surprising that the majority of primary studies (62% of surveyed papers) adopt NLP techniques to support review analysis (see Section 2.3.2). At a high level, pre-processing can be simply seen as turning review content into a form that is analysable for a specific mining task (see Section 2.3.2). There are different ways to pre-process reviews including text normalization, cleaning and augmenting [62, 122, 167]. These pre-processing steps typically involve converting texts into lowercase [57, 108, 164], breaking up a text into individual sentences [78, 95, 168], separating out words i.e., tokenization [13, 26, 103], spelling correction [103, 117] as well as turning words into their base forms e.g., stemming or lemmatization [12, 62, 169]. Of course, not all the review content is meaningful [8, 29, 120]. Some parts are noisy and obstruct text analysis [63, 103, 128]. The content is thus cleaned by removing punctuation [147, 170], filtering out noisy words like stop words [9, 116, 128], or non-English words [63, 165]. Such normalized and cleaned text tends to be augmented with additional information based on linguistic analysis e.g., part-of-speech tagging (PoS) [128, 155, 170] or dependency parsing [14, 101, 171].

A review can be modelled as a words sequence [9], bag-of-words (BoW) [12] or in vector space model (VSM) [64] to sereve as an input for other mining techniques. In particular, primary studies refer to NLP techniques comparing text similarity [92, 109], pattern matching [5, 9, 171] and collocations finding (e.g., [8, 107, 111, 131]).

Text similarity techniques (employed in 21 studies) determine how "close" two textual snippets (e.g., review sentences) are [167]. These snippets, represented in VSM or BoW, are compared using similarity measure like Cosine similarity [35, 64], Dice similarity coefficient [63, 78] or Jaccard index [13]. These techniques support Searching and Information Retrieval e.g., to link reviews with issue reports from issue tracking systems [114], Recommendation e.g., to recommend review responses based on old ones that have been posted to similar reviews [157], Clustering e.g., to group semantically similar user opinions [105, 172], and Content Analysis e.g., to compare review content [144].

Pattern matching techniques (employed in 22 studies) localize parts of review text (or its linguistic analysis) matching hand-crafted patterns. Such patterns can take many forms, such as, regular expressions [5, 93, 173], PoS sequences [9, 105], dependencies between words [79, 104, 126] or simple keyword matching [15, 79, 86]. The techniques have been adopted in Information Extraction e.g., to extract requirements from reviews [5, 93], Classification e.g., to classify requirements into functional and non-functional [93] and Summarization e.g., to provide a bug report summary [5].

Collocation finding techniques are employed for Information Extraction e.g., to extract features [8, 169] or issues [16] from reviews. Such collocations are phrases consisting of two or more words, where these words appear side-by-side in a given context more commonly than the word parts appear separately [166]. The two most common types of collocation detected in the primary studies are bigrams i.e., two adjacent words [8, 131]. Co-occurrences may be insufficient as phrases such as all the may co-occur frequently but are not meaningful. Hence, primary studies explore several methods to filter out the most meaningful collocations, such as Pointwise Mutual Information (PMI) [16, 172] and hypothesis testing [4, 8, 166].

2.3.3.3 Machine Learning

Table 2.12: Distribution of machine learning techniques used in primary studies in the period form 2010 to December 31, 2020.

Type	**Machine Learning Techniques**	**No. Studies**	**Percentage**
Supervised	Naïve Bayes	43	24%
	Support Vector Machine	39	21%
	Decision Tree	31	17%
	Logistic Regression	23	13%
	Random Forest	20	1%
	Neural Network	12	7%
	Linear Regression	7	4%
	K-Nearest Neighbor	4	2%
Unsupervised	Latent Dirichlet Allocation	36	20%
	K-Means	8	4%

Overall, 108 of 182 primary studies (59%) reported the use of machine learning (ML) techniques to facilitate mining tasks and review analysis. Table 2.12 reports ten most commonly applied ML techniques. Most of them (i.e., 8 techniques) are supervised, whereas 2 of them are unsupervised [174]. The widespread interest in ML techniques may be attributed to the fact that Clustering e.g., to group reviews discussing the same topics [57, 82] and Classification e.g., to categorize user feedback based on user intention [78, 175], among the most common review analysis types (see Table 2.7), are mainly facilitated using ML. When looking at the whole spectrum of review analysis these ML techniques support, we have also recorded their use for Sentiment Analysis e.g., to identify feature-specific sentiment [14], Recommendation e.g., to assign priorities to reviews reporting bugs [18] and Information Extraction e.g., to identify features [30, 68].

Scholars experimented with many textual and non-textual review properties[4] to make ML techniques work best [12, 59]. Choosing informative and independent properties is a crucial step to make these techniques effective [15, 174]. Textual properties, for example, concern: text length, tense of text [97, 98], importance of words e.g., td-idf [72, 95], a word sequence e.g., n-gram [12, 26] as well as linguistic analysis e.g., dependency relationship [176]. These properties are commonly combined with non-textual properties like user sentiment [15, 177], review rating [97] or app category [158]. We found that primary studies experiment with different properties [15, 26, 98].

2.3.3.4 Statistical Analysis

Statistical analysis is used in many papers to report research results [61, 75, 164], demonstrate their significance [55, 178], and draw conclusions of a large population of reviews by analysing their tiny portion [2, 89, 139]. We observed an interest in use of descriptive and inferential techniques for Content Analysis (e.g., [2, 89, 139, 179]). Summary statistics, box plots, and cumulative distribution charts help to gain understanding of review characteristics like their vocabulary size [55, 136], issue type distribution [67, 72, 146], or topics these reviews convey [2, 180]. Scholars employ different statistical tests to test check their hypothesis [11, 143, 178], to examine relationship between reviews characteristics [75, 143, 180], and to study how sampling bias affects the validity of research results [61].

Guzman et al., for example, conducted an exploratory study investigating 919 reviews from eight countries [143, 179]. They studied how reviews written by male and female users differ in terms of content, sentiment, rating, timing, and length. The authors employed Chi-square (e.g., content) and Mann-Whitney (e.g., rating) non-parametric tests for nominal and ordinal variables respectively [143]. Srisopha and Alfayez studied whether a relationship exists

[4]We refer to a property as a concept denoting a feature in the machine learning domain.

between user satisfaction and the applications internal quality characteristics [180]. Having employed Pearson correlation coefficient, the authors studied to what extent do warnings reported by static code analysis tools correlate with different types of user feedback and the average user ratings. Similarly, another study employed the Mann-Whitney test to examine if densities of such warnings differ between apps with high and low ratings [178].

RQ2: Mining Techniques

- Primary studies employ 4 broad types of techniques to realize app review analyses: (1) manual analysis; (2) natural language processing; (3) machine learning and (4) statistical analysis.
- Manual analysis is used to study review content; and to develop datasets for training/evaluating data mining techniques. The technique is time-consuming and requires substantial human effort.
- NLP techniques play an important role for review analysis. The majority of primary studies (62%) use the techniques for a wide spectrum of review analyses: Search and Information Retrieval, Classification, Clustering, Content Analysis, Information Extraction, Summarization or Recommendation.
- ML is employed by ca. 59% of papers for Clustering, Classification, Sentiment Analysis, Recommendation, or Information Extraction. Scholars experiment with textual and non-textual review properties to boost the effectiveness of the techniques.
- Statistical analysis is used to support Content Analysis: to summarize findings; to draw statistically significant conclusions; or to check their validity.

2.3.4 RQ3: Supporting Software Engineering

To answer RQ3 (what software engineering activities might be supported by analysing app reviews), we used data extracted in F8 (software engineering activity) and F9 (justification) as well as the classification schema of SE activities derived from the software engineering body of knowledge (see Sect. 2.2.4). Table 2.13 provides mapping between primary studies and SE activities that the studies claim to support; it also reports the number and the percentage of the studies per each activity. It is worth noting that some papers fall into more than one category i.e., claim to support more than one activity. In such case, we assigned the study to all the claimed activities. We can observe that primary studies motivated their approaches to support activities across different software engineering phases, including requirements (36%), maintenance (36%), testing (15%) and design (4%); 14 SE activities are supported in total; mostly research effort is focused on requirements elicitation (26%), requirements prioritization (10%), validation by users (11%), problem and modification analysis (23%), and requested modification prioritization (11%). We also recorded that 62 studies (34%) did not specify any SE activity that their approaches support.

To support the SE activities, primary studies used 9 broad types of app review analysis we identified with answer to RQ1 (see Sect. 2.3.2). Table 2.14 shows how often a type of app review analysis was used for a SE activity; papers that did not specify any SE activity are excluded; in case of papers supporting multiple SE activities, we assigned their facilitated analyses to all the claimed activities. It can be observed that each SE activity was supported using multiple analyses; classification was the most commonly used one; this was also the only analysis motivated for all the activities. A further result analysis revealed studies used the analyses in combination to mine useful information and support SE activities; we recorded 53 unique combinations; each composed of 1 to 5 types of analysis with the median of 2. Table 2.15 lists combinations used at least in 2 primary studies. The following sections provide a through synthesis on how mining useful information from app reviews might support SE activities.

2.3.4.1 Requirements

Requirements engineering includes involving system users, obtaining their feedback and agreeing on the purpose of a software to be built [66]. It therefore is not surprising that review analysis has received much attention to support requirements engineering activities, including requirements elicitation, requirements classification, requirements prioritization and requirements specification (see Table 2.13).

Requirements Elicitation. In app reviews, users give feedback describing their experience with apps, expressing their satisfaction with software products and raising needs for improvements [2, 7]. Software engineers can make use of the reviews to elicit new requirements [4, 7, 110, 131]. For instance, they can employ opinion mining approaches to examine reviews talking negatively about app features [4, 8, 111, 183]. This can help developers to understand user concerns about problematic features, and potentially help eliciting new requirements [4, 9, 110, 131]. Additionally, searching and retrieving users reviews that refer to a specific feature they are responsible for will allow them to quickly identify what users have been saying about their feature [110, 111, 123]. In line with this direction, approaches have been proposed to classify reviews by their user intention (e.g., reviewer requesting a

Table 2.13: Software engineering activities supported by app review analysis.

SE Activity	No. Studies	Percentage	Reference
REQUIREMENTS	66	36%	
Requirements Elicitation	55	30%	[2, 4, 5, 7, 8, 9, 12, 13, 15, 18, 56, 58, 65, 66, 68, 71, 74, 79, 80, 82, 84, 85, 90, 93, 95, 96, 99, 104, 106, 107, 108, 110, 111, 121, 122, 123, 131, 137, 151, 160, 162, 163, 171, 173, 175, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190]
Requirements Classification	10	5%	[5, 68, 80, 90, 92, 93, 94, 95, 96, 187]
Requirements Prioritization	19	10%	[2, 4, 5, 7, 8, 9, 18, 55, 59, 66, 75, 82, 97, 98, 120, 136, 151, 155, 163]
Requirements Specification	6	3%	[2, 15, 66, 72, 97, 98]
DESIGN	8	4%	
Design Rationale Capture	5	3%	[5, 97, 98, 99, 187]
User Interface Design	3	2%	[10, 11, 190]
TESTING	28	15%	
Validation by Users	20	11%	[4, 7, 8, 12, 14, 15, 17, 58, 57, 84, 88, 100, 101, 107, 140, 150, 160, 163, 169, 191]
Test Documentation	3	2%	[13, 113, 117]
Test Design	4	2%	[5, 35, 66, 150]
Test Prioritization	3	2%	[60, 150, 156]
MAINTENANCE	66	36%	
Problem and Mods. Analysis	46	25%	[7, 9, 13, 16, 17, 26, 57, 59, 60, 62, 63, 65, 69, 71, 73, 78, 85, 86, 87, 88, 90, 91, 100, 103, 105, 108, 112, 113, 115, 127, 131, 139, 153, 154, 161, 162, 163, 169, 172, 173, 186, 189, 192, 193, 194, 195]
Requested Mods. Prioritization	18	10%	[13, 14, 18, 60, 73, 75, 87, 110, 114, 115, 120, 147, 150, 151, 152, 154, 195, 196]
Help Desk	7	4%	[126, 148, 149, 157, 158, 159, 177]
Impact Analysis	5	3%	[63, 78, 103, 112, 116]
NOT SPECIFIED	62	34%	[27, 29, 30, 31, 61, 64, 67, 70, 76, 77, 81, 83, 89, 102, 109, 118, 119, 124, 125, 128, 129, 130, 132, 133, 138, 141, 142, 143, 144, 145, 146, 164, 165, 168, 170, 176, 178, 179, 180, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219]

Table 2.14: How often a type of app review analysis are used to realise a SE activity.

		Software Engineering Activity														Studies	
		REQUIREMENTS				DESIGN		TESTING				MAINTENANCE					
		Requirements Elicitation	Requirements Classification	Requirements Prioritization	Requirements Specification	Design Rationale Capture	User Interface Design	Validation by Users	Test Documentation	Test Design	Test Prioritization	Problem and Mods. Analysis	Requested Mods. Prioritization	Help Desk	Impact Analysis	**Number**	**Percentage**
App Review Analysis	Information Extraction	25	4	6	1	1	0	13	1	3	1	20	5	0	0	56	31%
	Classification	32	9	7	5	4	1	9	2	1	1	27	11	4	3	105	58%
	Clustering	13	0	4	2	0	0	8	0	0	0	13	7	0	2	44	24%
	Search and Info. Retrieval	8	0	1	0	0	0	0	3	1	0	9	5	0	5	24	13%
	Sentiment Analysis	14	1	3	0	0	1	10	0	0	0	12	3	1	0	40	22%
	Content Analysis	10	2	6	2	1	3	4	0	1	2	8	5	4	0	54	30%
	Recommendation	7	0	3	0	0	2	2	0	2	1	12	8	3	2	30	16%
	Summarization	14	0	1	0	0	0	5	2	0	0	9	1	0	3	25	14%
	Visualization	5	0	1	1	0	0	8	0	1	1	10	4	1	0	20	11%

new feature) [151, 160, 171] and by the type of requirements these reviews formulate (e.g., functional or non-functional) [90, 93, 187]. Such aggregated information can be further summarized and visualized to developers as a report of all the feature requests reported for an app [13, 121, 160].

Requirements Classification. User feedback can be classified in a number of dimensions [41]. Several studies classified user comments based on types of requirements the feedback conveys [90, 93, 94, 95, 187]. These works typically classified the feedback into two broad categories: functional requirements (FRs) specifying the behavior of an app, and nonfunctional requirements (NFRs) describing the constraints and quality characteristics of the app. The classification at a further level of granularity has been also demonstrated [80, 90, 95, 96, 187]; User feedback can be classified into the concrete quality characteristics it refers to e.g., defined by ISO 25010 model [220] so that software engineers could analyse candidate requirements more efficiently.

Requirements Prioritization. Statistics about user opinions and requests can help prioritizing software maintenance and evolution activities [2, 8, 9, 66]. Bugs and missing features that are more commonly reported can be prioritized over those less commonly reported [75, 98, 151]. Users' request may not by themselves be sufficient for prioritization (one must also consider costs and the needs of other stakeholders) but can provide valuable evidence-based information to support prioritization [66, 120, 163].

Table 2.15: How often certain combinations of app review analyses are used to realise a SE activity; IE refers to Information Extraction; CL denotes Classification; CU signifies Clustering; CA presents Content Analysis; SA denotes Sentiment Analysis; SIR refers to Search and Information Retrieval; RE presents Recommendation; SU denotes Summarization; and VI signifies Visualization.

	Software Engineering Activity														**Studies**	
	REQUIREMENTS				DESIGN		TESTING				MAINTENANCE					
Combinations of App Review Analysis	Requirements Elicitation	Requirements Classification	Requirements Prioritization	Requirements Specification	Design Rationale Capture	User Interface Design	Validation by Users	Test Documentation	Test Design	Test Prioritization	Problem and Mods. Analysis	Requested Mods. Prioritization	Help Desk	Impact Analysis	**Number**	**Percentage**
CL	9	4	4	2	3	0	1	0	0	0	6	1	1	0	18	10%
CU	2	0	0	0	0	0	0	0	0	0	1	1	0	0	3	2%
CA	1	0	2	0	0	1	1	0	0	2	3	1	1	0	9	5%
CL + CA	2	2	0	0	1	1	0	0	0	0	2	1	0	0	6	3%
CL + SU	2	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1%
SIR + SU	0	0	0	0	0	0	0	0	0	0	2	0	0	2	2	1%
CA + RE	0	0	0	0	0	0	0	0	0	0	1	2	1	0	2	1%
IE + CA + SU	2	0	0	0	0	0	2	0	0	0	0	0	0	0	2	1%
IE + CL + CU	1	0	0	0	0	0	1	0	0	0	1	1	0	0	4	2%
CL + SIR + SU	0	0	0	0	0	0	0	1	0	0	1	0	0	1	2	1%
CL + CU + SU	1	0	0	0	0	0	0	0	0	0	2	0	0	0	2	1%
IE + SA + RE	1	0	0	0	0	0	0	0	0	0	1	0	0	0	2	1%
IE + SA + CL + VI	1	0	1	0	0	0	2	0	0	0	2	0	0	0	2	1%
IE + CL + CU + RE + VI	0	0	0	0	0	0	0	0	0	0	2	1	0	0	2	1%

Requirements Specification. Requirements specification consists in structuring and documenting detailed descriptions of the software required behaviour and quality properties [221]. App reviews can instead serve for generating lightweight partial documentation of user requirements; they convey information about functional and non-functional requirements, usage scenarios and user experience [2, 15, 66, 72, 97, 98]. Software engineers can immediately benefit from review mining approaches to facilitate this information in the form of first drafts of software requirements specifications (SRS) or user stories [2, 15, 66]. These approaches can for example classify reviews by the type of requests users make (e.g., asking for new functions); summarise reviews referring to the same requests and generate provisional SRS based on the information. Such SRS may list new functions that users require; recap scenarios in which these functions are used; and report statistics indicating relative importance of the requirements e.g., by the number of users requesting the functions [15]. Since users often justify their needs and opinions, SRS may also document user rationales serving later for requirements negotiation or design decisions [97, 98].

2.3.4.2 Design

A few studies motivated app review analysis to assist software design activities: user interface (UI) design [10, 11, 190] and capturing design rationale [5, 97, 98, 99, 187].

User Interface Design. The success of mobile applications depends substantially on user experience [7, 11]. For the app to be successful, software engineers should design the interface to match the experience, skills and needs of users [41]. Alqahtani and Orji performed the content analysis of user reviews to identify usability issues in mental health apps [10]. They manually tagged 1,236 reviews with different types of usability issues for 106 apps from Apples App Store and Google Play. Poor design of user interface was the second most frequently reported issue. It has been found that user-submitted content concerning interface may provide valuable design recommendations on how to improve interface layout, boost readability and easy app navigation. UI/UX designers should therefore take advantage of the feedback. If addressed, it would likely increase user engagement with the apps and reduce the attrition rate [11].

Design Rationale Capture. Design rationale is essential for making the right design decisions and for evaluating architectural alternatives for a software system [222, 223]. A few studies motivated their approaches to capture potential reasons for design decisions [5, 97, 98, 99, 187]. Kurtanovi and Maalej devised a grounded theory for gathering user rationale and evaluated different review classification approaches to mine the information from app reviews [97, 98]. User justifications e.g., on problems they encounter or criteria they chose for app assessment (e.g., reliability or performance) can enrich documentation with new design rationale and guide design decisions. Similarly, user-reported NFR can convey architecturally significant requirements and serve as rationale behind an architecture decision [5, 99, 222]. To capture such requirements, app reviews can be classified by quality characteristics users discuss [5, 222].

2.3.4.3 Testing

App reviews analysis can be used to support various testing activities: validation by users (e.g., [58, 140, 160]), test documentation [13, 113, 117], test design [5, 35, 66, 150] and test prioritization [156].

Validation by Users. Evaluating a software system with users usually involves expensive usability testing in a laboratory [160] or acceptance testing performed in a formal manner [224]. In the case of mobile apps, software engineers can exploit user feedback to assess user satisfaction [4, 163, 169] and to identify any glitches with their products [7, 17, 160]. A recent survey with practitioners has shown that developers release the alpha/beta version of their apps to test the general reaction of users and to discover bugs [7].

In line with the direction, several approaches have been proposed to mine user opinions (e.g., [4, 8, 100, 107, 163]) and to generate bug reports (e.g., [12, 101, 160, 163]). Opinion mining approaches help to discover the most problematic features and to quantify the number of negative opinions. Knowing what features users praise or hate can give a developer a hint about user acceptance of these features [4, 7, 88]. Assuming core features have been modified, the team may want to know how users react to these features so that they can fix any issues quickly and refine these features. Analogously, identifying and quantifying reported bugs within a given time frame can help a development team during beta testing before official release [17, 84, 140, 160, 163]. If the number of reported issues is unusually high, development teams can reschedule the release of a new version in order to refocus on quality management and testing [12, 15].

Test Documentation. Test documentation can be partly supported by analysing app reviews [13, 113, 117]. Iacob et al. developed a tool that produce a summary of bugs reported in reviews with a breakdown by app version and features that these bugs refer to [13]. Such summary can form the basis for later debugging the app and fixing the problems. User comments can also be integrated into mobile app testing tools [113, 117]. Originally, the tools generate a report of stack traces leading to an app crash [113, 117]. Analyzing the information to understand the root of the problems can be often counterintuitive. In such case, user comments can be used as a human readable companion for such report; linked to a related stack trace, user-written description of the problem can instantly guide testers where to look up for the emerged fault [113, 117].

Test Design. Analysing app reviews can support test case design [5, 35, 66, 150]. Analysing reported issues can help testers determine the app behavior, features, and functionality to be tested [150]. Reviews may describe a particular use of the software in which users encountered an unusual situation (e.g., crashing without informing users of what happened) or inform about the lack of supporting users in finding a workaround [66]. Such information may help testers to design test cases capturing exceptions leading to a problem or to exercise new alternative scenarios other those initially considered [5, 35, 66]. Additionally, identifying negative comments on quality characteristics can help in specifying acceptance criteria an app should hold [5]. For example, user complaints about performance efficiency can indicate performance criteria for functions that are expected to finish faster or more smoothly [5].

Test Prioritization. Reviews and their ratings have been found to correlate with a download rank, a key measure of the apps success [1, 60]. User complaints about specific issues can have a negative impact on rating, and in turn discourage users from downloading apps [60]. Therefore, it has been suggested to prioritize issue-related test cases based on frequency and impact of these complaints [60, 150]. To address device-specific problems a development team must test their apps on a large number of devices, which is inefficient and costly [225]. The problem can be partially ameliorated by selecting devices submitted from reviews having the greatest impact on app ratings [156]. The strategy can be particularly useful for the team with limited resources that can only afford to buy a few devices. Using the strategy, they can determine the optimal set of devices they can buy on which to test their app [156].

2.3.4.4 Maintenance

In attempt to support software maintenance, review analysis has been proposed for problem and modification analysis, requested modification prioritization, help desk and impact analysis (see Table 2.13).

Problem and Modification Analysis. Software engineers strive continuously to satisfy user needs and keep their app product competitive in the market [7]. To this end, they can exploit approaches facilitating problem and modification analysis (e.g., [115, 154, 193, 194]). The approaches detect user requests in app store feedback and classify them as problem reports and modifications requests [78]. Fine-grained classification can be carried out too, for example, to detect specific issues like privacy [86, 87, 91] or concrete change requests like features enhancement [26, 103]. Mining such information allows software engineers to determine and analyze user demands in timely and efficient fashion [16, 69, 73]. By analysing the dynamics of reported problems over time, software engineers can immediately spot when a ”hot issue” emerges and link it to a possibly flawed release [57, 100, 162, 163]. Moreover, they can generate a summary of user demands to obtain interim documentation serving as change request/problem report [13, 65, 161].

Requested Modification Prioritization. App developers may receive hundreds or even thousands of reviews requesting modifications and reporting problems [18, 87, 114]. It is therefore not a trivial task for developers to select those requests which should be addressed in the next release [18]. As with requirements, developers can investigate statistics concerning these requests (e.g., how many people requested specific modifications), estimate their impact on perceived app quality (e.g., expressed as user rating) or analyze the how these requests change over time (e.g., [110, 114, 120, 147, 196]). Assuming developers have to decide which change to address first, they could select one with the largest share in the numbers of requests, or the one whose feedback most drives down the most app rating [14, 75, 110]. Similarly, observing a sharp growth in feedback reporting of a specific problem (e.g., security and privacy), it may suggest that the issue is harmful to users and should be resolved quickly.

Help Desk. Help desk typically provides end-users with answers to their questions, resolves their problems or assists in troubleshooting [41]. Analogously, app developers can respond to specific user reviews to answer users' questions, to inform about fixing problems or to thank users for their kind remarks about apps [126, 148, 149, 177]. Though the task is not traditionally included in the typical responsibilities of software engineers, user support and managing the product reputation on the app store are essential to the app success; they should be viewed as important activities in the software lifecycle. In fact, responding to reviews motivate app users to revise their feedback and ratings to be more positive [148]. Some users even update their feedback to inform developers that the response solved users' problems or to thank for help [148, 149]. Since responding to a large number of reviews can be time-consuming, developers can make use of approaches highlighting reviews that are more likely to require a response [126, 177]; and generate automatic replies to these reviews [149, 157, 158, 159].

Impact Analysis. Review mining approaches help developers to discover modification requests posted in reviews; to identify app source code affected by these modifications [78]; and to estimate how implementing the modifications may impact users satisfaction [63, 103, 112, 116]. The approaches typically cluster feedback requesting the same modifications [78, 103, 116], then search and retrieve links between review clusters and corresponding source code artefacts referring to the modifications [63, 78, 103, 112, 116]. Such information can be useful for engineers before an issue of new release as well as afterwards. Software engineers can track which requests have (not) been implemented; monitor the proportion of reviews linked to software changes; and estimate the number of users affected by these changes. After the release has been issued, software engineers can also use the approaches to observe gain/loss in terms of average rating with respect to implemented changes.

RQ3: Supporting Software Engineering

- Analysing app reviews can support software engineers in requirements, design, testing, and maintenance activities.
- Most primary studies analyse app reviews to support (i) requirements elicitation, (ii) requirements prioritization, (iii) validation by users, (iv) problem and modifica-

tion analysis, and (v) requested modification prioritization.

- More than a third of primary studies (62 out of 182, 34%) do not explicitly mention the software engineering use cases of the proposed mining approach.

2.3.5 RQ4: Empirical Evaluation

To answer RQ4 (how are app review analysis approaches empirically evaluated), we used data items: F10 (evaluation objective), F11 (evaluation procedure), F12 (metrics and criteria), F14 (annotated datasets), F15 (annotation task), F16 (number of annotators), F17 (quality measure) and F18 (replication package). We found that 109 primary studies performed empirical evaluation of review mining approaches; 105 studies included evaluation of effectiveness and 23 of user-perceived quality.

2.3.5.1 Effectiveness Evaluation

A common procedure for effectiveness assessment consists of four steps: (i) formulate an evaluation objective, (ii) create an annotated dataset, (iii) apply the approach on the annotated dataset, and (iv) quantify the effectiveness. The evaluation objective refers to assessing the degree to which an approach can correctly perform a specific mining task or analysis (see Section 2.3.2). Human judgement is usually required to create the annotated dataset. Primary studies involved humans performing the task manually on a sample of reviews and annotating the sample with correct solutions. Such annotated dataset (called the “ground truth”) served as a baseline for evaluating the approach and quantifying the outcome.

Most studies provided a detail description of how each step of their evaluation methods have performed. Hence, we could record additional information:

Availability of Dataset and Tool. 30% of papers included information for obtaining their tool and/or annotated dataset. In addition to the reported information in the surveyed literature; we also contacted the authors of 105 primary studies to request replication packages. Tables 2.16 provides an overview of 23 annotated datasets that are publicly available, reporting the reference to the paper, a short description of the dataset and its size in terms of number of reviews, whereas Table 2.17 presents 16 available tools, providing the reference to the paper and a short description of the characteristics of the tool. The references to the tools and the datasets are available in the supplementary material [54].

Evaluation Objective. Scholars evaluated the effectiveness of their app review mining approaches in performing: Classification, Clustering, Sentiment Analysis, Information Extraction, Searching and Information Retrieval, Recommendation and Summarization.

Annotation Procedure. The number of annotators labeling the same review sample (or their fragment) ranged from 1 to 5 with the median of 2 human annotators. Only 26 primary studies (25%) reported how the quality of their annotated datasets has been measured. The three most common metrics for inter-rater agreement evaluation were Cohen's Kappa [226], Percentage Agreement [227] and Jaccard index [167]. Percentage Agreement and Cohen's Kappa were used to measure the quality of human annotation for Classification, Sentiment Analysis, or Feature Extraction; Jaccard index was used for assessing the human agreement for the task of Searching and Information Retrieval; whereas Fleiss' Kappa was used to assess the quality of manual Clustering. No study reported how the agreement was measured when annotators performed, Recommendation, or Summarization task.

Characteristics of Dataset. Most annotated datasets were created using reviews coming from Google Play and Apple Store (84% in total); the remaining datasets have been created using reviews from Amazon Appstore, Black Berry App World; Huawei Store, Windows Phone Store and 360 Mobile Assistant. On average, an annotated dataset has been prepared using 2,800 reviews collected from a single app store; the reviews were collected for 19 apps from 6 app categories. Table 2.18 provides five-number summary that details descriptive statistics about the datasets.

Effectiveness Quantification. Three most common metrics used for assessing the effectiveness of app review mining approach are precision, recall, and F1-measure [167]. The metrics were employed for evaluating Classification, Clustering, Information Extraction, Searching and Information retrieval, Sentiment Analysis, Recommendation and Summarization.

A few studies deviate from the common procedure outlined above. The studies evaluated their review mining approaches without annotated datasets:

- Eight studies asked annotators to assess the quality of an output produced by their approaches, instead of creating an annotated dataset before applying the mining approach. This was practiced for evaluating Classification [70], Clustering [8, 64, 103], Information Extraction [9, 70], Searching and Information Retrieval [115], and Recommendation [35].

- Seven studies used other software artefacts as an evaluation baseline rather than creating an annotated dataset [16, 73, 107, 126, 150, 173, 177]. To evaluate Recommendation (e.g., determining priorities for reported issues), the studies compared recommended priorities for issues with priorities for the issues reported in user forums or changelogs; to assess the quality of Clustering, the studies benchmarked the output of their approaches with topics from app changelogs; whereas to evaluate their approaches in Recommending reviews that need to be responded, the studies used information of already responded reviews that developers posted in app stores.

Table 2.16: Publicly-available datasets of annotated reviews.

Reference	Description	Size
Chen et al. [29]	Indicated whether the content of each review is informative or uninformative.	12,000
Guzman et al. [59]	Tagged reviews with topics (e.g., bug report, feature shortcoming, complaint, usage scenario).	4,500
Gu and Kim [14]	Identified a type of user request each review convey (e.g., bug report, feature requests).	2,000
Maalej and Nabil [12]	Reviews labeled with a type of user requests (bug report, feature request, rating, user experience).	4,400
Di Sorbo et al. [65]	Reviews labeled with 12 topics (e.g. security) and user intention (e.g., problem discovery).	3,439
Panichella et al. [81]	Reviews labeled with 5 categories useful from the maintenance perspective (e.g., problem discovery).	852
S´anger et al. [164]	Identified user opinions (feature and sentiment).	1,760
Ciurumelea et al. [116]	Tagged reviews with mobile specific categories (e.g. performance, resources, battery, memory).	4,166
Goren et al. [5]	Labeled reviews with software quality requirements (e.g., usability, reliability, portability, compatibility).	360
Lu and Liang [95]	Reviews labeled with functional and non-functional requirements (e.g., usability, performance).	2,000
Grano et al. [117]	Annotated reviews with their topics and a type of issue users reports.	6,600
Jha et al. [215]	Annotated a type of user feedback (feature request, bug reports, and others).	2,930
Nayebi et al. [210]	Annotated reviews with a type of a user request (e.g., problem discovery).	2,383
Pelloni et al. [113]	Reviews labeled with a crash report category.	534
Scoccia et al. [212]	Annotated reviews with 10 categories of userss concern.	1,000
Al Kilani et al. [188]	Labeled reviews with 5 categories: bug, new feature, performance, security, usability or sentimental.	7,500
Dąbrowski et al. [110]	Reviews annotated with 20 app features.	200
Jha et al. [187]	Labeled reviews with non-functional requirements users discuss (e.g. usability, dependability).	6,000
Scalabrino et al. [151]	Reviews labeled with a feedback category (e.g., bug report, feature request).	3,000
Shah et al. [31]	Identified features discussed in reviews.	3,500
Stanik et al. [165]	Annotated a type of user feedback (problem reports, inquiries, and irrelevant).	6,406
Dąbrowski et al. [4]	Annotated reviews with 1,521 user opinions i.e., pairs of features and their related users' perceived sentiment.	1,000
Guo and Singh [71]	Annotated reviews with user stories i.e., action-problem pairs.	200

2.3.5.2 User Study

Twenty three studies evaluated their review mining approaches through user studies. The objective of these evaluation was to qualitatively assess how the approach and/or their facilitated

Table 2.17: Publicly-available app review mining tools.

Reference	Description
Di Sorbo et al. [65]	SURF tool classifies reviews by users' intention; cluster them then generates their summaries.
Panichella et al. [81]	ARdoc tool classifies reviews with a type of user requests (e.g., feature request, problem discovery, information seeking, information giving and other.)
Johann et al. [9]	SAFE tool extracts features from reviews and matches them with features present in app descriptions.
Wei et al. [115]	OASIS tool classifies reviews by reported issue; links them to warnings from static analysis tools; and recommends the priorities of these warnings.
Deshpande et al. [211]	The tool classifies reviews by a type of a user request (e.g., problem discovery).
Dhinakaran et al. [175]	The tool classifies reviews based on types of user feedback i.e., feature request, bug report, user experience and rating.
Scoccia et al. [212]	The tool classifies app reviews into users' concerns related to android run-time permission.
Shah et al. [176]	A tool classifying app reviews based on their feedback type (e.g., feature request, problem report).
Jha et al. [187]	Tool classifies app reviews by non-functional requirements user discuss (e.g. usability, dependability).
Pelloni et al. [113]	BECLoMA tool links stack traces from testing tools to user reviews referring to the same crash.
Scalabrino et al. [151]	CLAP tool classifies reviews by their types; clusters them; then recommends their relative-importance.
Shah et al. [31]	SAFE tool reimplementation facilitating feature extraction from reviews and app descriptions.
Shah et al. [217]	A reimplementation of a tool facilitating feature extraction using supervised ML technique.
Stanik et al. [165]	A tool classifying reviews by the type of user feedback (problem reports, inquiries, and irrelevant).
Guo and Singh [71]	CASPER tool for extracting and synthesizing user stories of problems from app reviews.
Hadi and Fard [108]	AOBTM tool discovers coherent and discriminative topics in reviews.

Table 2.18: Five-summary numbers providing descriptive statistics of annotated datasets that primary studies used to evaluate app review mining approaches.

Characteristics	Min.	Q1	Med.	Q3	Max.
No. App Stores	1	1	1	2	3
No. Apps	1	7	19	185	1,430,091
No. App Categories	1	4	6	10	35
No. App Reviews	80	1,000	2,800	4,400	41,793

analysis is perceived by intended users (e.g., software engineers). Such an evaluation procedure typically consists of the following steps: (i) define an evaluation subject and assessment criteria, (ii) recruit participants, (iii) instruct participants to perform a task with an approach or a produced analysis, (iv) elicit participant's opinions of the approach through questionnaire and/or interviews.

We looked in details at how studies perform each of the steps. The extracted data yields the following insights:

Evaluation Subjects. User studies evaluated the following types of app review analyses: Clustering, Classification, Sentiment Analysis, Information Extraction, Search and Information Retrieval, Recommendation, Summarization, and Visualization.

Assessment Criteria. Five evaluation criteria were typically taken into account: 1) Usefulness denoting the quality of being applicable or having practical worth; 2) Accuracy indicating the ability of being correct; 3) Usability signifying the quality of being easy to use; 4) Efficiency indicating the capability of producing desired results with little or no human effort; and 5) Informativeness denoting the condition of being informative and instructive. Table 2.19 provides reference mapping of user studies with a breakdown of evaluation criteria and evaluated subjects.

Study Participants. The number of participants involved in the study ranges from 1 to 85 with the median of 9 participants. The participants included professionals, scientists and students; Table 2.20 details the types of participants taking part in user studies and provides references to the corresponding studies.

Evaluation Procedure. A The participants were instructed to either perform a specific task with or without the use of the mining approach being evaluated, to review the outputs produced by the approach, or to simply trial the proposed approach without being given any specific tasks.

Table 2.19: Reference mapping of user studies with a breakdown of an evaluation criterion and app review analysis.

Criterion	**App Review Analysis**
Accuracy	Information Extraction [16, 131], Classification [18, 65, 81, 151], Clustering [78, 103], Summarization [79].
Efficiency	Classification [29, 84, 116], Recommendation [35, 157], Summarization [65, 79, 86, 123].
Informativeness	Classification [101, 116, 131], Recommendation [122] Summarization [65, 79, 86], Visualization [16, 100].
Usability	Recommendation [157], Summarization [65, 79, 131].
Usefulness	Information Extraction [16, 102, 131], Classification [15, 65, 81, 84, 101, 116], Clustering [103], Search and Information Retrieval [103], Sentiment Analysis [102], Recommendation [18, 35, 122, 151], Summarization [79, 86, 123], Visualization [14, 121].

RQ4: Empirical Evaluation

- Review mining approaches are evaluated in terms of their effectiveness in performing review analyses and their user-perceived quality.

Table 2.20: Reference mapping of user studies with a breakdown of the types of participants taking part in the studies.

Sector	Participant	Reference
Academia	Student	[16, 35, 79, 86, 101, 116, 121, 123, 157]
	Researcher	[15, 29, 65, 79, 84, 123]
Industry	Architect	[15]
	Business Analyst	[131]
	Developers	[14, 15, 65, 78, 79, 81, 100, 101, 102, 103, 116, 121, 122, 123]
	Product Manger	[131]
	Project Manager	[15, 18, 65, 151]
	Requirement Engineer	[15]
	Software Engineer	[65, 79, 101, 131]
	Software Tester	[65, 79]

- To evaluate effectiveness, studies compare outputs of mining approaches with human-generated ones on sample datasets. Most datasets, however, have not been published.
- To assess perceived quality, studies perform user studies with software professionals, scientists and students. Participants are typically tasked to use a mining approach with a certain objective; then assess it based on specific quality criteria e.g., usefulness.

2.3.6 RQ5: Empirical Results

We answered RQ5 (how well do existing app review analysis approaches support software engineers) based on data item F13 (evaluation result). The data comes from 109 studies reporting results of their empirical evaluations: effectiveness evaluations (105 studies) and user studies (23 studies). We synthesize results of these studies in the subsequent subsections.

2.3.6.1 Effectiveness Evaluation Results

The methodology that primary studies employed for effectiveness evaluation was too diverse to undertake a meta-analysis or other statistical synthesis methods [228]; these studies characterized for example diversity in their treatment (e.g., review mining approach), population (e.g., review dataset) or study design (e.g., annotation procedure). We thus employed 'summarizing effect estimates' method [228]; Table 2.21 reports the magnitude and the range of effectiveness results that primary studies reported for different review analyses with a breakdown of a mined information type.[5]

[5]No effectiveness evaluation was performed w.r.t. content analysis and visualization.

Information Extraction. The effectiveness of extracting information from reviews depends on the type of mined information. Techniques for extracting features from reviews have the lowest performance: the median precision of 58% [8] and the median recall of 62% [164]; and the most diverging results: precision varies from 21% to 84% [31, 122]. Techniques for extracting user requests and NFRs from reviews have higher performance with a median precision above 90% [5, 13] and only small variations between techniques.

Classification. App reviews can be classified by information types these reviews contain, such as user requests, NFRs and issues. State-of-the-art review classification techniques have the median precision above 81% (e.g., [93, 95, 211, 212]) and the median recall around 83% (e.g., [104, 187, 210, 212].)

Clustering. Studies have shown the accuracy of clustering semantically related reviews to be 83% [64]; this result is in line with findings concerning the quality of review clustering, where authors reported MojoFM of 80% [18, 151].

Search and Information Retrieval. Mining approaches showed effectiveness in retrieving reviews to specific information needs; in particular, the results show that tracing information between reviews and issues in ticketing systems and between reviews and source code can be precise: the median precision above 75% [103, 112, 113]; and complete: the median recall above 70% [63, 112, 113, 117]; whereas linking reviews to goals in goal-models has been achieved with the median precision of 85%; and the median recall of 73% [121, 122]. Similarly, finding reviews related to specific features has been reported with 70% of precision and recall of 56% [9]. The variability of the results e.g., precision between 36%-80% [110, 123], however, may lead to inconclusive findings.

Sentiment Analysis. A single study reported the review's sentiment can be identified with the overall accuracy of 91% [129]; the result was however claimed for a single app only. Identifying the sentiment of a review with respect to a specific app feature is less effective with the median precision of 71% and the median recall of 67% [4, 88].

Recommendation. Recommending priorities for user requests was reported with medium to high effectiveness: the median accuracy of 78% [18, 151] and precision of 62% [16, 73]. Whereas, generating review responses was reported with BLEU-4[6] greater than 30% [158], which reflects human-understandable text.

Summarization. Mining techniques were recorded to generate a compact description outlining the main themes present in reviews with recall of 71% [215].

[6]The metrics quantifying the quality of generated text on a scale of 0% to 100%.

Table 2.21: Summary of the results of controlled experiments measuring the effectiveness of app review analysis techniques.

App Review Analysis	**Mined Information**	**Results**
Information Extraction	Features	Precision range from 21% to 84% [31, 122] with the median precision of 58% [8]; recall range from 42% to 77% [31, 127] with the median recall of 62% [164].
	User Request	Precision range from 85% to 91% [58, 160] with the median precision of 91% [13]; recall range from 87% to 89% [58, 160] with the median recall of 89% [13].
	NFR	Precision at the level of 92% [5].
Classification	User Request Type	Precision range from 35% to 94% [61, 207] with the median precision of 80% [171]; recall range from 51% to 99% [170, 207] with the median recall of 82% [74, 104].
	NFR Type	Precision range from 63% to 100% [89, 92] with the median precision of 74% [93, 95]; recall range from 63% to 92% [93, 92] with the median recall of 79% [95, 187].
	Issue Type	Precision range from 66% to 100% [67, 89] with the median precision of 76% [212]; recall range from 65% to 92% [67, 89] with the median recall of 79% [212].
Clustering	Similar Reviews	Accuracy range from 80% to 99% [114, 161] with the median accuracy of 83% [64]; MojoFM range from 73% to 87% with the median MojoFM of 80% [18, 151].
Search and Information Retrieval	Feature-Related Review	Precision range from 36% to 83% [110, 123] with the median precision of 70% [9]; recall range from 26% to 70% [110, 123] with the median recall of 56% [9].
	Review-Goal Links	Precision range from 84% to 85% with the median precision of 85% [121, 122]; recall range from 71% to 74% with the median recall of 73% [121, 122].
	Review-Issue Links	Precision range from 77% to 79% [63, 114] with the median precision of 77% [112]; recall at the level of 73% [63, 112].
	Review-Code Links	Precision range from 51% to 82% [116, 117] with the median precision of 82% [103, 113]; recall range from 70% to 79% [103, 116] with the median recall of 75% [113, 117].
Sentiment Analysis	Feature-Specific	Precision ranges from 68% to 74% with the median precision of 71%; recall ranges from 64% to 70% with the median of 67% [4, 88].
	Review	Accuracy at the level of 91% reported in a single study only [129].
Recommendation	User Request Priority	Accuracy range from 72% to 83% with the median accuracy of 78% [18, 151]; precision range from 60% to 63% with the median precision of 62% [16, 73].
	Review Response	BLEU-4 at the level of 36% [158].
Summarization	Review Summary	Recall at the level of 71% [215].

Table 2.22: Summary of user studies evaluating the perceived quality of app review analysis techniques.

App Review Analysis	Criterion	Results
Information Extraction	Accuracy	Machine learning techniques extract issues with an acceptable accuracy [16]; collocation algorithms generate too many false positives for feature extraction [131].
	Usefulness	Extracting issues is useful for app maintenance [16]; feature extraction may help to analyze user opinions, and to prioritize development effort [102, 131].
Classification	Accuracy	Classifying reviews by types of user requests, topics is sufficient for a practical use [18, 65, 81, 151]; ML techniques are superior in accuracy to simple NLP [131].
	Efficiency	Classifying reviews automatically (e.g., by user requests) could reduce up to 75% of the time that app developers spend for a manual feedback inspection [29, 84, 116].
	Informativeness	Grouping feedback by topics is instructive; a group may express shared users' needs specific to e.g., app features, emerging issues, new requirements [101, 131, 116].
	Usefulness	Classifying reviews is useful for identifying different users' needs [65, 81, 101]; comprehending reported issues and problematic features [15, 84, 101, 116].
Clustering	Accuracy	Clustering reviews conveying semantically similar information is performed with high precision and completeness [78, 103].
	Usefulness	Useful for grouping reviews bearing a similar message e.g. reporting the same issues [103].
Search and Information Retrieval	Usefulness	Helpful for linking reviews to source components (e.g., methods, class or field name) [103].
Sentiment Analysis	Usefulness	Helpful for detecting user opinions about app features; it can support prioritising development effort to improve these features [102].
Recommendation	Efficiency	Suggesting reviews requiring responses lower app developers' workload [157]; suggesting test cases triggering bugs can save developers' effort on manual bug reproduction [35].
	Informativeness	Recommending users' goals that an app needs to satisfy is informative from the developers' perspective [122].
	Usability	Automatic review response system shows higher usability compared to the original app store mechanism; survey respondents favored the the system [157].
	Usefulness	Recommending priority of user requests is useful for release planning [18, 151]; suggesting goals an app needs to satisfy may support this app evolution [122]; whereas recommending test cases triggering bugs can help developers to reproduce bug-related user reviews [35].
Summarization	Accuracy	Summarizing topics and requests that users post can be made with high accuracy [79].
	Efficiency	Prevents more than half of the time required for analyzing feedback [65, 79, 86]; it helps to immediately understand user opinions, user requests or security issues [86, 123].
	Informativeness	Listing user requests (e.g., bug reports), topics or security issues through a summary table is informative and expressive for software engineers [65, 79, 86].
	Usability	Table showing frequently posted user requests, and topics is easy to read [65, 79, 131].
	Usefulness	Summarizing user requests, topics and/or security issues that reviews convey is useful for better understanding users' needs [79, 86, 123].
Visualization	Informativeness	Presenting what topics users discuss and how they change over time can inform about emerging issues [16], or relevant user opinions [100].
	Usefulness	Showing heat-map of feature-specific sentiments or their trends over time can help developers to identify problematic features [14]; and to understand to what extent an app satisfied users' goals [121].

2.3.6.2 User Study Results

Twenty three studies evaluated user-perceived quality of review mining approaches. Table 2.22 provides synthesis of user study results that primary studies reported for different review analyses with a breakdown of an evaluation criterion.

Information Extraction. Extracting information from reviews e.g., issue reports and user opinions is useful for developers [16]; it can help to elicit new requirements or prioritize development effort [102, 131]. In particular, machine learning techniques are able to identify issues with an acceptable accuracy [16]; feature extraction methods instead produce too imprecise analyses to be applicable in practice [131].

Classification. Review classification showed their utility for identifying different users' needs e.g., feature requests, or bug reports (e.g., [65, 78, 81, 84, 101, 116]). Such categorized feedback is informative and ease further manual review inspection [101, 116, 131]. Practitioners reported to save up to 75% of their time thanks to the analysis [29, 84, 116]; and that their accuracy is sufficient for the practical application [18, 65, 81, 151].

Clustering. Review clustering is convenient for grouping feedback conveying similar content; for example, those reporting the same feature request or discussing the same topic [78, 103]. Evaluated approaches can perform the analysis with a high level of precision and completeness [78, 103].

Searching and Information Retrieval. Developers admitted the usefulness linking reviews to the source code components to be changed [103]; the task traditionally requires enormous manual effort and is highly error-prone.

Sentiment Analysis. Analyzing user opinions can help to identify problematic features and to prioritize development effort to improve these features [102].

Recommendation. Project managers found recommending priorities of user requests useful for release planning [18, 151]; it can support their decision-making w.r.t. requirements and modifications that users wish to address. Developers perceived an automatic review response system as more usable than the traditional mechanism [157]; recommending reviews that require responding and suggesting responses to the reviews can reduce developers' workload [157]. Similarly, recommending goals that an app needs to satisfy is informative and may guide this app evolution [122]; whereas suggesting test cases triggering bugs can be useful for developers to reproduce bug-related user reviews; and save cost on manual bug reproduction [35].

Summarization. Compact description outlining most important review content is useful for developers in their software engineers activities [79, 86, 123]; in particular, summaries conveying information about frequently discussed topics, user opinions, user requests and security issues. Facilitating this information in a tabular form is easy to read and expressive [65, 79, 131]. Such summaries are generated with sufficient accuracy to be used in practical scenarios [79, 86]; in fact, developers reported to save up to 50% of their time thanks to the analysis [65, 79, 86, 123].

Visualization. Presenting trends of frequently discussed topics can inform developers about urgent issues, 'hot features', or popular user opinions [16, 100]. Heat-map illustrating feature-specific sentiment (i.e., user options) helps developers to understand users experience with these features [14]; it indicates which features users praise and which are problematic. Visualizing how user opinions change over time aids developers in examining users' reactions e.g., to newly implemented modifications for these features; and understanding to what extent an app satisfies users' goals [121].

RQ5: Evaluation Results

- Effectivenesses evaluations revealed mining approaches achieve promising accuracy for 5 app review analyses: (1) Classification; (2) Clustering; (3) Searching and Information Retrieval; (4) Recommendation and (5) Summarization.
- In user studies, software engineers found 8 types of app review analyses useful: (1) Information Extraction; (2) Classification; (3) Clustering; (4) Search and Information Retrieval; (5) Sentiment Analysis; (6) Recommendation; (7) Summarization and (8) Visualization. The analyses can ease their software engineering activities; reduce their workload; and support their decision-making.
- User studies showed software engineers find accuracy of most app review analyses promising for the practical usage; yet the quality of both Information Extraction and Sentiment Analysis seems to be insufficient.

2.4 Discussion

In this section we highlight and discuss some of the findings from our study, summarize literature gaps and point to directions for future research.

2.4.1 Mining App Reviews Is a Growing Research Area

Mining app reviews for software engineering is a relatively new research area. The first paper on analysing app review for SE was published in 2012. Nevertheless, the analysis of demographics has revealed that the research area increasingly attracts the attention of scholars. A recent survey in app store analysis found 45 papers relevant to app review analysis published up to 2015 [1]; whereas this SLR recorded 182 studies published as of the end of 2020. In fact, the total number of papers published in line with the directions has grown substantially in the last three years by over 53%. We thus anticipate the field will continue to grow in the next years. The most frequent venues where scholars have published their work concern high-quality software engineering conferences and journals (see Table 2.5). These imply there is not only an increasing effort on exploring the research direction, but also suggest contributions of this effort is relevant from a software engineering perspective; in fact, empirical evidences (RQ5) demonstrate that software engineers find mining app reviews useful in support of their SDLC activities; mining approaches can reduce their workload; facilitate knowledge that would be difficult to obtain manually. As other work [1], we also hypothesize factors leading to the research interest in the field concern increased popularity of mobile apps, an easy access to user feedback on a scale not seen before as well as a general interest in adopting data mining techniques for mining software repository.

2.4.2 Software Engineering Goals and Use Cases

App reviews analysis has broad applications in software engineering (RQ3). It can be used to support a variety of activities in requirements, design, testing and maintenance (see Table 6). Researchers however do not always clearly describe the envisioned software engineering use cases for their techniques. So far, research in this area has been driven mostly by the opportunity to apply ML techniques on app reviews. Most studies (66%) relate their approaches to potential software engineering activities, but they remain vague about details of how they envision the techniques to be used in practice. A greater focus on software engineering goals and use cases would increase the relevance and impacts of app review analysis techniques. This systematic literature review includes a complete inventory of already envisioned software engineering use cases for the various app review analysis technique (RQ3). This inventory can provide the basis for a more detailed investigation of software engineering goals and use cases for app review analysis tools. This investigation will contribute to designing future app review analysis tools that best serves the needs of software engineers.

2.4.3 Need Of Reference Model For Review Mining Tools

A reference model of stakeholders goals, use cases and system architectures for review mining tools would help structuring a research effort in this area, and communicate how fitting review mining techniques together help to address real stakeholders needs. In the future, scholars can elaborate such a model by generalizing existing review mining solutions; explaining how different components help to realize intended use cases and satisfy stakeholders' goals. The model would also help researchers to identify and reuse common components in a typical architecture of review mining tools as well as explain the novelty and the contribution of their work within that framework.

2.4.4 Small Size Of Evaluation Datasets

A great deal of effort has been made to evaluate the effectiveness of data mining techniques (RQ4). Primary studies, however, used evaluation datasets of small size (on average 2,800 reviews). This is a tiny portion of user-submitted feedback in app stores. Popular mobile apps (like WhatsApp or Instagram) can receive more than 5,000 reviews per day, and more than one million reviews in a year [21]. This is a significant threat to the validity of their results when trying to generalize them e.g., [94, 110, 116]. The problem is attributed to the substantial effort of a manual review annotation; labeling 900 reviews can take up to 12.5 hours [8]. As none of the surveyed studies tried to tackle the problem, it opens an avenue for the future research. Researchers may experiment with semi-automated data labeling techniques currently exploited to minimize an effort for preparing training datasets [186, 175, 229]. Providing the problem was handled, scholars should still be mindful of a sampling bias when curating dataset [230]. Techniques to ameliorate the latter problem, however, have been well-studied in a recent study [61].

2.4.5 Replication Packages

Most papers did not make available their review mining tools and evaluation datasets (see Table 2.16 and Table 2.17). This hinders the replicability of these works as well as new comparative studies. Our survey contains a single replication study and that study reported the challenge in validating results of the original work due the absence of annotated dataset and an insufficiently documented evaluation procedure [31]. Future studies should provide replication packages, including evaluation datasets, procedures, and approaches so that researchers will be able to validate existing works and confirm reported findings. It will also help in benchmarking approaches and provide a baseline for evaluating new approaches aiming at improving performance of review mining techniques.

2.4.6 Impacts On Software Engineering Practice

Although empirical measures of effectiveness give promising results (e.g. precision and recall above 70% for app review classification techniques), it is not yet clear to what extent app review analysis techniques are already good enough to be useful in practice (RQ5). Identifying what performance the approaches should have to be useful for software engineers is an important open question [231, 232]. Essentially, an approach facilitating review analysis should synthesize reviews so that the effort for further manual inspection of the outcomes of that analysis would be negligible or at least manageable. Clearly, the effort would depend on a scenario an approach aims to realize. In addition to evaluating review analysis tools in terms of ML performance metrics (e.g precision and recall), it will become increasingly important to evaluate them in terms of software engineering concerns: Does it save time? Does it improve the quality of, for example, the requirements elicitation and prioritisation process? etc. Evaluating techniques with respect to software engineering concerns is more difficult but necessary to ensure research effort is aligned with real stakeholders' goals. Such evaluation will involve a combination of quantitative and quantitative studies aimed at reducing our current uncertainty about potential impacts of review mining techniques on software engineering activities.

2.4.7 Practitioners Requirements For App Review Mining Tools

Numerous tools have been developed in the context of app review analysis research; they satisfy requirements coming mainly from scholars rather than practitioners. We have recorded no research studying what features the tools should facilitate nor what goals they should satisfy. The current research is *data-driven* rather than *goal-driven*. The studies apply different types of app review analyses and techniques to mine information from app reviews without explicitly examining the practitioners' perspective. It is not clear to what extent the tools satisfy the real practitioners goals. Though existing user studies provide evidence software practitioners find certain types of analyses valuable e.g., Classification [103], yet more systematic research is necessary in such directions to understand practitioners' needs. Future research should plan to actively involve practitioners, for example via interview sessions or the analysis of their development practices, to understand why the tools are needed; what SE goal they want to satisfy with the tools; what features the tools should facilitate; and how the tool would be used in the organizational settings. Such knowledge will help to understand the actual use cases scenarios of the tools, and to identify whether there is misalignment between what state-of-the-art tools offer and what practitioners actually need.

2.4.8 Verifying the Industrial Needs for App Review Analysis

Most studies motivated their mining approaches to reduce the manual effort for app review analysis. Such rationale seems to be reasonable in the context of popular apps (e.g., WhatsApp or Facebook Messenger) that are frequently commented and receive hundreds or thousands reviews per day. However, an average app receives 22 reviews per day [2]. It seems therefore legitimate to study the potential impact of the app review analysis research on the app store industry; and to what extent the mining tools would be useful in the industrial settings. Such a study could address this problem from multiple perspectives e.g., what small, medium and large app development organization are interested in app review mining tools? who in the organization would use the tools? is the manual app review analysis 'the real pain of the practitioners? if so, how 'the pain manifests itself? are any tasks obstructed? is the problem generating additional costs? Answering the questions could help to understand who are the actual beneficiaries of the app review analysis research; and what is the size of that market. Not only it would help to scope and justify the future research directions, but it would also provide insights to commercializing this research.

2.4.9 Pay Attention to Efficiency and Scalability of Mining Tools

Primary studies are mostly focused on evaluating effectiveness and perceived quality of their mining tools. We however recorded no study focused on assessing the efficiency and the scalability of their tools; studying the efficiency informs how much time the tools take to produce their outcomes; whereas scalability informs how the time changes when the input of the tools increase. The efficiency and the scalability are fundamental qualities of analytics tools [233]; app review mining tools are no exception. The number of reviews that an app receives can vary from a few to more than thousands. Existing approaches e.g., for feature extraction [8] or app review classification [12] rely on NLP and ML techniques that may be challenging to scale-up [234]. Future studies, therefore, should take the efficiency and the scalability into consideration when developing and evaluating their mining tools to demonstrate the tools can be used in the practical settings.

2.4.10 The Problem of Training ML Techniques

Machine learning is the most frequent type of techniques used for app review analysis (RQ2). Most of these techniques, however, are supervised one and require a training dataset consisting of manually annotated reviews. Preparing manually annotated dataset is time-consuming and often error-prone [8]. More importantly, such annotated dataset might be domain- and time-specific; annotated reviews of one app might not be re-usable for training a technique for the feedback of the other app; further, the dataset may be prone to data drift - a phenomenon in which the characteristics of app reviews change over the time. In such a case, ML techniques must be periodically trained with up-to-date training dataset to maintain their predictive abilities [235]. Recent studies thus experimented with active learning [175] and semi-supervised techniques [186] to reduce the cost of annotating a large amount of data.

More research is however needed to understand how many reviews should be annotated for preparing a training dataset when the techniques are used in the industrial settings; how often such dataset needs to prepared; and whether or not the practitioners would accept the cost of preparing this dataset.

2.5 Threats to Validity

One of the main threats to the validity of this systematic literature review is incompleteness. The risk of this threat highly depends on the selected list of keywords forming search queries. To decrease the risk of an incomplete keyword list, we have used an iterative approach to keyword-list construction. We constructed two queries: generic and one specific. The generic query was formed using keywords appearing in the index of terms in sample studies analysing app reviews for SE. Specific query was formed based on a set of keywords representing concepts of our research objective. As in any other literature survey, we are also prone to a publication bias. To mitigate this threat, we complemented a digital library search with other strategies. We conducted an issue-by-issue search of top-level conferences and journals as well as performed the backward and the forward snowballing.

To ensure the quality and reliability of our study, we defined a systematic procedure for conducting our survey, including research questions to answer, searching strategies and selection criteria for determining primary studies of interest. We conducted a pilot study to assess the technical issues such as the completeness of the data form and usability issues such as the clarity of procedure instructions. The protocol was reviewed by the panel of researchers in addition to the authors of the study. It was then revised based on their critical feedback. Consequently, the selection of primary studies followed a strict protocol in accordance to well-founded guidelines [40, 236, 237].

Another threat to validity we would like to highlight is our subjectivity in screening, data extraction and classification of the studied papers. To mitigate the threat, each step was performed by one coder, who was the first author of this study. Then, the step was cross-checked by a second coder. Each step was validated on a randomly selected sample of 10% of the selected papers. The percentage inter-coder agreement reached for all the phases was equal or higher than 80%, indicating high agreement between the authors [47]. In addition, the intra-rater agreement was performed. The first author re-coded once again a randomly selected sample of 20% of studied papers. Then an external evaluator, who has no relationship with the research, verified the agreement between the first and the second rounds. The percentage intra-coder agreement was higher than 90%, indicating near complete agreement [47].

Table 2.23: Main differences between our study and previous surveys.

Dimensions	Our Study	Martin [1]	Genc-Nayebi [25]	Tavakoli [22]	Noei [39]
Study Type	SLR	Survey	SLR	SLR	Survey
Time Period	’10-’20	’00-’15	’11-’15	’11-’17	’12-’19
No. Papers	182	45	24	34	21
Paper Demographics	✓	✓	✓	✓	
App Review Analyses (RQ1)	✓	✓			✓
Mining Techniques (RQ2)	✓		✓	✓	✓
Supporting SE (RQ3)	✓				
Empirical Evaluation (RQ4)	✓				
Empirical Results (RQ5)	✓				

A potential limitation of our study is the lack of quality assessment of selected primary studies. Not every study is the same quality; and therefore, their strength of evidence may be different too [40]. There is however neither an agreed definition of the study ‘quality’ nor standard assessment criteria [40]; the existing procedures for performing systematic reviews suggest performing quality assessment of individual studies based on their designs, limitations, and reported information [40]. Rigorous and objective quality assessment of an individual studies is however a challenge in itself; it requires in-depth expertise in the research area and considerable manual effort. Therefore, to ensure the selected studies were sufficient quality for the information synthesis and to minimize the manual effort, we only selected peer-reviewed studies. In principle, in the peer-review process, one or more experts in the field assess the overall quality of a manuscript before it is published. We admit the synthesis of empirical results (e.g., the effectiveness) still needs an individual examination of each study design to make comprehensive understanding of the strength of their evidence. The methodology of primary studies was however too diverse for a meta-analysis or other statistical synthesis methods [228]; the studies used diverse treatments, evaluation datasets and study designs. We thus employed ‘summarizing effect estimates’ method to make the holistic interpretation of empirical results while considering the different strength of their evidence [228].

A similar threat concerns whether our taxonomies are reliable enough for analysing and classifying extracted data. To mitigate this threat, we used an iterative content analysis method to continuously develop each taxonomy. New concepts which emerged when studying the papers were introduced into a taxonomy and changes were made respectively. These taxonomies were discussed between all the authors and agreed upon their final form.

2.6 Related Work

This review is not the first effort synthesizing knowledge from the literature analysing app reviews for SE [1, 22, 25, 39]. Our SLR, however, differs substantially from previous studies in scope of the literature surveyed and depth of our analysis. Table 2.23 shows the differences between our study and previous works in accordance with dimensions we considered for the comparison. We grouped the dimensions into information related to study characteristics and topics surveyed in our study. The characteristics concern a study type (i.e., systematic literature review or survey), time period covered and the number of papers surveyed. The topics concern: Paper Demographics, App Reviews Analyses (RQ1), Mining Techniques (RQ2), Supporting Software Engineering (RQ3), Empirical Evaluation (RQ4) and Empirical Results (RQ5).

Martin et al. surveyed literature with the aim to demonstrate a newly emerging research area i.e., app store analysis for software engineering [1]. The scope of their survey is much broader than of our study, as it covers literature analyzing various types of app store data (e.g., API, rank of downloads, or price). Our work has much narrower scope, focussing only on app review analysis, but studies the paper in greater depths in order to answer our five research questions. Though the related survey also addresses (RQ1), our study is more up-to-date and larger in scale, covering 182 papers. More importantly, most dimensions of our SLR i.e., RQ2-RQ5, are missing in this other study.

Two other studies addressed our RQ2, but partially, as they are narrower in scope [22, 25]. Tavakoli et al. surveyed the literature in the context of techniques and tools for mining app reviews [22]. Similarly, Genc-Nayebi et al. consolidated literature to synthesize information on techniques for opinion mining [25]. Our SLR addresses the dimension more broadly, rather than in context of techniques for a specific review analysis or tool-supported approaches. We have made an effort to consolidate general knowledge on techniques the literature employs for 9 broad types of review analyses. We also provided mapping between different review analyses and techniques facilitating their realization.

Noei and Lyons summarized 21 papers analysing app reviews from Google Play [39] . The authors provided an overview of each paper, briefly explaining the applications, and mention their limitations. The surveyed papers were selected subjectively, rather than following a systematic searching procedure. In contrast, our study is a SLR rather than a summary. Following a systematic procedure, we selected 182 studies that we carefully read and then synthesized to answer five research questions. The related work marginally covers information for RQ1 and RQ2.

In summary, previous studies do not cover our research questions related to software engineering activities (RQ3) and empirical evaluations (RQ4 and RQ5). They partly cover our research questions RQ1 and RQ2 but on a smaller set of papers and in less details.

2.7 Conclusion

In this study, we presented a systematic literature review of the research on analysing app reviews for software engineering. Through systematic search, we identified 182 relevant studies that we thoroughly examined to answer our research questions. The findings have revealed a growing interest in the research area. Research on analysing app reviews is published in the main software engineering conferences and journals e.g., ICSE, TSE or EMSE and the number of publications has tripled in the last four years. The research in this area will likely continue to gain importance as a consequence of increased interest in mobile app development.

This systematic literature review structures and organizes the knowledge on the different types of app review analyses as well as data mining techniques used for their realization. With that knowledge, researchers and practitioners can understand what useful information can be found in app reviews, and how app review analysis can be facilitated at abstract and technical levels. More importantly, the literature review provides a new light on why mining app reviews can be useful; the findings identifies 14 software engineering activities that have been the target of previous research on app review analysis. Important future research for app review analysis will involve developing a deeper understanding of the stakeholders' goals and context for app review analysis tools in order to increase the applicability, relevance and value of these tools.

The findings have revealed that software engineers find mining approaches useful and with promising performance to generate different app review analyses. It however remains unclear to what extent these approaches are already good enough to be used in practice. It will become increasingly important to evaluate them in terms of software engineering specific concerns: Does it improve the quality of, for example, the requirements elicitation and prioritization process? We also recommend an empirical evaluation will continue to improve in scale and reproducibility. Research in this area is currently inconsistent quality in terms of an evaluation method and an ability for the research to be reproduced. Future studies should share evaluation datasets and mining tools, allowing their experiments to be replicated. They should also pay more attention to the scalability and the efficiency of their mining approaches.

In conclusion, this study helps to communicate knowledge on analyzing app reviews for software engineering purposes. We hope our effort will inspire scholars to advance the research area and assist them in positioning their new works.

Chapter 3

Use Cases and Reference Architecture For Mining App Reviews

3.1 Introduction

This chapter has been accepted for publication in the 30th International Requirements Engineering Conference 2022 [238]. The first author's contribution to the paper was to formulate the idea, design and execute the experimentation, collect the results, analyse them, and write the manuscripts; other authors of the papers contributed to the research conceptualization and manuscript revision.

The literature review, in Chapter 2, synthetised the wide range of research in the field of app review analysis for software engineering. The study has founded the existing research takes *a data-driven perspective* for mining app reviews; and focuses on applying different tools and techniques for discovering useful information. Little attention however has been paid to the software engineering use cases of these approaches: how these tools and techniques can be integrated together to address real stakeholders‘ needs is still an open problem.

In this Chapter, we present a study taking *a goal-oriented perspective*. This study addresses the problem by elaborating a unified description of software engineering use cases for mining app reviews; and defining a reference architecture that realize these use cases.

The purpose of the reference architecture is to help researchers and tool developers to identify what components can be included in app review mining tools and how the components can be used together to realize these specific software engineering use cases. The use cases describe the usage scenarios of this architecture for software engineering purposes. More specifically, we consider the following research questions to answer:

RQ1: What are the software engineering use cases for mining app reviews?

RQ2: What reference architecture can realise these use cases?

RQ3: What partial implementations of this reference architecture already exist?

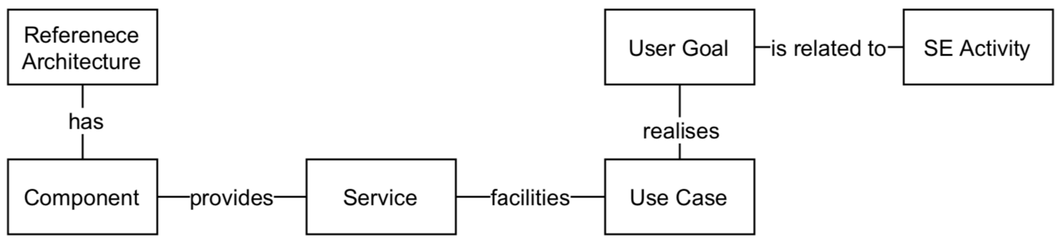


Figure 3.1: The main concepts that we use to describe our reference architecture.

We answered these questions by analysing the scientific literature using a dataset collected in our systematic literature review (see Chapter 2); we extended this dataset in this study with additional information about features of commercial tools that we collected from their vendors' websites. We used the dataset to identify and model software engineering use cases for mining app reviews. We generalized a set of tools proposed in the literature to define the reference architecture. To evaluate the feasibility of the architecture, we mapped the components of this architecture to features found in existing, publicly available research and commercial tools.

The primary contributions of the study are: i) a synthesis of software engineering use cases for mining app reviews, ii) a reference architecture realizing the use cases, and iii) a mapping of architectural components to publicly available tools.

The use cases describe the potential benefits and the use of mining techniques for software engineers. The reference architecture synthesises the diversity of research to realise these benefits and provides a general framework guiding the development and evaluation of future research and tools. The mapping provides evidence of the perceived usefulness of app review analysis in practice and identifies opportunities for commercialisation or technology transfer.

The remainder of the chapter is structured as follows: In Section 3.2, we introduce terminology for this study. In Section 3.3, we discuss the research methodology. In Section 3.4, we present software engineering use cases for mining app reviews. In Section 3.5, we propose the reference architecture, and its realization of the use cases. In Section 3.6, we present the validation of our architecture. In Section 3.7, we discuss perspectives for researchers and practitioners that are facilitated by the use cases and reference architecture presented in this chapter. In Section 3.8, we discuss threats to validity, then related works in Section 3.9. Conclusion is given in Section 3.10.

Table 3.1: What information collected from 182 publications analysing app reviews for SE are in this study used to answer RQ1–RQ3. The information was collected in SLR (see Chapter 2).

ID	Data Item	Description
F1	Software Engineering Activity	What software engineering activity is supported by mining app reviews. We refer to activities generally accepted in the SE community (e.g., requirements elicitation) [41]
F2	Justification	An explanation of why software engineering activity is supported.
F3	App Review Analysis	App review analysis used to support SE activity. We separated collected data item into an app review analysis type (F3.1) e.g., classification; and a mined information type (F3.2) e.g., bug report.
F4	Replication Package	An availability of the replication package, including details about its content such as a tool implementation or an evaluation dataset.

3.2 Terminology

Figure 3.1 illustrates the main concepts that we refer to describe our reference architecture and their software engineering use cases for mining app reviews. We now introduce the terminology, starting with the definition of a reference architecture. A *reference architecture* is a generic architecture for a class of systems; it can be used as a foundation for the design of concrete architectures from this class for a particular domain [239].

In this study, we consider a reference architecture as a generalization of app review analysis tools; each facilitating one or more app review analyses. Reference architectures can generally be presented at different level of abstraction [240]; but they typically show a list of services (a.k.a. functions) organized into components as well as their interactions. We also define the components and services of our reference architecture and communicate how integrating the services can realise software engineering use cases; where a *use case* is a description of how an end-user wants to use a system to meet their goal; it is written from the end-user's perspective [241].

We consider a software engineering use case as a description of the ways an app developer uses an app review analysis tool ('What') to accomplish their goals ('Why') related to software engineering activities; where a *software engineering activity* refers to any activity in the development, evolution, operation and maintenance of software [41].

3.3 Research Methodology

We followed four main steps to answer the research questions RQ1–RQ3: i) data collection, ii) information modeling, iii) knowledge synthesis and iv) reference architecture validation. We first used information collected from scientific publications to model the application of app review analysis in the context of software engineering activates, and then to synthetise this knowledge into software engineering use cases for mining app reviews (RQ1). We also used the collected information to define a reference architecture by generalizing a set of tools that researchers have proposed in the literature (RQ2). To evaluate the feasibility of the reference architecture (RQ3), we mapped their components to the features of research and commercial app review analysis tools that are publicly available. We collected information about the tools and their features from the literature as well as the tools' vendor websites.

i) Data Collection. We used the dataset collected for our systematic literature review (see Chapter 2) as it contains information to answer RQ1–RQ3. The inclusion criteria for this SLR were all the peer-reviewed papers about app review analysis for software engineering activities published between January 1, 2012 and December 31, 2020. The SLR identified 182 publications satisfying these criteria. These publications were identified and analysed using recommended practices for systematic literature reviews [40]. The SLR analysed each publication by systematically extracting and collecting 18 pre-specified types of information (from now called 'data items') that have been reported in a publication. These data items concerned the information about the supported software engineering activity, the type of app review analysis, the type of mining technique, and the empirical evaluation presented in a publication. The complete set of the extracted information is stored in a spreadsheet [54]. Table 3.1 lists data items collected from each publication that we used in this study; we had selected these data items as they facilitated information to answer RQ1–RQ3. SLR applied classification schemas on selected data items for the purpose of information synthesis. The schemas were applied on such collected information of a given type which could not be grouped directly; the information, reported among the publications, was too diverse to be grouped and then synthetized directly. In particular, the software engineering activity (F1) was classified as one of 14 standard activities identified in the software engineering body of knowledge [41]; whereas the app review analysis (F3) was classified as one of 9 broad types identified from the previous surveys on intelligent mining techniques [22] and text analytics [51, 52, 53]. The classification categories have been systematically derived from 182 publications using content analysis [49]. The SLR evaluated the quality of its data extraction and data classification by measuring inter- and intra-rater agreements between the first author of that study and an external assessor.

ii) Information Modeling. We used the collected information F1–F3 ('software engineering activity', 'justification' and 'app review analysis') to model relationships between software engineering activities and different types of app review analysis; and to model the realization of a software engineering activity when the app review analysis is applied in their context. To construct the models, we followed the inductive reasoning method proposed for software system modeling [242, 243]; the method supports a creative process in which different pieces of information are combined to form a new artifact [243]. We first manually examined and compared the collected information about the use of different types of app review analysis for a software engineering activity (F1–F3); we then interpreted this information and iteratively constructed the models. We modeled each software engineering activity realization from the process and the service viewpoints using ArchiMate modelling language [244]; we chose ArchiMate as it is a standard modeling language in business and information technology domains. The title of a model documents the name of a modeled software engineering activity. The process viewpoint in a model documents steps that a developer would follow to complete a software engineering activity; whereas the service viewpoint documents the types of app review analysis that support the developer's steps (e.g., through their automation). We identified the types of app review analysis used in the context of a software engineering activity from the collected information F1 ('software engineering activity') and F3 ('app review analysis'); we specified each type of app review analysis at the coarse-grained level (e.g., 'Classification') and fine-grained level (e.g., 'Classification by request type'). We identified the developer's steps for a software engineering activity and their relationships with the types of app review analysis from the collected information F1 ('software engineering activity') and F2 ('justifications'). As a result of the modeling step, we obtained 19 models for 14 software engineering activities reported in the literature; each model presents the realisation of a software engineering activity with the use of app review analysis. The number of models is greater than the number of the activities as the literature proposed alternative realisations of some of these activities (e.g., requirements elicitation). The models can be found in full as supplementary material [245].

iii) Knowledge Synthesis. We used the models derived from the information modeling step to define the use cases (RQ1) and the reference architecture (RQ2) [246]; we manually examined the models, identified their commonalities, and synthetised the obtained knowledge to form the target artifacts. We first used the models to define the description of the use cases; each model was initially transformed into a separate use case description. A use case description included four types of information: the description of a developer's interaction with an app analysis tool (so-called 'What'), a goal of a use case (so-called 'Why'), a software engineering activity related to this goal, and the justification of how app review analysis supports this activity. We used the information about the elements in the process view of a model (i.e., developer's steps) to elaborate the description of a developer's interaction with an app analysis tool; each developer's step corresponded to a sentence describing such an interaction (e.g., 'developer can identify user sentiment about feature of an app'). We also examined all the steps in a model to infer an overall developer's goal for using the tool; we used this information to define the goal of a use case. In principle, an overall goal of using any data analytics tool is to analyse a certain type of information from data [247]; we therefore examined the developer's steps in the process view of a model to determine what type of information a developer would intend to analyse in app reviews. We then defined the goal of a use case based on this finding. Having analysed all the models derived from the information modeling step, we identified three broad categories of information that a developer would aim to interpret from app reviews: user opinions, user requests and non-functional requirements. We consequently defined the goal of each use case using one of these three categories (e.g., 'analyze user opinions'). To determine a software engineering activity related to this goal, we referred to the title of model recoding this information (e.g., 'requirements elicitation'). As a result, we obtained 19 use cases (from now called 'fine-grained use cases'); each fine-grained use case described how a developer accomplishes one out of 3 goals related to 14 software engineering activities through an interaction with an app review analysis tool. We subsequently grouped the fine-grained use cases based on their goals; and defined one coarse-grained use case per each group. We combined a group of fine-grained use cases into a coarse-grained one as the fine-grained use cases shared a common goal and described the same or related developer's use of an app review analysis tool. A coarse-grained use case aggregated information from a group of fine-grained use cases: their description of the developer's use of an app review analysis tool, their overall goal for using the tool, software engineering activities related to the goal, and the justification of how an app review analysis supports the software engineering activities. As a result, we obtained 3 coarse-grained use cases that we used to answer RQ1. We next defined the reference architecture using the information about the elements in the service view of the models derived from the information modeling step; we defined the components of this architecture and their services using the information about the types of app review analysis modeled in this view. We used information about both fine- and a coarse-grained types of app review analysis that the models described. We first extracted information about all the fine-grained types of app review analysis that have been described in the models (e.g., 'classify reviews by request type', 'identify feature', etc.); we then grouped the extracted information thematically. To define the grouping categories, we used the information about the coarse-grained type of app review analysis that have been described in the models. We used this information to define the grouping categories as the coarse-grained types of app review analysis were more general than fine-graine ones; and coarse-grained types of app review analysis (e.g., 'classification') were shared among fine-grained ones (e.g., 'classify reviews by request type' and 'classify reviews by non-functional requirements'). We consequently obtained 8 broad groups of 18 fine-grained types of app review analysis. Each group aggregated one or more fine-grained types of app review analysis. We defined the architectural components and their services based on these groups. We defined an architectural component using a group of related fine-grained types of app review analysis. We derived the name of a component using a grouping category (e.g., 'classification component'); whereas the information about a fine-grained type of app review analysis in a group corresponded to a service of that component. As a result, we obtained 8 components, each facilitating between 1 and 3 services. We next structured these components into three logical layers of our reference architecture [248]. We used the list of the components to describe the presentation and the service layers of the architecture; we then added a new 'Database' component to describe the data layer. We added this new component as none of the previously identified components were intended to store and facilitate data to the other already identified components. We inferred the missing services of the 'Database' component using the list of the already identified components and the input/output dependency matrix [249]; we listed all the services of the already identified components in the rows to the left of the matrix and in the columns above the matrix; and next marked their input/output dependencies on off-diagonal cells. Having analysed the dependencies, we identified 3 missing services whose outputs served as inputs to the other already identified components and their services. We added the missing services to the 'Database' component. We consequently obtained the target list of 9 components and their 21 services that we used to answer RQ2.

iv) Reference Architecture Validation. The objective of our reference architecture is to facilitate the future design and comparisons of app review mining tools for software engineers. The success of our reference architecture can only be established in the long term by the extent to which it supports this objective. In this study, we validated the feasibility of the reference architecture by verifying whether the partial implementations of this architecture already exist (RQ3). As previous studies (e.g., [250, 251]), we validated our reference architecture using matrix traceability method [252]; this method supports the re-use of parts of a system by comparing components of new and existing systems. We therefore mapped the features of app review mining tools to the components of our architecture. We selected both research and commercial tools that are publicly available. We opted for publicly available tools to increase the reliability of our findings [253]. To identify research tools and their features, we used collected information F3 ('app review analysis') and F4 ('replication package'). To identify commercial tools, we used popular software comparison platforms: TrustRadius [254] and G2 [255]; in each platform, we listed all the tools in the 'Mobile Analytics Tools' category. We then examined short descriptions of the tools and selected those facilitating app review analysis. For the selected tools, we further examined the full descriptions of the tools on their vendor's websites to identify features that these tools facilitate. As a result, our analysis considered 29 app review mining tools: 20 peer-reviewed and publicly available research tools referenced in the SLR and 9 commercial tools that we identified from the software comparison platforms. We then constructed a traceability matrix showing which components of the reference architecture are implemented in each tool [239].

3.4 Software Engineering Use Cases

We now present three coarse-grained software engineering use cases for mining app reviews (RQ1). They cover all the usage scenarios of app review analysis tools that have been envisioned in the literature; we have inferred the use cases from the literature using data items F1–F2 ('software engineering activity' and 'justification') as described in the modeling and knowledge synthesis steps in the previous section. The use cases can help app developer to: i) analyze user opinions, ii) analyze user requests and iii) analyze non-functional requirements; and they are related to 14 software engineering activities [41]. The following sections provide the description of each use case and explain how they contribute to these activities. For each use case, we give a brief description of the intended use of a system from the app developer's perspective (the 'What'); we describe what app developers do with a system to accomplish their goal; and specify the relation of this use case to software engineering activities (the 'Why') [241]. We give references to exemplary papers that justify the relevance of the use cases for the software engineering activities; the complete list of papers is available in the supplementary material to the SLR [54].

3.4.1 Use Case 1: Analyze User Opinions

Description (What): The app developer's goal is to analyze user opinions. To accomplish this goal, an app developer can use a system to: identify users' sentiments (positive, negative or neutral) about app features (functional attributes); search for reviews referring to a concrete opinion of their interest; and to summarize these reviews presenting the main points of the reviews.

Software Engineering Activities (Why): This use case contributes to several software engineering activities:

- *Requirements Elicitation, Problem and Modification Analysis:* Negative opinions may indicate problems with app features or dissatisfied users [2, 7]. Examining reviews providing users justifications for these opinions can help to identify problems with app features [4, 110], requested modification [9] or new requirements [110, 131].

- *Requirements Specification:* Reviews justifying negative opinions can be used as information for requirements specification or ad-hoc documentation [2, 97]; they can communicate why negatively commented features do no meet users' goals and provide rationale for alternative variants of app behavior.

- *Requirements Prioritization, Requested Modification Prioritization:* When added with statistics, user opinions may help developers prioritize their work [8, 9]. Developers may compare how often these opinions appear, for how long they have been made, and whether their frequency is increasing or decreasing [4, 110].

- *Validation by Users:* Knowing what features users like or dislike can indicate user acceptance of these features [88, 163]. Examining reviews of these opinions may help to know what users say about these features [7, 140].

3.4.2 Use Case 2: Analyze User Requests

Description (What): The app developer's goal is to analyze user requests. To accomplish this goal, an app developer can use a system to: classify app reviews by the types of user requests (e.g., bug report or feature request); identify what specific user requests have been made in reviews (e.g., "add ability to make a phone call"); summarize app reviews; identify which app reviews should be replied and generate a response to the reviews; and to link user requests to the other software artifacts (e.g., stack traces, or source code).

Software Engineering Activities (Why): This use case contributes to several software engineering activities:

- *Requirements Elicitation, Problem and Modification Analysis, User Interface Design:* Reviews requesting new features may point to new requirements [12, 15], whereas those requesting changes or reporting problems may indicate potential perfective and corrective modifications [2, 18].

- *Requirements Specification, Test Documentation:* Reviews with user requests can serve as ad-hoc specification as they include information about bugs and ideas for new features that users communicate [15, 66]. Reviews, describing steps to reproduce bugs, can be useful to complement crash reports of stack traces which could be difficult to understand standalone [113, 117].

- *Test Design:* Analyzing reviews reporting problems may describe scenarios in which an unusual situation emerged or there was lack of workarounds [66, 150]. The information can be used to design test cases capturing such an exception and exercising the scenarios [66, 5].
- *Requirements Prioritization, Requested Modification Prioritization, Test Prioritization:* User requests may help developers prioritize their work when added with statistics (e.g., [114, 151, 154]). Comparing how often these requests appear, for how long these requests have been made, and whether their frequency is increasing or decreasing may indicate relative users importance of these requests [60, 110].
- *Validation by Users:* Identifying and quantifying reviews reporting problems can help during public beta testing before an official release [15, 160]. If the number of problems is high, or the problems concern core features, a new release can be lingered [12, 110]. Analyzing the reviews can help to know what user says about these problems [4, 7].
- *Help Desk:* Responding to reviews may answer app users questions or assist in troubleshooting [148, 149]. Such responses may also inform users about addressing their requests (e.g., new modifications or fixing problems) [148, 149].
- *Impact Analysis:* Reviews with change requests can be linked to source code and indicate code snippets requiring modifications [103, 112]. And vice versa, source code modifications can be traced back to reviews; Quantifying these reviews could help determine the impact of implemented modifications e.g., how many user requests have been satisfied by the modifications [63, 116].

3.4.3 Use Case 3: Analyze Non-Functional Requirements

Description (What): The app developer's goal is to analyze non-functional requirements (NFRs) i.e., non-functional attributes of an app. To accomplish this goal, an app developer can use a system to: classify app reviews by the types of NFRs they convey; identify what specific requirements have been made in app reviews; summarize app reviews; and to identify user sentiment related to the reviews.

Software Engineering Activities (Why): This use case contributes to several software engineering activities:

- *Requirements Elicitation, Problem and Modification Analysis, Test Design:* Negative reviews discussing quality attributes may indicate poor app quality, problems with features, or conditions an app must satisfy to be accepted by users (e.g., [93, 95, 187]). Understanding what users say about these attributes may help to elicit new requirements, identify issues, or serve as inspiration for designing test scenarios or acceptance criteria (e.g., [66, 5, 150]).

- *Requirements Categorization:* Reviews discussing NFRs can be labeled with quality attributes they discuss (e.g., performance, or usability) [95, 187].
- *Requirements Specification, Design Rationale Capture:* Reviews discussing NFRs can serve as ad-hoc documentations of user requirements [15, 187]. Such specification may be later used by a software engineer to justify reasons why certain decisions have been made e.g., why a certain mobile phone model has been supported, or why a certain security protection mechanism has been implemented [97, 98].
- *Requirements Prioritization, Requested Modification Prioritization:* Added with statistics, reviews reporting NFRs may help the development team to prioritize their works [7, 97]. Comparing how frequently NFRs are reported, for how long do users reported them and whether these numbers are increasing or decreasing may indicate users relative importance of these requirements (e.g., [18, 98, 151]).

3.5 Reference Architecture

This section presents the reference architecture that we defined upon the generalisation of existing app review mining tools (RQ2). We inferred the components and their services from the literature using data item F3 (‘app review analysis’) as described in the modeling and knowledge synthesis in the research methodology (see Sec. 3.3). Figure 3.2 illustrates the structure of the reference architecture that is comprised of functional components; each component is a modular set of services. The components are organized in three layers: presentation, service and data layer. In the following subsections, we define the components and demonstrate how they can be wired to realize the software engineering use cases.

3.5.1 Architectural Components and Services

Visualization Component. The component aids developers in interpreting mined information from reviews. The component provides three services, each generating analytics dashboard per review mining use case: *Display User Opinions Analysis, Display User Request Analysis and Display NFRs Analysis*. Generated dashboards organize and visualize mined information using typical graphical widgets such as tables, pie charts, bar charts and trend analysis.

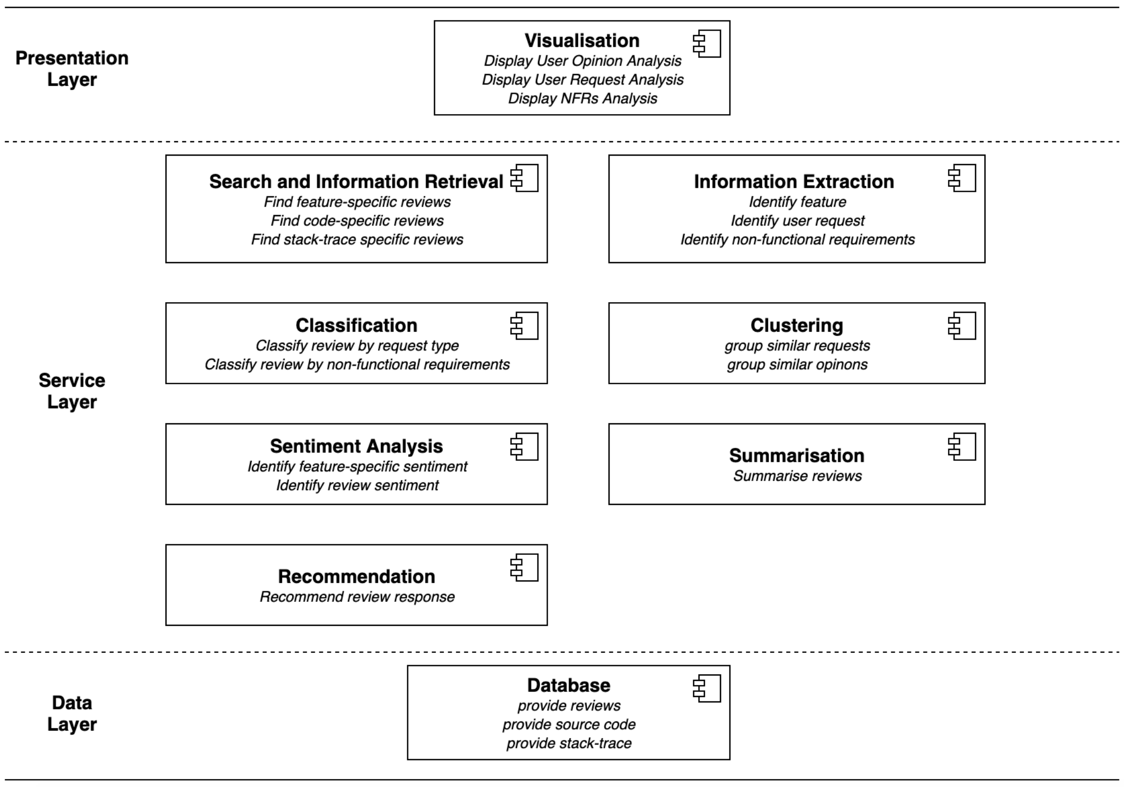

Presentation Layer
Visualisation
Display User Opinion Analysis
Display User Request Analysis
Display NFRs Analysis
Service Layer
Search and Information Retrieval
Find feature-specific reviews
Find code-specific reviews
Find stack-trace specific reviews
Information Extraction
Identify feature
Identify user request
Identify non-functional requirements
Classification
Classify review by request type
Classify review by non-functional requirements
Clustering
group similar requests
group similar opinons
Sentiment Analysis
Identify feature-specific sentiment
Identify review sentiment
Summarisation
Summarise reviews
Recommendation
Recommend review response
Data Layer
Database
provide reviews
provide source code
provide stack-trace

Figure 3.2: Reference architecture for mining app reviews: components illustrated by rectangles; their name written in bold; services they facilitate written in italics.

Search and Information Retrieval Component. Searching component provides three services: *Find feature-related reviews* searches for reviews discussing a specific feature of developer interest. An example of a retrieved review for a queried feature “add reservations” is “Please, improve adding reservations”; *Find code-specific reviews* links reviews requesting modifications to a source code component these modifications refer to (e.g., class or method). An example of such a link is between a review “I cant download most of the songs” and a code class “SongDownloadManager”; *Find stack-trace specific reviews* finds reviews referring to a specific stack trace. An example of a retrieved review for a stack trace “..com.hideKeyboard..” is a review “Crashing when I hide the keyboard”.

Information Extraction Component. The information relevant from a software engineering perspective can be found at a different text location of app review (e.g., in a middle of a sentence). The information extraction component locates such information in a review and pulls it out. The component facilitates three services extracting information from reviews; *Identify features* extracts features discussed in reviews. An example of an extracted feature from a review “I have to send message to my colleague” is “send message”. *Identify user requests* extracts user requests reported in reviews. Given a review “Please, add the ability to send message. I need it.”, the extracted user request is “please add the ability to send message". *Identify non-functional requirements* extracts phrases discussing quality attributes of an app. An example of a non-functional requirement extracted from a review "I like the app, but the navigation should work faster on my iPhone" is "navigation should work faster on my iPhone".

Classification Component. Classification component assigning predefined categories to reviews. The component provides two services classifying reviews based on different categorization schemas; *Classify reviews by request types* categories reviews based on types of user request reviews convey (e.g. bug report, feature request or modification suggestion). An example of a review classified as "bug report" is "The app crash when I send message. Please, fix it". *Classify reviews by non-functional requirements* categories reviews based on quality attributes that users discuss in reviews. An example review classified as "performance" is "The uploading file works very slow, any improvements?".

Clustering Component. Clustering component organizes reviews, their sentences or textual snippets into groups (called clusters) whose members are similar in some way e.g., discussing the same problem. The component provides two services grouping different types of information; *Cluster user requests groups* user requests based on their similar content. An example of user requests clustered together are "The application is slow" and "the app could work faster"; *Cluster user opinions* groups opinions referring to the same features. User opinion referring to features "attach file in message" and "add file to message" are an example of the same cluster.

Sentiment Analysis Component. Sentiment Analysis provides services interpreting users sentiments discussed in reviews. The component provides two services; *Identify feature-specific sentiment* interprets users sentiments about features that users discuss in reviews (also known as user opinions). A "positive" sentiment about a feature "send message" is an example of a user opinion in a review "I like the current app version". *Identify review sentiment* interprets overall sentiments expressed in reviews. An example of review expressing "positive" sentiment is "I like the current app version".

Summarization Component. Summarization facilities *summarize reviews* service that produces a short and compact description outlining the overall content of a review collection. "Fix the problem with sending messages" is an exemplary summary of reviews "The app creates some problems. Sending message crashes whenever I try it. Fix it" and "Love this app, but it often crashes. You should fix the problem with sending messages".

Recommendation Component. The component provides *suggest review response* service. The service identifies reviews that should be replied and generates responses to these reviews. A sample response to a review “The app drains my phone battery. Please fix it” is “Thank you for the comment. We will fix the problem in the next release.”

Database Component. Database stores review mining related data and provides the data to other architectural components. The component provides access to stored data through three services; *Provide reviews* facilitates reviews collected from app store and stored in the database; *Provide stack-trace shares stack traces* generated from automatic testing tool and stored in the database; *Provide source-code* retrieves files with source code stored in the database.

3.5.2 The Realisation of Software Engineering Use Cases

We have already presented the static view of the reference architecture; we specified their components and services. We now present their dynamic view [256]; we present an example of how the architecture could facilitate the use cases defined in the previous section (see Sec. 3.4). Like previous works [251, 257], we present an example of how—and in what order—the components of the architecture can be wired together to realise these use cases.

For each use case realisation, we first give a short paragraph describing what a developer does with a system based on the previous use case specification (see Sec. 3.4). We then present what components can facilitate the intended system use; what services these components exploit; and what data-flows between these components are.

Use Case 1: Analyse User Opinion. We demonstrate the main scenario in which: (i) an app developer identifies users’ sentiments about features of an app, and then (ii) app developer summarises reviews to present the main points. We also present an alternative scenario in which: (iii) an app developer searches for reviews referring to a concrete opinion of their interest and read them. For the sake of illustration, we consider a system storing two app reviews: ’I love send message’ (R1) and ’Video conferences is useless’ (R2). Table 3.2 presents what components and in what orders realise the intended developer’s use of a system in each scenario. Figure 3.3 illustrates dataflows between components in these scenarios.

Use Case 2: Analyse User Request. We demonstrate the main scenario in which: (i) an app developer identifies what user requests have been made in reviews, and then (ii) app developer summarises reviews to present the main points for a bug report. We also present an alternative scenario in which: (iii) an app developer responses to reviews requesting new features. For the sake of illustration, we consider a system storing three app reviews: ‘Please, add the ability to make video call’ (R1), ‘Fix sending messages’ (R2) and ‘Sending messages is broken, it crashes with large attachments’ (R3). Table 3.3 presents what components and

Table 3.2: The Realisation of Analyse User Opinion Use Case.

Precondition	Database stores app reviews: 'I love send message' (R1) and 'Video conferences is useless' (R2).
Main Scenario	**(i) App developer identifies users' sentiments about features of an app.** (1) Database provides reviews R1, R2. The reviews are sent to Information Extraction. (2) Information Extraction identifies features 'send message' and 'video conferences'. The features are sent to Sentiment Analysis. (3) Database provides reviews R1, R2. The reviews are sent to Sentiment Analysis. (4) Sentiment Analysis identifies two opinions: 'positive' for 'sending message' and 'negative' for 'video conferences'. The opinions are sent to Visualization. **(ii) App developer summarises reviews to present the main points.** (5) Database provides review R1. The review is sent to Summarization. (6) Summarization generates a review summary 'I love sending messages'. The summary is sent to Visualization.
Alternative Scenario	**(iii) App developer searches for reviews referring to a concrete opinion of their interest (e.g., 'video conference').** (1) Database provides reviews R1 and R2. The reviews are sent to Searching and Information Retrieval. (2) Searching and Information Retrieval finds feature-related reviews R2. The review is sent to Information Extraction. *Steps 2 to 4 from the main scenario are followed.*

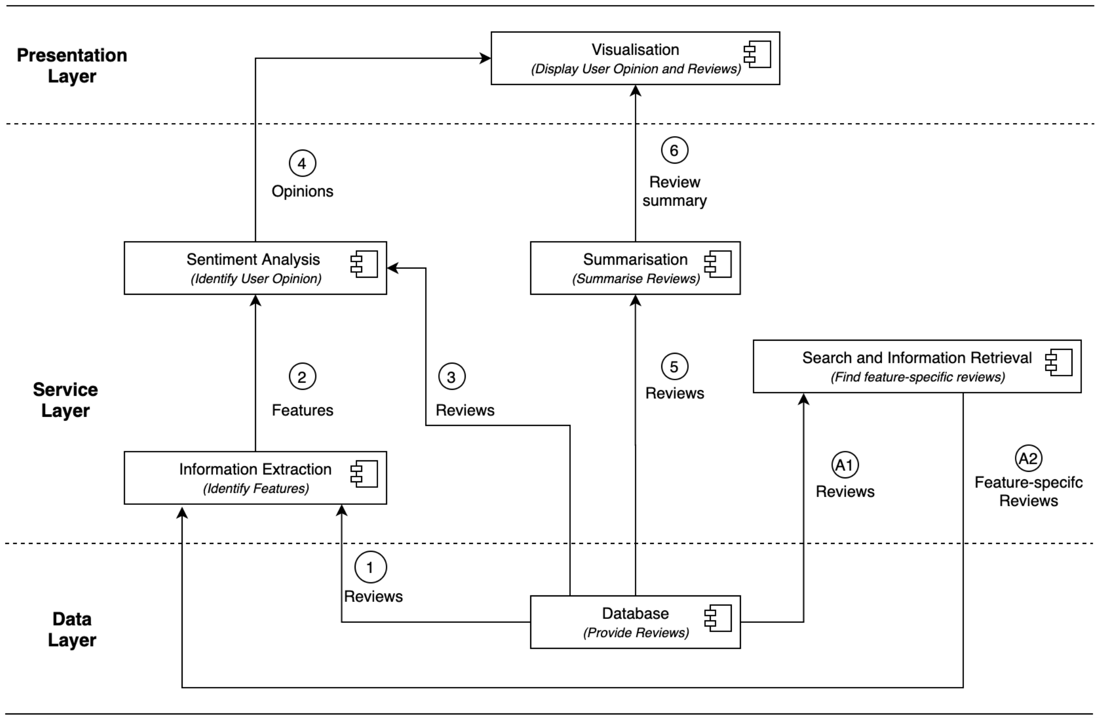

Presentation Layer
Visualisation (Display User Opinion and Reviews)
4 Opinions
6 Review summary
Sentiment Analysis (Identify User Opinion)
Summarisation (Summarise Reviews)
Service Layer
2 Features
3 Reviews
5 Reviews
Search and Information Retrieval (Find feature-specific reviews)
Information Extraction (Identify Features)
A1 Reviews
A2 Feature-specifc Reviews
1 Reviews
Data Layer
Database (Provide Reviews)

Figure 3.3: Dataflow between components for Analyse User Opinion Use Case.

in what orders realise the intended developer's use of a system in each scenario. Figure 3.4 illustrates dataflows between components in these scenarios.

Use Case 3: Analyse Non-functional Requirements. We demonstrate the main scenario in which: (i) an app developer classifies reviews by the types of NFRs they convey, (ii) app

Table 3.3: The Realisation of Analyse User Request Use Case.

Precondition	Database stores app reviews: 'Please, add the ability to make video call' (R1), 'Fix sending messages' (R2) and 'Sending messages is broken, it crashes with large attachments' (R3).
Main Scenario	**(i) App developer identifies what user requests have been made in reviews.** (1) Database provides reviews R1, R2, R3. The reviews are sent to Classification. (2) Classification categorizes R1 as 'feature request', R2 and R3 as 'bug report'. The classified reviews are sent to Information Extraction. (3) Information Extraction identifies user requests: 'add the ability to make video call' (U1), 'Fix sending messages' (U2) and 'Sending messages is broken' (U3). The requests are sent to Clustering. (4) Clustering groups U2, U3 together and cluster U1 separately. The request groups are sent to Visualization. **(ii) App developer summarises reviews to present the main points for a bug report.** (5) Review Database provides reviews R2, R3. The reviews are sent to Summarization. (6) Summarization generates a summary of reviews 'Fix sending messages, it crashes with large attachments'. The review summary is sent to Visualization.
Alternative Scenario	**(iii) App developer responses to reviews requesting new features.** *Step 1 from the main scenario is followed.* (1) Classification categorizes R1 as 'feature request', R2 and R3 as 'bug report'. The classified reviews are sent to Recommendation. (2) Recommendation suggests review response 'Thx for the suggestion. We will add the feature it in the next release'. The response is sent to Visualization.

developer identifies user sentiment per each NFR category, and then (iii) app developer identifies what specific requirements have been made in app reviews. For the sake of illustration, we consider a system storing two app reviews: 'New GUI is too old-fashioned. Improve it.' (R1) and 'This app is not secure anymore. Continuously collects my data.' (R2). Table 3.4 presents what components and in what orders realise the intended developer's use of a system in this scenario. Figure 3.5 illustrates dataflows between components.

Table 3.4: The Realisation of Analyse NFR Use Case.

Precondition	Database stores app reviews: 'New GUI is too old-fashioned. Improve it.' (R1) and 'This app is not secure anymore. Continuously collects my data.' (R2).
Main Scenario	**(i) App developer classifies reviews by the types of NFRs they convey.** (1) Database provides reviews R1, R2. The reviews are sent to Classification. (2) Classification categorizes R1 as 'Usability' and R2 as 'Security'. The classified reviews are sent to Visualization. **(ii) App developer identifies user sentiment per each NFR category.** (3) Database provides reviews R1, R2. Reviews are sent to Sentiment Analysis. (4) Sentiment Analysis identifies 'negative' sentiment in R1 and 'positive' sentiment in R2. Review-specific sentiments are sent to Visualization. **(iii) App developer identifies what specific requirements have been made in app reviews (e.g., about Usability).** (5) Database provides review R1. The review is sent to Information Extraction. (6) Information Extraction identifies requirement 'GUI is too old-fashioned'. The requirement is sent to Visualization.

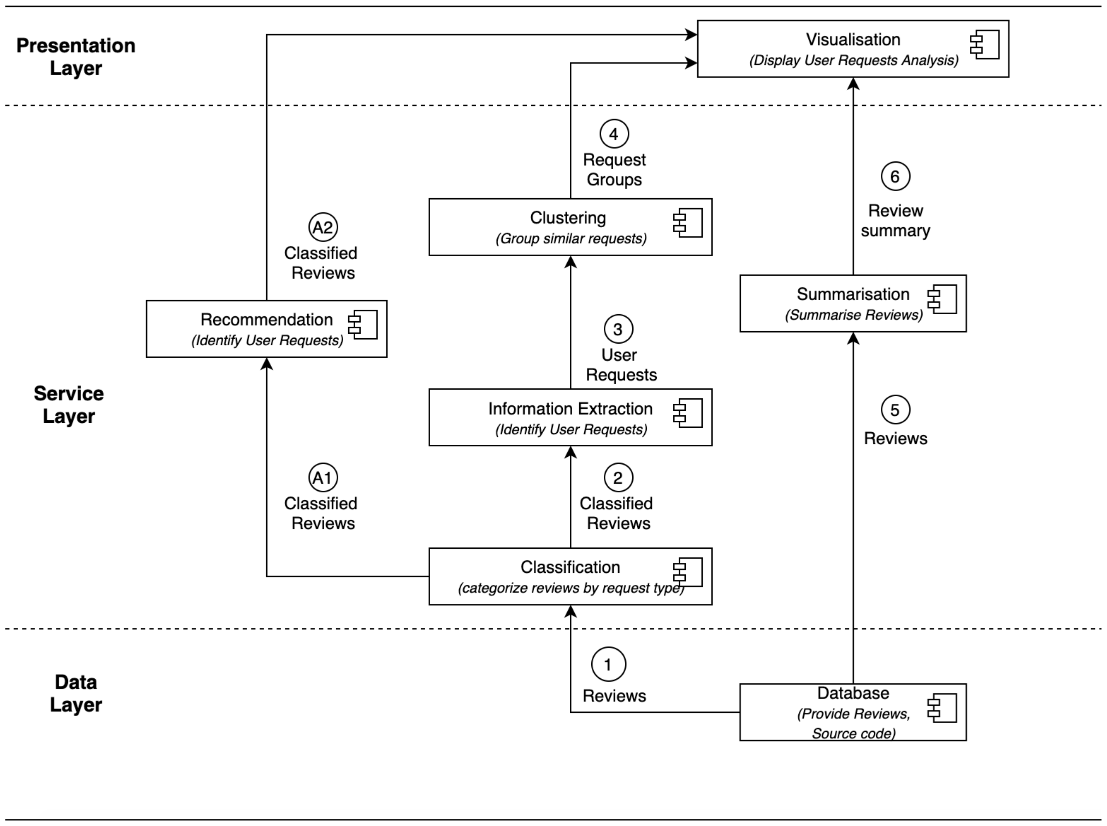


Figure 3.4: Dataflow between architectural components for User Request Use Case.

3.6 Validation

We now answer RQ3 (what partial implementations of this reference architecture already exist?) to validate the feasibility of the reference architecture [258]. Similarly as previous works [250, 251], we validate our reference architecture using traceability matrix method [252]; this method supports reuse of parts of a system by comparing components of new and existing systems. We used information about features of available app review mining tools that we collected from research literature and commercial websites (see Sect. 3.3), and then we related these features to the components of our reference architecture.

Table 3.5 illustrates the mapping between features of app review mining tools and the components of our reference architecture. Rows denote app review mining tools with a breakdown of research prototypes and commercial tools, whereas columns indicate the architectural components and their services. A “✓” at an intersection indicates that the particular tool implements the service of the concrete architectural component. The bottom row reports the number of tools implementing a specific service, whereas the far-right column shows the number of services implemented in a tool.

The table reports 29 publicly available tools in total: 20 research prototypes and 9 com-

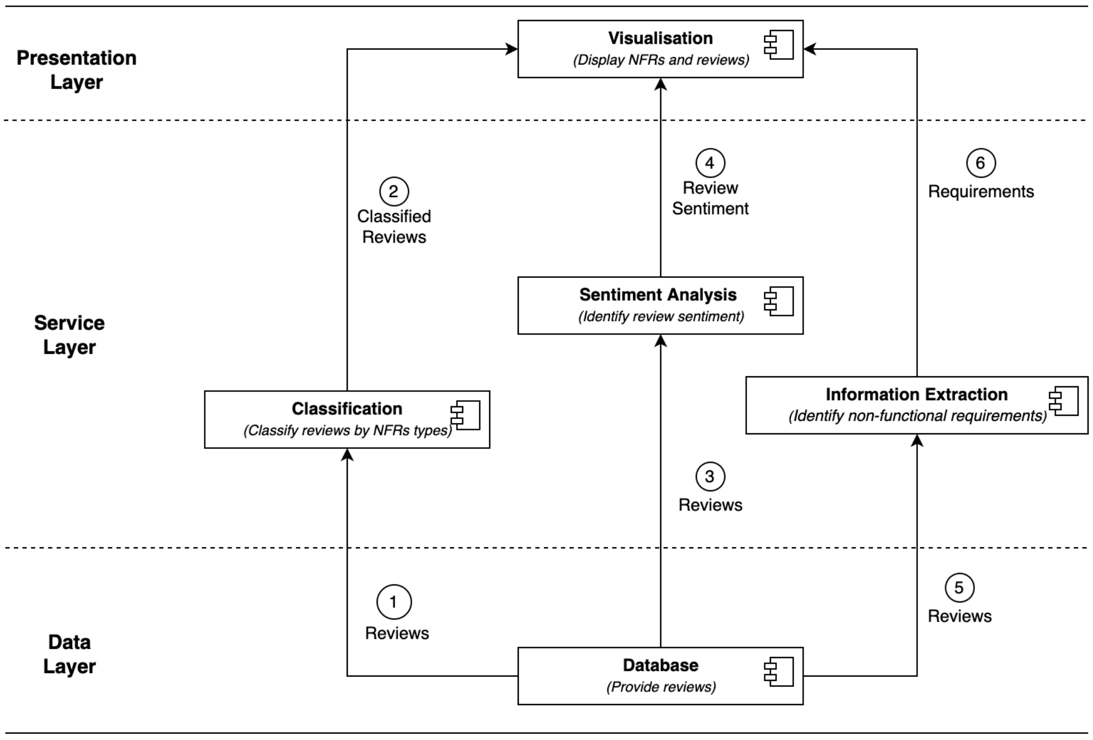


Figure 3.5: Dataflow between architectural components for Analyse NFR Use Case.

mercial ones. Looking at the columns, the results show the number of tools implementing each service ranges from 1 to 19 tools. On average, each service is implemented in 5 tools. The most frequently implemented service is *Classify reviews by request type*, facilitated in 19 tools. The least frequently implemented services are: *Display NFRs analysis*, *Identify NFRs*; each implemented in only one tool. The results show all the services (and thus components) of our architecture are implemented in publicly available tools. When looking at the rows of the table, each tool implements between 1 and 9 services (out of 17), with an average of 3 services per tool. Both commercial tools: Appbot [259] and Applysis [260] implement 9 services - the largest number. Whereas seven research prototypes implement only a single service of the architecture. A closer analysis of these results also indicates most of the tools implement a different set of services; and the partial implementations of our architecture are scattered across these tools. Table 3.5 therefore indicates that the reference architecture provides a suitable generic model for the class of existing app review mining tools in research and industry. This reference architecture is not intended to be a definitive final model but rather to provide the basis for comparing tools and for further refinement and extension.

Table 3.5: Tracability Matrix Mapping Architectural Components and Their Services to Features of Publicly Available Tools.[1]

			Architectural Components																	
			V			SIR			IE			C		CL		S	SA		R	
			Display user opinions analysis	Display user request analysis	Display NFRs analysis	Find feature-related reviews	Find code-specific reviews	Find stack-trace specific reviews	Identify features	Identify user requests	Identify NFRs	Classify reviews by request types	Classify reviews by NFRs	Cluster user requests groups	Cluster user opinions	Summarize reviews	Identify feature-specific sentiment	Identify review sentiment	Suggest review response	**No. Services Implemented in the Tool**
App Review Analysis Tool	**Research**	SAFE [9]				✓			✓											2
		CLAP [151, 18]										✓		✓						2
		MARK [64]				✓			✓											2
		ChangeAdvisor [103]					✓					✓		✓						3
		SURF [79, 65]		✓								✓		✓		✓				4
		MARC [168, 215, 187]									✓	✓	✓			✓				4
		Ardoc [81]										✓						✓		2
		URR [116]					✓					✓				✓				3
		IDEA [16]		✓						✓				✓						3
		RRGen [158]																	✓	1
		OASIS [115]					✓					✓								2
		AOBTM [108]												✓						1
		BECLoMA [113]						✓						✓						2
		Deshpande et al. [211]												✓						1
		Dhinakaran et al. [175]										✓								1
		Scoccia et al. [212]										✓								1
		Shah et al. [176]										✓								1
		Grano et al. [117]						✓				✓								2
		Stanik et al. [165]										✓								1
		Ali et al. [145]										✓						✓		2
	Commercial	Appbot [259]	✓	✓					✓	✓		✓		✓	✓		✓	✓		9
		RankMyApp [261]	✓						✓			✓		✓	✓		✓	✓		7
		WonderFlow [262]	✓	✓	✓				✓			✓	✓				✓			7
		AppAnie [21]	✓						✓								✓			3
		Applysis [260]	✓	✓					✓	✓		✓		✓	✓		✓	✓		9
		AppRadar [263]		✓														✓	✓	3
		AppFollow [264]	✓	✓						✓		✓		✓				✓	✓	7
		AppFigures [265]	✓			✓												✓		3
		Apptopia [266]	✓	✓								✓		✓				✓		5
No. Tools Implementing the Service			8	8	1	3	3	2	7	4	1	19	2	12	3	3	5	9	3	

[1] V stands for Visualization Component; SIR denotes Search and Information Retrieval Component; IE signifies Information Extraction Component; C indicates Classification Component; CL marks Clustering Component; S denotes Summarization Component; SA stands for Sentiment Analysis Component; and R marks Recommendation Component.

3.7 Perspectives

The use cases and reference architecture provide the beginning of a common terminology for researchers and practitioners to discuss and compare their approaches. Identifying the intended use cases of app review analysis techniques and describing the architectural components involved in the realisation of the use cases can help researchers position their work with respect to related work.

Table 3.5 allows researchers to identify what app review analysis have already been implemented in commercial tools (e.g. 'classify review by type'). Commercial implementations provide evidence of the perceived usefulness of the analysis in practice. Commercial tools however do not report the performance of their techniques, and such performance has so far not be evaluated independently. Such a scientific evaluation and a comparison with the corresponding analysis in research tools would be beneficial to researchers, tool vendors, and tool users. Evaluating app review analysis in commercial tools may also enable a better understanding of the real-world contexts and uses cases for the techniques. This would lead to refining and extending the uses cases in Sect. 3.4, which in turn would enable researchers to study how to improve existing techniques or to develop entirely new techniques to better support the use cases.

Table 3.5 also allows researchers to identify techniques currently absent from commercial tools (e.g. the identification of NFRs, the summary of app reviews, and the identification of reviews related to stack traces). This can help researchers to identify opportunities for commercialisation or technology transfer. The lack of commercial implementation may also indicate the envisioned technique is not aligned with real needs or that the technique's performance is not yet sufficient enough to be useful in practice.

Taking the practitioners' perspective, the proposed reference architecture can offer them concrete examples of how feedback mining techniques can be integrated to realise a software engineering use case, thus facilitating their systematic use in practice. Moreover, it can help understand of how to exploit mining techniques into variants of the proposed use case that better fit their own way to develop software.

3.8 Threats to Validity

Internal Validity. A limitation of our reference model is that it relied on our interpretation of the literature for modelling the relations between software engineering goals, use cases and architectural components in our reference model. We have attempted to make the process as objective as possible by following a systematic procedure for constructing empirically-grounded reference architectures [246]. We systematically identified all model elements and their relationship from data extracted from a complete survey of all 182 papers on app review analysis tools published between 2012 and 2020 (see Chapter 2). We then used standard principles to design and represent our reference architecture and use cases [246, 256].

Another threat concerns the usefulness and the completeness of the identified use cases. Regarding their usefulness, the use cases have been identified from a systematic study of the literature. Such study of the literature may not reflect the practitioners' real needs for app review analysis tools. We however argue this threat is marginal as the previous user studies confirmed their usefulness with practitioners [7]. As of their completeness, the use cases are limited to the scenarios envisioned in the literature. We however are not aware about any commercial app review analysis tool for software engineering; the existing commercial tools analyse user feedback mostly for marketing purposes.

External Validity. The main threat of our work is that studies upon we constructed the reference architecture and use cases are not representative. To reduce this threat, we searched and selected the relevant studies using a systematic procedure [236], including in total 182 papers published between 2012 and 2020 (see Chapter 2). Another threat refers to the extent to which the contributions of our study can be applied to other context. The previous surveys showed app review mining techniques and approaches can be used to mine information from other sources of on-line user feedback (e.g., twitter, or online discussion) [23, 267]. We thus argue the result of our study can be generalised these domains too.

3.9 Related Work

We now discuss related work on references architectures in other domains. We compare our study with state-of-the-art using four criteria: research goal, methodology, contributions and limitations.

Several reference architectures have been proposed for different types of systems and domains, for example for big data systems (e.g., [250, 251]), control systems for self-driving vehicles (e.g., [268]), intelligent systems for unmanned vehicle (e.g., [269]), or industrial Internet-of-Things systems (e.g., [270]). Like in our work, the overall purpose of these reference models is to provide a template solution for their domain-specific problems; and to communicate their use cases.

Similarly, the related work presents their reference architectures from the use cases and the logical perspectives [271]. The use cases are presented using a textual description (e.g., [257, 269]), a use case diagram [268] or a class diagram showing the hierarchy of user requirements [272]. However, the related work rarely explains what user‘s goals these use cases help to accomplish but mostly focuses on presenting what end-users can do with a system (e.g., [250, 273]). Differently, our use cases communicate what the users‘s goals are and how these goals can be satisfied when using a system. Like our study, the related work presents the logic of their reference architectures in terms of their static and dynamic views (e.g., [251, 272]). Their static view shows the architectural structure using block or component diagrams, whereas their dynamic view presents the interaction of architectural components using a textual description or data flow diagram.

Similar to our study, related work constructed their references models based on a generalization of a set of existing systems; they have surveyed the literature and collected information to model their reference architecture and use cases [257, 272]. Unlike our work, their methodology was not systematic and lacking details how specific steps have been conducted (e.g., [246, 256]). These works, for example, did not report how they selected the relevant literature (e.g., [257]); how many publications they used (e.g., [251, 272]); nor what information they extracted (e.g., [268]). It is also not clear how these studies used the collected information to construct their reference models. Differently, we followed a systematic methodology to construct an empirically-grounded reference architecture [246]; and detailed each step to justify resulting models.

The related work validated their references architectures using different methods, including a prototype implementation to assess their functional capabilities [268, 270]; interview with stakeholders to assess their usefulness [257]; or by mapping architectural components to available implementations to demonstrate their feasibility (e.g., [257, 272]). Similarly, we validated our architecture using the mapping method as it was not built from scratch but rather constructed using components that have been previously elaborated [246].

In summary, our study differs than related work in terms of their contribution and research methodology. Our study provides the first reference architectures for app review analysis tools in software engineering domain (see Chapter 2). The research methodology for elaborating and validating this architecture is more systematic and rigorous compared to previous studies.

3.10 Conclusion

Mining app reviews can be useful to guide different software engineering activities along requirements, design, maintenance and testing phases. Yet little is known about how to make use of review mining approaches to support software engineering. Existing literature paid superficial attention of software engineering use cases of their approaches.

To address the problem, we have presented a study consolidating the knowledge from a large body of app review mining literature (182 papers, published between January 1, 2012 and December 31, 2020). We provided a thorough synthesis of software engineering use cases and explained how these use cases could potentially support software engineering activities. We then introduced a reference architecture generalizing existing app review mining solutions; and demonstrated how the architecture can realize the use cases. Finally, we validated their feasibility and generalizability by mapping their components to features of publicly available research and commercial tools.

Our synthesis of the software engineering use cases for analysing app reviews will help software engineers and researchers to understand the potential use and benefits of app review analysis tools. The reference architecture consolidates the diversity of research to achieve these benefits; and provides a general framework directing the development and evaluation of future research and tools.

In the following chapters, we present two empirical studies evaluating techniques that can be used to implement parts of our reference architecture. The first study benchmarks techniques for feature identification and feature-specific sentiment analysis; the second study evaluates techniques that support searching for feature-related app reviews.

Chapter 4

Mining User Opinions in App Reviews

This chapter was published in the 32nd International Conference on Advanced Information Systems Engineering 2020 [4]; it won a distinguished paper award. This study and the one in the next chapter are also parts of an extended version submitted to the special issue of Information Systems. The first author‘s contribution to the paper was to formulate the idea, design and execute the experimentation, collect the results, analyse them, and write the manuscripts; other authors of the papers contributed to the research conceptualization and manuscript revision.

4.1 Introduction

App reviews are a rich source of user opinions [7, 19]. These opinions can help developers to understand how users perceive their app, what are users requirements, or what are users preferences [7, 135]. Not surprisingly, knowing user opinions is an important information need developers seek to satisfy [134, 135]. The information can affect different software engineering practices [7, 19].

Analysing app reviews to find user opinions, however, is challenging [1, 19]; Developers may receive thousands of reviews per day [7, 19]. Moreover, these reviews contain mostly noise [1, 2]. Consequently, the possibility of obtaining useful user opinions to support engineering activities is limited [7, 19].

To address the problem, studies in requirement engineering proposed a few opinion mining approaches (e.g., [8, 9]). These approaches facilitate mining user opinions by performing two tasks: extracting features discussed in reviews and identifying their associated users sentiments [8, 274]. In particular, two approaches have become adopted in the community [1], GuMa [8][1] and SAFE [9].

Unfortunately, replicating the studies to confirm their results and to compare their approaches is a challenging problem. In fact, different methods and datasets have been used.

[1]We refer to the approach using abbreviations derived from their authors’ surnames.

The unavailability of their annotated datasets and their evaluation procedures challenges their replicability even more [31, 217].

The aim of the study is to address the problem by extending previous evaluations and performing comparison of these app review analysis approaches. We consider the following research questions to answer:

RQ1: What is the effectiveness of feature extraction approaches?

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

To answer them, we conducted an empirical study in which we evaluated three approaches: GuMa [8], SAFE [9] and ReUS [275]. We evaluated them in performing feature extraction and sentiment analysis tasks using our annotated dataset.

The primary contributions of the study are: (i) an empirical evaluation expanding previous evaluations of the opinion mining approaches, (ii) a comparison of the approaches performing feature extraction and feature-specific sentiment analysis, and (iii) a new dataset of 1,000 reviews annotated with 1,521 opinions [276].

The remainder of the study is structured as follows: In Section 4.2, we introduce terminology and the problem, then we give an overview of the opinion mining approaches we evaluate. In Section 4.3, we present scenarios motivating opinion mining. In Section 4.4, we present our study design. The results are detailed in Section 4.5, and the findings are discussed in Section 4.6. In Section 4.7, we provide threats to validity, then we discuss related works in Section 4.8. Conclusion is given in Section 4.9.

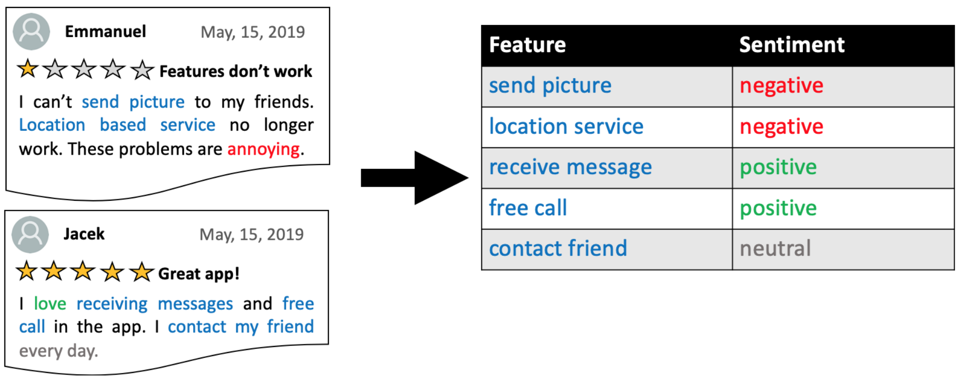
4.2 Background

This section introduces terminology and the formulation of opinion mining problem. It also provides an overview of approaches we evaluated.

4.2.1 Terminology and Problem Formulation

This study defines a *feature* as a user-visible functional attribute of an app: a functionality (e.g., send message), a module providing functional capabilities (e.g., user account) or a design component that can be utilized to perform tasks (e.g., UI). The software engineering literature is generally inconsistent about the feature definition; a part of the literature pertains to features as functional attributes (e.g., [277, 278]), while the other defines features as both functional and non-functional attributes (e.g., [279]).

In this study, the feature definition is focused on functional attributes as the evaluated tools (see Sect. 4.2.2) are neither intended to analyse non-functional attributes (e.g., [9]), nor the studies proposing the tools provide sufficient evidence about their suitability for this pur-


Emmanuel May, 15, 2019
☆☆☆☆☆ Features don't work
I can't send picture to my friends. Location based service no longer work. These problems are annoying.

Jacek May, 15, 2019
☆☆☆☆☆ Great app!
I love receiving messages and free call in the app. I contact my friend every day.

Feature	Sentiment
send picture	negative
location service	negative
receive message	positive
free call	positive
contact friend	neutral

Figure 4.1: Opinion Mining.

pose [8]; in fact, the surveyed literature, in Chapter 2, suggests that custom-built techniques need to be adopted for this purpose (e.g., [5, 187]).

App reviews can describe features seen at a different level of abstraction, at a high-level (e.g., communicate with my friends) and at a low-level one (e.g., click send message button) [8]. A *feature expression* is a non-empty set of words $f = \{w_1, ..., w_m\}$ describing the actual feature in an app review; this definition uses a set of words rather than a multi-set for the feature description as neither our manual analysis of app reviews nor the literature suggests features are described using repeated words. Further on in the text, we will refer to a feature expression as a feature for the sake of simplicity.

Like other types of on-line user feedback [267], app reviews can convey information about user attitude towards features. We here define a *sentiment* s as a user attitude which can be either $positive$, $negative$ or $neutral$; and an *opinion* as a tuple $o = (f, s)$, where f is a feature in a review r, s is a sentiment referencing to f in r.

This study focuses on the *opinion mining* problem, where given a set of reviews $R = \{r\}$ on an app a, the problem is to find a multi-set of all the opinions $O = \{o\}$ in a set of reviews R; this definition refers to a multi-set of all the opinions as the same opinion can be given in many reviews. Figure 4.1 illustrates the opinion mining problem. This problem can be decomposed into two sub-problems, feature extraction and feature-specific sentiment analysis.

The *feature extraction* problem is to find a multi-set of all the features $F = \{f\}$ in a set of reviews $R = \{r\}$ on an app a; whereas, in the *feature-specific sentiment analysis*, given a set of pairs $\{(f, r)\}$ where f is a feature in a review r, the problem is to find a multi-set $S = \{s\}$ where s is a sentiment referring to f in r; these definitions refer to a multi-set of features and a multi-set of feature-specific sentiments as the same feature or the same feature-specific sentiment can be given in many reviews.

4.2.2 Approaches For Mining User Opinions

In our study, we selected three approaches: GuMa [8], SAFE [9] and ReUS [275]. We selected GuMa and SAFE as they are state-of-the-art approaches widely known in requirement engineering research [1, 131]. We opted for ReUS [275] as the approach achieves a competitive performance in the context of opinion mining and sentiment analysis research [274, 275]. We also have its original implementation.

GuMa performs feature extraction and feature-specific sentiment analysis. These tasks are performed independently of each other. To extract features, the approach relies on a collocation finding algorithm; the algorithm identifies expressions of multiple words which commonly co-occur in a set of documents [280]. For predicting sentiment, the approach uses the SentiStrength tool [281]. First, the approach predicts the sentiment of a sentence, then assigns sentiments to features in the sentence. Unfortunately, GuMa's source code and evaluation dataset are not available. We have therefore re-implemented GuMa's approach using SentiStrength for sentiment analysis. We tested that our implementation is consistent with GuMa's original implementation on examples in the original paper and produces the same outputs.

SAFE supports feature extraction, but not sentiment analysis. The approach extracts features based on linguistics patterns, including 18 part-of-speech patterns and 5 sentence patterns. These patterns have been identified through manual analysis of app descriptions. The approach conducts two main steps to extract features from a review: text preprocessing and the application of the patterns. Text preprocessing includes tokenizing a review into sentences, filtering-out noisy sentences, and removing unnecessary words. The final step concerns the application of linguistic patterns to each sentence to extract app features. We used the original implementation of the approach in our study.

ReUS exploits linguistics rules comprised of part-of-speech patterns and semantic dependency relations. These rules are used to parse a sentence and perform feature extraction and feature-specific sentiment analysis. Both tasks are performed at the same time. Given a sentence, the approach extracts a feature and an opinion word conveying a feature-specific sentiment. To determine the sentiment, the approach exploits lexical dictionaries. We used the original implementation of the approach, and set up it to identify one out of three sentiment polarities.

4.3 Motivating Scenarios

We describe three scenarios in which the use of opinion mining can provide benefits. They are inspired by real-world scenarios, which were analysed in previous research (e.g., [7, 19]).

Scenario 1 (Validation by Users) In any business endeavour, understanding customer opinions is an important aspect; app development is no exception [19, 135]. Knowing what features users love or dislike can give project managers an idea about user acceptance of these features [7, 135]. It can also help them draw a conclusion whether invested effort was worth it [19]. As an example, imagine the development team changed core features in WhatsApp (e.g. video call). The team may want to know what users say about these features so that they can fix any glitches as soon as possible and refine these features. Mining user opinions could help them discover What are the most problematic features? or How many users do report negative opinions about a concrete feature (e.g. video call)?

Scenario 2 (Supporting Requirements Elicitation) Imagine now that WhatsApp receives negative comments about one of their features (e.g. group chat). It can be intimidating for developers to tackle a problem if they have to read through a thousand reviews. Using an opinion mining approach, developers could discover the issue within minutes. App mining tools could group reviews based on discussed features and their associated users sentiment. Developers could then examine reviews that talk negatively about a specific feature (e.g. group chat). This could help developers understand user concerns about a problematic feature, and potentially help eliciting new requirements.

Scenario 3 (Supporting Requirements Prioritization) When added with statistics, user opinions can help developers prioritize their work [7, 19]. Suppose the team is aware about problems with certain features which are commented negatively. Finding negative opinions mentioning these features could help them to compare how often these opinions appears, for how long these opinions have been made, and whether their frequency is increasing or decreasing. This information could provide evidence of their relative importance from a users perspective. Such information is not sufficient to prioritize issues, but it can provide useful evidence-based data to contribute to prioritization decisions.

For these scenarios having a tool that (i) mines user opinions and (ii) provides their summary with simple statistics could help the team to evolve their app.

4.4 Empirical Study Design

This section describes the empirical study design we used to evaluate the selected approaches. We provide the research questions we aimed to answer, the manually annotated dataset and evaluation metrics used to this end.

4.4.1 Research Questions

The objective of the study was to evaluate and compare approaches mining opinions from app reviews. To this end, we formulated two research questions:

RQ1: What is the effectiveness of feature extraction approaches?

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

In RQ1, we evaluated the capability of the approaches in correctly extracting features from app reviews. In RQ2, we investigated the degree to which the approaches can correctly predict sentiments associated with specific features. A conclusive method of measuring the correctness of extracted features/predicted sentiments is by relying on human judgment. We used our dataset in which opinions (feature-sentiment pairs) have been annotated by human-coders (see Section 4.4.2). We compared extracted features/predicted sentiments to those annotated in ground truth using automatic matching methods (see Section 4.4.3). In answering the questions, we report precision and recall.

4.4.2 Manually Annotated Dataset

This section describes the manually annotated dataset we created to answer RQ1 and RQ2 [276]. To create this datatset, we collected reviews from previously published datasets [64, 282] and asked human-coders to annotate a selected samples of these reviews.

A) Data Collection

We have selected reviews from datasets used in previous review mining studies [64, 282]. We selected these datasets because they include millions of English reviews from two popular app stores (i.e., Google Play and Amazon) for different apps, categories and period of times. We selected 8 apps from these datasets, 4 apps from Google Play and 4 from Amazon app stores. For each subject app, we also collected their description from the app store. Table 4.1 illustrates the summary of apps and their reviews we used in our study. We selected subject apps from different categories to make our results more generalizable. We believe that the selection of popular apps could help annotators to understand their features, and to reduce their effort during the annotation.

B) Annotation Procedure

The objective of the procedure was to produce an annotated dataset that we use as ground truth to evaluate the quality of solutions produced by feature extraction and sentiment analysis approaches [226]. Figure 4.2 illustrates the overview of the procedure. Given a sample of reviews, the task of human-coders was to label each review with features and their associated sentiments.

Table 4.1: The overview of the subject apps.

App Name	Category	Platform	#Reviews
Evernote	Productivity	Amazon	4,832
Facebook	Social	Amazon	8,293
eBay	Shopping	Amazon	1,962
Netflix	Movies & TV	Amazon	14,310
Spotify Music	Audio & Music	Google Play	14,487
Photo Editor Pro	Photography	Google Play	7,690
Twitter	News & Magazines	Google Play	63,628
Whatsapp	Communication	Google Play	248,641

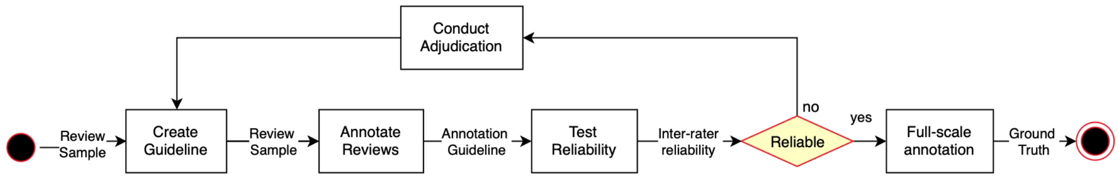


Figure 4.2: The Method for Ground Truth Creation.

We started by elaborating a guideline describing the annotation procedure, the definition of concepts and examples. We then asked two human-coders[2] to label a random sample of reviews using the guideline [276]. We evaluated the reliability of their annotation using the inter-rater agreement metrics F_1 and Fleiss' Kappa [283, 284]. F_1 is suitable for evaluating text spans' annotations such as feature expressions found in reviews; Fleiss kappa is suitable to assess inter-rater reliability between two or more coders for categorical items' annotations such as users' sentiment (positive, negative, or neutral). We evaluated inter-rater agreement to ensure the annotation task was understandable, unambiguous, and could be replicated [226]. When disagreement was found, the annotators discussed to adjudicate their differences and refined the annotation guidelines. The process was performed iteratively, each time with a new sample of reviews until the quality of the annotation was at an acceptable level [283]. Once this was achieved, annotators conducted a full-scale annotation on a new sample of 1,000 reviews that resulted in our ground truth.

C) Ground truth

Table 4.2 reports statistics of our ground truth. These statistics concern subject app reviews, annotated opinions (feature-sentiment pairs) and inter-rater reliability measures. The average length of reviews and sentences is measured in words. Statistics of opinions are reported separately for features and sentiments. The number of features has been given for all the annotated features, distinct ones, and with respect to their length (in words). The

[2]The first author and an external coder who has no relationship with this research. Both coders have an engineering background and programming experience.

Table 4.2: Statistics of the ground truth for 1,000 reviews for 8 subject apps.

		App Name								
		Evernote	Facebook	eBay	Netflix	Spotify	Photo Editor	Twitter	WhatsApp	**Overall**
Reviews	No. reviews	125	125	125	125	125	125	125	125	1,000
	Avg. review length	48.30	37.90	32.54	43.46	23.62	12.38	15.79	14.47	28.59
	No. sentences	367	327	294	341	227	154	183	169	2,062
	Avg. sentence length	16.45	14.49	13.84	15.93	13.00	10.05	10.79	10.70	13.85
	Sentence per review	2.94	2.62	2.35	2.73	1.82	1.23	1.46	1.35	2.06
Sentiment	No. sentiments	295	242	206	262	180	96	122	118	1,521
	No. positive	97	49	95	79	32	39	5	20	416
	No. neutral	189	168	102	159	122	47	93	84	964
	No. negative	9	25	9	24	26	10	24	14	141
Features	No. features	295	242	206	262	180	96	122	118	1,521
	No. distinct features	259	204	167	201	145	80	99	100	1,172
	No. single-word features	82	80	78	94	69	39	39	49	530
	No. multi-word features	213	162	128	168	111	57	83	69	991
	Feature per review	2.36	1.94	1.65	2.10	1.44	0.77	0.98	0.94	1,52
Agrmt.	F_1 measure	0.76	0.73	0.77	0.75	0.67	0.78	0.79	0.83	0.76
	Fleiss' Kappa	0.64	0.77	0.77	0.55	0.75	0.86	0.69	0.80	0.73

number of sentiments has been described including their number per polarity.

The ground truth consists of 1,000 reviews for 8 subject apps. In total, 1,521 opinions (i.e., feature-sentiment pairs) have been annotated. Their sentiment distribution is unbalanced: most feature-sentiment pairs are neutral. Among 1,521 annotated features, 1,172 of them are distinct (i.e. mentioned only once).

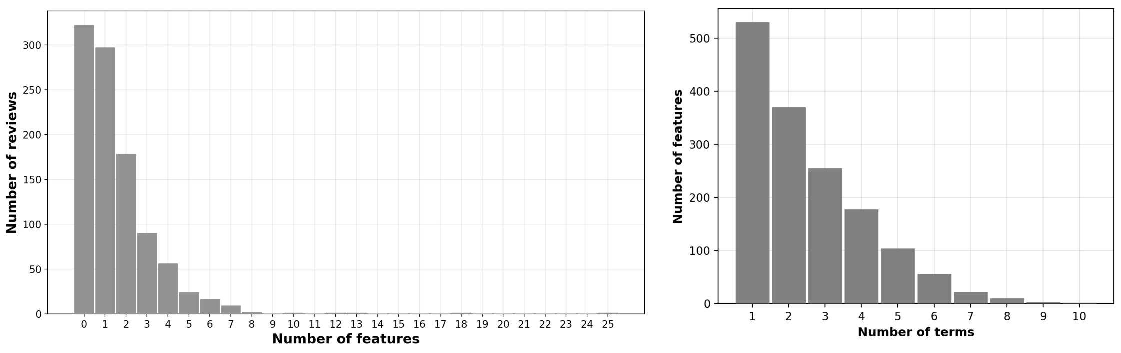
The feature distribution in app reviews can be found in Figure 4.3a. A large number of reviews do not refer to any specific feature. 75% of reviews refers to no feature or to only one or two features. Figure 4.3b provides the feature length distribution. The median length for a feature is 2 words, 75% of features has between 1 and 3 words, and nearly 5% has more than 5 words.

4.4.3 Evaluation Metrics

We used precision and recall metrics [283] to answer RQ1 and RQ2. We used them because feature extraction is an instance of information extraction problem [283], whereas sentiment analysis can be seen as a classification problem [274].

A) Evaluation Metrics For Feature Extraction

In answering RQ1, precision indicates the percentage of extracted features that are true positives. Recall refers to the percentage of annotated features that were extracted. An ex-



(a) Feature distribution in app reviews. **(b)** Distribution of feature length.

Figure 4.3: Feature distribution in app reviews, and Feature length distribution.

tracted feature can be true or false positive. True positive features correspond to features that were both extracted and annotated; False positives are features that were extracted but not annotated; Annotated but not extracted features are called false negative. To determine whether an extracted feature is true or false positive, we compared them with annotated features in the ground truth. To this end, we used the following *feature matching method*:

Let Γ be the set of words in a review sentence and $f_i \subseteq \Gamma$ be the set of words used to refer to feature i in that sentence. Two features $f_1, f_2 \subseteq \Gamma$ match at level n (with $n \in \mathbb{N}$) if and only if (i) one of the feature is equal to or is a subset of the other, i.e. $f_1 \subseteq f_2$ or $f_2 \subseteq f_1$, and (ii) the absolute length difference between the features is at most n, i.e. $||f_1| - |f_2|| \leq n$.

B) Evaluation Metrics For Feature-Specific Sentiment Analysis

In answering RQ2, precision indicates the percentage of predicted sentiments that are correct. Recall refers to the percentage of annotated sentiments that are predicted correctly. To determine whether predicted sentiments are correct, we compared them with annotated ones in the ground truth.

We measured precision and recall for each polarity category (i.e. positive, neutral and negative). We also calculated the overall precision and recall of all three sentiment polarities. To this end, we used the weighted average of precision and recall of each polarity category. The weight of a given polarity category was determined by the number of annotated sentiments with the sentiment polarity.

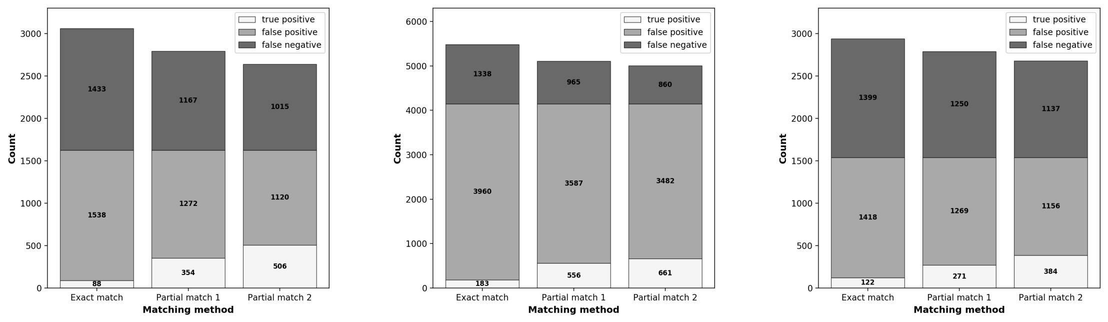
4.5 Results

RQ1: What is the effectiveness of feature extraction approaches?

To answer RQ1, we compared extracted features to our ground truth using *feature matching method* at levels 0, 1 and 2 (see Sect. 4.4.3). We selected these levels as extracted and

Table 4.3: RQ1. Results for feature extraction at varied levels of feature matching.

	Exact Match (n=0)						**Partial Match 1 (n=1)**						**Partial Match 2 (n=2)**					
	GuMa		**SAFE**		**ReUS**		**GuMa**		**SAFE**		**ReUS**		**GuMa**		**SAFE**		**ReUS**	
App Name	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
Evernote	0.06	**0.13**	**0.07**	0.08	**0.07**	0.08	0.15	**0.35**	**0.22**	0.24	0.19	0.20	0.17	**0.39**	**0.32**	0.35	0.27	0.29
Facebook	0.03	0.07	0.03	0.03	**0.09**	**0.09**	0.10	**0.28**	**0.15**	0.17	**0.15**	0.14	0.13	**0.36**	**0.23**	0.26	0.20	0.19
eBay	0.04	**0.07**	0.04	0.05	**0.06**	0.06	0.14	**0.26**	**0.22**	**0.26**	0.14	0.14	0.17	0.32	**0.34**	**0.39**	0.22	0.21
Netflix	0.03	**0.13**	0.03	0.03	**0.06**	0.07	0.11	**0.45**	**0.19**	0.21	0.18	0.21	0.13	**0.55**	**0.27**	0.29	0.25	0.29
Spotify	0.05	0.10	0.05	0.04	**0.15**	**0.13**	0.18	**0.37**	**0.24**	0.23	0.23	0.20	0.21	**0.43**	**0.36**	0.34	0.29	0.26
Photo Editor	0.12	0.11	0.12	0.09	**0.14**	**0.13**	0.26	0.25	**0.34**	**0.27**	0.23	0.21	0.29	0.27	**0.38**	**0.30**	0.27	0.25
Twitter	**0.06**	**0.19**	**0.06**	0.07	0.02	0.02	0.16	**0.49**	**0.23**	0.24	0.11	0.11	0.18	**0.58**	**0.35**	0.36	0.27	0.26
WhatsApp	0.05	**0.21**	**0.11**	0.11	0.06	0.06	0.14	**0.56**	**0.32**	0.33	0.19	0.20	0.16	**0.64**	**0.39**	0.40	0.24	0.25
Mean	0.05	**0.13**	0.06	0.06	**0.08**	0.08	0.15	**0.37**	**0.24**	0.24	0.18	0.18	0.18	**0.44**	**0.33**	0.34	0.25	0.25



(a) SAFE **(b)** GuMa **(c)** ReUS

Figure 4.4: RQ1. No. TPs, FPs and FNs as the level of features matching changes.

annotated features may differ by a few words but still indicating the same app feature. We then computed precision and recall at these levels. Table 4.3 reports precision and recall for each approach at different matching levels (best in bold). The results show the approaches achieved low precision, recall given Exact Match. For all three approaches, precision and recall increase when we loosen the matching criteria to partial matching with $n = 1$ or 2. The growth can be attributed to the changed numbers of true positives (TPs), false positives (FPs) and false negatives (FNs) when n increases. Figures 4.4 shows their behavior as the matching level n increases; $\Delta TPs = -\Delta FPs = -\Delta FNs$ when n increases.

RQ2: What is the effectiveness of feature-specific sentiment analysis approaches?

In answering RQ2, we report the effectiveness of ReUS and GuMa in feature-specific sentiment (see Section 4.4.3). To this end, we compared predicted and annotated sentiments, and exploited a subset of the ground truth with opinions (feature-sentiment pairs) we used to answer RQ1. Indeed, since ReUS predicts sentiments only for extracted features, we considered only true positive features obtained in answering RQ1 and formed three datasets, each corresponding to true positive features (and their sentiment) from Exact Match, Partial Match$_1$ and Partial Match$_2$. Table 4.4 reports for each dataset the total number of opinions, and their

Table 4.4: RQ2. Dataset used for evaluating feature-specific sentiment analysis.

Dataset	# opinions	# positive	# neutral	# negative
Exact Match	122	56	52	14
Partial Match 1	271	97	149	25
Partial Match 2	384	120	226	38
All Annotated	1521	416	964	141

Table 4.5: RQ2. Results for feature-specific sentiment analysis (overall).

Dataset	Approach	# correct prediction	Precision	Recall
Exact Match	ReUS	**85**	**0.74**	**0.70**
	GuMa	77	0.65	0.63
Partial Match 1	ReUS	**184**	0.69	**0.68**
	GuMa	176	**0.72**	0.65
Partial Match 2	ReUS	**265**	0.69	**0.69**
	GuMa	252	**0.73**	0.66
All Annotated	ReUS	-	-	-
	GuMa	**958**	**0.73**	**0.63**

breakdown to polarity categories. We also evaluated GuMa with these datasets and with all the annotated opinions in our ground truth.

The answer to RQ2 can be given at two levels of details, the overall effectiveness of predicting a sentiment, and the effectiveness of predicting a specific polarity (e.g., positive). We report our results at both levels of details.

Overall effectiveness. Table 4.5 reports the number of correct predictions, and weighted precision/recall for inferring overall sentiment (best in bold). We can observe that ReUS achieves higher precision and recall than GuMa for Exact Match dataset, whereas both approaches have similar performances on the Partial Match$_1$ and Partial Match$_2$ datasets.

Specific effectiveness. In Table 4.6, we report the metrics showing the effectiveness of the approaches in predicting specific polarities (best in bold). The results show that on positive opinions ReUS achieves higher precision while suffering from lower recall. Conversely, on neutral opinions GuMa provides better precision but lower recall than ReUS. When looking at the approaches, the analysis of the results revealed that none of the approaches was able to reliably assess the sentiment of negative opinions. Both approaches were good at discriminating between positive and negative opinions. Most incorrect predictions were caused by misclassifying positive/negative sentiment with neutral one and vice versa.

Table 4.6: RQ2. Results for feature-specific sentiment analysis (per each polarity).

		Positive			Neutral			Negative		
Dataset	**Approach**	**# correct prediction**	**Precision**	**Recall**	**# correct prediction**	**Precision**	**Recall**	**# correct prediction**	**Precision**	**Recall**
Exact Match	ReUS	35	**0.90**	0.62	**45**	0.60	**0.87**	5	**0.62**	0.36
	GuMa	**47**	0.68	**0.84**	21	**0.68**	0.40	**9**	0.41	**0.64**
Partial Match 1	ReUS	47	**0.80**	0.48	**131**	0.66	**0.88**	6	**0.43**	0.24
	GuMa	**86**	0.61	**0.89**	73	**0.85**	0.49	**17**	0.40	**0.68**
Partial Match 2	ReUS	53	**0.80**	0.44	**205**	0.68	**0.91**	7	**0.41**	0.18
	GuMa	**107**	0.59	**0.89**	122	**0.86**	0.54	**23**	0.38	**0.61**
All Annotated	ReUS	-	-	-	-	-	-	-	-	-
	GuMa	**355**	**0.52**	**0.85**	**510**	**0.87**	**0.53**	**93**	**0.36**	**0.66**

4.6 Discussion

The results indicate that the approaches have limited effectiveness in mining user opinions. Our findings bring into question their practical applications.

A) Feature Extraction

In our experiment, feature extraction methods have lower precision and recall than previously reported. SAFE was reported with 0.71 recall [9]. Our results show the approach achieves 0.34 recall for the least demanding evaluation strategy. The majority of features extracted by GuMa are incorrect. Although GuMa initially reported precision and recall of 0.58 and 0.52 [8], our experiment found lower figures of 0.18 precision and 0.44 recall.

Although the difference may be due to our re-implementation of the GuMa method, we have taken great care in implementing the method as described in the paper as rigorously as possible. Unfortunately, the original GuMa implementation was not available for comparison. We believe ReUS suffered from low precision and recall because it was designed to extract features from product reviews in an online commerce website (Amazon) rather than from app reviews in app stores [274, 275]. Our findings support a conjecture that the original evaluation procedures of SAFE and GuMa led to over-optimistic results. The limitations of these procedures have been questioned recently [31, 217]. These procedures did not define a feature matching strategy [31], relied on a subjective judgment [9, 217], and used a biased dataset [31, 217]. We hope our new annotated dataset and description of our evaluation method will contribute to improving the quality of feature extraction techniques and their evaluations.

B) Feature-Specific Sentiment Analysis

Our investigation of results (RQ2) concludes that the overall effectiveness of the approaches is promising (see Table 4.5). However, it reveals that their precision and recall differ considerably by sentiment class (positive, negative, or neutral). The approaches provide satisfactory performance for predicting positive and neutral sentiments. But they suffer from inaccurate predictions for negative sentiments. Overall, we are surprised by the comparable effectiveness of both approaches. We expected ReUS to outperform GuMa. ReUS exploits a sophisticated technique to detect an opinion word in a sentence that carries a feature-specific sentiment; GuMa makes predictions based on a simplified premise that a feature-specific sentiment corresponds to the overall sentiment of a sentence.

C) Implication on Requirement Engineering Practices

Identifying what precision and recall app review mining techniques should have to be useful for requirements engineers in practice is an important open question [285]. In principle, a tool facilitating opinion mining should synthesize reviews so that the effort for their further manual analysis would be negligible or at least manageable. Clearly, this effort depends on a scenario the approach intends to support. Given a scenario of prioritizing problematic features, a developer may seek for information about the number of specific features that received negative comments, for example to understand their relevance. To this end, both information about extracted features and their predicted sentiments should be accurate and complete. Our results, however, show that feature extraction techniques generate many false positives. Given the large number of extracted features, filtering out false positives manually may not be cost-effective. We may imagine that the problem could be partially addressed using a searching tool [110]; Requirements engineers could use the tool to filter out uninteresting features (including false positives) and focus on those of their interest.

However, other issues remain unsolved. Feature extraction techniques fail to identify many references to features (they have low recall), and sentiment analysis techniques perform poorly for identifying feature-specific negative sentiments.

4.7 Threats to Validity

Internal Validity. The main threat is that the annotation of reviews was done manually with a certain level of subjectivity and reliability. To overcome the risk we followed a systematic procedure to create our ground truth. We prepared an annotation guideline with definitions and examples. We conducted several trial runs followed by resolutions of any conflicts. Finally, we evaluated the quality of the annotation using inter-rater agreement metrics.

External Validity. To mitigate the threat, we selected reviews for popular apps belonging to different categories and various app stores. These reviews are written using varied vocabulary. We, however, admit that the eight apps in our study represent a tiny proportion of all the apps in the app market. Although our dataset is comparable in size to datasets in previous studies (e.g., [8, 9]), we are also exposed to sampling bias.

Construct Validity. The main threat is the extent to which our operationalization of a feature,

Table 4.7: The summarized differences between our study and related works.

	Criterion	**Our Study**	**SAFE** [9]	**GuMa** [8]	**ReUS** [275]
Evaluation	No. Approaches	**3**	2	1	1
	Feature Extraction	**Yes**	**Yes**	No	**Yes**
	Sentiment Analysis	**Yes**	-	**Yes**	**Yes**
	Method	**Automatic**	Manual	Manual	**Automatic**
Ground Truth	Released	**Yes**	No	No	No
	No. Apps	**8**	5	7	-
	No. Reviews	1000	80	**2800**	1000
	No. App Stores	**2**	1	**2**	-
	Dataset Analysis	**Yes**	No	No	**Yes**

a sentiment and a user opinion reflects the actual constructs under study [286]. To mitigate the threat, we first defined their conceptual meaning referring to the requirement engineering and opinion mining literature [274, 287]. We then chose standard variables to represent each concept in a text document: a bounded textual phrase referring to a mentioned feature; the polarity of a review sentence where the phrase appears referring to the user's associated sentiment; and their pair referring to a user opinion. To ensure the conceptual meaning and the operational definitions are understandable, we discussed them with an independent panel of researchers, including experts in requirements engineering and natural language processing. To verify the reliability of the operationalization, we tasked two human-coders to annotate a sample of app reviews using the operational definitions, and then checked their inter-rater agreement is of sufficient quality.

4.8 Related Works

Previous works have proposed benchmarks for app review analytics (e.g. [31, 288]) but with objective different than ours.

Table 4.7 shows the differences between our study and previous works, pointing out the different criteria that guided the evaluations, which are grouped into Evaluation and Ground Truth categories. The first includes criteria such as the number of evaluated approaches, evaluated tasks and a method type used for their evaluation. The latter includes characteristics of datasets.

In our study, we evaluated three approaches: SAFE, GuMa and ReUS. We assessed them in addressing problems of feature extraction and sentiment analysis. Johann et al [9] also compared SAFE to GuMa [8]. Our study extends their evaluation by including ReUS [275]. Unlike the original study [8], we evaluated GuMa in performing feature extraction rather than modeling feature topics. We also compared the approach to ReUS in inferring a feature-specific sentiment.

We used a different methodology for evaluating SAFE and GuMa [8, 9]; The correctness of their solutions has been evaluated manually [8, 9]. The judgement criteria, however, has not been defined. Such a procedure suffered from several threats to validity such as human error, authors' bias and the lack of repeatability [217]. To address the limitations, we adopted automatic matching methods and defined explicit matching criteria.

The ground truth in our study differs from that used in previous works. Unlike Dragoni et al. [275], we evaluated ReUS using app reviews. The authors used a dataset composed of comments for restaurant and laptops. As Johann [9] and Guzman [8], we created an annotated dataset for the evaluation. We, however, used a systematic procedure and assessed the quality of ground truth using acknowledged measures [226, 274]. Previous studies did not report a systematic annotation procedure [9] nor measured the quality of their annotation [8]. Their datasets were not analyzed nor made public [8, 9].

4.9 Conclusion

Mining user opinions from app reviews can be useful to guide requirement engineering activities such as user validation [2, 7], requirements elicitation [7, 135], or requirement prioritization [7]. However, the performance of app review mining techniques and their ability to support these tasks in practice are still unknown.

We have presented an empirical study aimed at evaluating existing opinion mining techniques for app reviews. We have evaluated three approaches: SAFE [9] relying on part-of-speech parsing, GuMa [8] adopting a collocation-based algorithm, and ReUS [275] exploiting a syntactic dependency-based parser. We have created a new dataset of 1,000 reviews from which 1,521 opinions are specific features were manually annotated. We then used this dataset to evaluate the feature identification capabilities of all three approaches and the sentiment analysis capabilities of GuMa and ReUS.

Our study indicates that feature extraction techniques are not yet effective enough to be used in practice [31, 131] and that have lower precision and recall than reported in their initial studies. Our study also indicates that feature-specific sentiment analysis techniques have limited precision and recall, particularly for negative sentiments. We hope our novel annotated dataset [276] and evaluation method will contribute to improving the quality of app review mining techniques.

Chapter 5

Searching for App Reviews Related to a Specific Feature

This chapter and the previous one are parts of a journal submission to the Special Issue of Information Systems. The first author‘s contribution to the paper was to formulate the idea, design and execute the experimentation, collect the results, analyse them, and write the manuscripts; other authors of the papers contributed to the research conceptualization and manuscript revision.

5.1 Introduction

Features are at the heart of mobile app development [289]; they are functional app attributes that deliver values to end-users [4, 7]; they contribute to software product differentiation; and they are one of determinants leading to the app success [7]. Knowing what users say about features is thus an important developers‘ concern [7, 19].

Around 40% of app reviews contains feature-related information [289], including user opinions [4], usage scenarios [2], or different types of user requests [1]. Surveyed developers reported that this information can be useful for different SE activities [3, 7]; however, obtaining it is challenging due to the large number and noisy character of app reviews [1, 7].

The literature survey, in Chapter 2, presented automatic approaches facilitating ‘feature-specific analysis’ that addresses this challenge. These approaches facilitate the analysis by performing one of the following tasks, or a combination of them: mining user opinions [4] and searching reviews that refer to a specific feature [110].

Unfortunately, the effectiveness of the approaches in performing these tasks has been evaluated using different methods and datasets [217]. Replicating these studies to confirm their results and to provide benchmarks of different approaches is a challenging problem [31].

To address the problem, Chapter 4 presented a study extending previous evaluations and performing comparison of opinion mining approaches. The opining mining approaches extract phrases referring to app features in app reviews and identify their associated users' sentiments [4]; then output user opinions consisting of pairs of feature and sentiment.

In this Chapter, we present a study evaluating and benchmarking the performance of approaches searching for app reviews that refer to a specific feature. Differently than opinion mining tools, the searching approaches take as an input a feature description as a query and attempts to find all the reviews related to that query.

We consider the following research question to answer in this study:

RQ1: What is the effectiveness of approaches in searching for app reviews pertinent to a particular feature?

To answer the question, we conducted an empirical study in which we evaluated three approaches: Lucene [290], MARAM [13] and SAFE [9]. We evaluated them in performing the task of searching for app reviews pertinent to a particular feature using a new annotated dataset we created for this study.

The main contributions of this study are: (i) an empirical evaluation expanding previous evaluations of approaches searching for feature-related reviews; (ii) a comparison of the approaches in performing this task, and (iii) a new dataset of 1,113 reviews annotated with 24 app features [291].

The chapter is structured as follows: Section 5.2 introduces terminology, defines the problems of searching for feature-related reviews, and gives an overview of approaches we evaluate. Section 5.3 presents scenarios motivating these approaches. Section 5.4 presents the empirical study design. The results are detailed in Section 5.5, and the findings are discussed in Section 5.6. In Section 5.7 we provide threats to validity, then we discuss related works in Section 5.8. Conclusion is given in Section 5.9.

5.2 Background

This section introduces terminology and the formulation of the searching for feature-related reviews problem. It also provides an overview of approaches we evaluated.

5.2.1 Terminology and Problem Formulation

This study defines a *feature* as a user-visible functional attribute of an app: a functionality (e.g., send message), a module providing functional capabilities (e.g., user account) or a design component that can be utilized to perform tasks (e.g., UI). The software engineering literature is generally inconsistent about the feature definition; a part of the literature pertains to features as functional attributes (e.g., [277, 278]), while the other defines features as both functional and non-functional attributes (e.g., [279]).

In this study, the feature definition is focused on functional attributes as the evaluated tools (see Sect. 5.2.2) are neither intended to analyse non-functional attributes (e.g., [9]), nor the studies proposing the tools provide sufficient evidence about their suitability for this purpose [13]; in fact, the surveyed literature, in Chapter 2, suggests that custom-built techniques need to be adopted for this purpose (e.g., [5, 187]).

App reviews can describe features seen at a different level of abstraction, at a high-level (e.g., communicate with my friends) and at a low-level one (e.g., click send message button) [8]. A *feature expression* is a non-empty set of words $f = \{w_1, ..., w_m\}$ describing the actual feature in an app review; this definition uses a set of words rather than a multi-set for the feature description as neither our manual analysis of app reviews nor the literature suggests features are described using repeated words. Further on in the text, we will refer to a feature expression as a feature for the sake of simplicity.

This study focuses on the problem of *searching for feature-related reviews*, where given a set of reviews $R = \{r\}$ on an app a and a feature, the problem is to find the subset of reviews in R that refer to this feature (i.e., feature-related reviews).

5.2.2 Approaches Searching for Feature-Related Reviews

We chose three approaches for the empirical evaluation: SAFE [9], MARAM [13] and Lucene [290]. We selected SAFE and MARAM as they are the only proposed approaches searching for feature-related reviews in the app store research [110]. We also included Lucene, a well-known general purpose search engine, as it is commonly used as a baseline in the empirical evaluation of SE research (see Chapter 2).

SAFE supports searching for feature-related reviews using NLP techniques. The approach first extracts features based on linguistics patterns, including 18 part-of-speech patterns and 5 sentence patterns, that have been identified through manual analysis of app descriptions. Then the approach compares queried app features with those extracted from reviews using semantic similarity. To improve the performance, the tool uses query expansion using WordNet lexical database. We used the original implementation of the tool in our study.

MARAM supports the task of searching for feature-related reviews. To this end, the approach exploits a simplistic model by representing both query and reviews as the bags (sets) of their words. It then computes the similarity score between a given query and reviews using Jaccard Similarity coefficient. The tool next outputs ranked reviews based on their similarity score with the query. Neither MARAM tool nor their source code are available. We have therefore re-implemented the approach rigorously following their description in the original study. We have also tested that our implementation is consistent with MARAM's original implementation on examples in the original paper and produces the same outputs.

Lucene is a free and open-source search engine software library suitable for any application requiring full-text indexing and searching capability [290]. It is widely known for its usefulness in the implementation of internet search engines and local, single-site searching. Lucene combines Vector Space Model and the Boolean Model to determine how relevant a given document is to a user's query. We implemented a basic app review searching tool that uses Lucene with its default setting.

5.3 Motivating Scenarios

We describe three scenarios in which the use of approaches searching for feature-related reviews can provide benefits. They are inspired by real-world scenarios, which were analysed in previous research (e.g., [7, 19]).

Scenario 1 (Supporting Change Request Prioritization) Suppose the issue tracking system for a messaging app (like WhatsApp) contains change requests for introducing new features and improving existing ones (e.g., the ability to receive custom notifications, the ability to group video call, etc.) and the product manager must decide which requests to implement first. Finding app reviews mentioning each of these features could help the product manager to compare the number of reviews referring to each request; for how long each request has been made, and whether the frequency of reviews referring to each request is increasing or decreasing [7, 110]. This information can provide concrete evidence of the relative importance of each request from the users' perspective [6]. Such information alone would not be sufficient to prioritize change requests as the perspective of other stakeholders must also be taken into consideration, but it can give evidence-based knowledge to partially support such decisions.

Scenario 2 (Supporting Requirements Elicitation) Suppose now a requirement engineer has been tasked to define and document requirements for one of the requested features (e.g., receive custom notifications). Finding users reviews that refer to the feature will allow them to quickly identify what users have been saying about the feature; and how to evolve the feature to best serve the users' needs [6, 110]. This cheap elicitation technique might be sufficient in itself or it might be the starting point for additional more expensive elicitation activities involving interviews, surveys, prototyping, or observations.

Scenario 3 (Supporting Impact Analysis) Imagine now the WhatsApp team have a tool notifying developers of what user are saying about a feature. Software developers may subscribe to this tool to be notified of all new reviews that mention features they have worked on or are interested in. Such notifications would help developers see the impacts of their work on users [3]. Positive feedback would be gratifying, and negative and mixed feedback would help them gain a better understanding of their users and of the feature limitations [3].

For these scenarios having a tool that supports searching for feature-related reviews could help the team to evolve their app.

5.4 Empirical Study Design

In this section, the research question we aimed to answer is presented, together with the collected and manually annotated dataset, the evaluation procedure as well as evaluation metrics we used in answering the question.

5.4.1 Research Questions

The objective of the study was to evaluate and compare approaches searching for feature-related app reviews. To this end, we formulated the following research question:
RQ1: What is the effectiveness of approaches in searching for app reviews pertinent to a particular feature?

For RQ1, we investigated the degree to which the selected approaches can correctly find reviews discussing concrete features of interest. A conclusive method of measuring the correctness of finding feature-related reviews is by relying on a human judgment; we first tasked human-coders to formulate a set of queries (i.e., app features) based on descriptions of selected apps. We next fed the subject approaches with these queries and a set of reviews. The human coders then evaluated the top-n reviews retrieved by the approaches for the queries. In answering RQ1, we report precision@n, average precision and relative recall for each approach.

5.4.2 Manually Annotated Dataset

To create the dataset we used to answer RQ1, we collected reviews and app descriptions for subject apps presented in our first empirical study (see Table 4.1). We then asked human-coders to identify queries (i.e., app features) from these descriptions. We next fed the approaches with these queries; then tasked the coders to annotate a samples of retrieved reviews with respect to the queries. The resulting dataset is publicly available [291].

A) Collected Dataset

We used the same dataset we had collected in the first study, including reviews and descriptions for 8 popular apps from different categories (see Table 4.1). We then randomly sampled 1,250 reviews per app (in total 10,000 reviews) as we strove to obtain a representative review sample. Table 5.1 reports the summary of our dataset, indicating the apps and the

Table 5.1: The overview of the selected apps and collected reviews.

App Name	Category	Platform	#Reviews
Evernote	Productivity	Amazon	1,250
Facebook	Social	Amazon	1,250
eBay	Shopping	Amazon	1,250
Netflix	Movies & TV	Amazon	1,250
Spotify Music	Audio & Music	Google Play	1,250
Photo Editor Pro	Photography	Google Play	1,250
Twitter	News & Magazines	Google Play	1,250
Whatsapp	Communication	Google Play	1,250

size of review samples we used.

B) Query Set

We built a query set based on the collected app descriptions conveying information about an app and their features. We first tasked two human-coders[1] to identify app features from the descriptions following a systematic procedure analogous to the one used in the experiment for feature identification from app reviews (see Sec. 4.4.2). Since human-coders had experience with an annotation process and a common understanding of app feature from the study (see Sec. 4.4.2), they conducted the feature annotation without pre-training. We however validated their inter-rater agreement to ensure their annotation was reliable and sufficient quality [226]. We evaluated the inter-rater using agreement F_1 metric as it is suitable for evaluating text spans annotations such as feature expressions found in app descriptions [283, 284]. Table 5.2 reports the statistics of the subject app descriptions, annotated features and inter-rater reliability measures. The complete list of the identified features can be found in the supplementary materials [291]. In total, 124 app features (candidate queries) have been annotated in the app descriptions. The average inter-rater agreement between coders was 0.78, indicating the substantial reliability of the annotation [283]. The length of the identified features ranges between 1 and 5 words, with the average of 2 words.

To form the evaluation query set for our experiment, we randomly selected 3 features for each app (in total 24 features). We selected this number as we wanted to obtain a broad and diverse set of queries for our evaluation [284]. Table 5.3 shows details of these queries. We used this query set in the review annotation procedure. We fed the approaches with these queries, then the human-coders assessed their outcomes.

C) Review Annotation Procedure

We used the pooling method to evaluate the performance of the selected approaches

[1]We tasked the same human-coders that the annotated dataset in the first study (see Chapter 4).

Table 5.2: Statistics of the identified features from app descriptions for 8 subject apps.

		App Name								
		Evernote	Facebook	eBay	Netflix	Spotify	Photo Editor	Twitter	WhatsApp	**Overall**
Description	No. words	500	72	424	264	144	195	269	355	2,223
	No. sentences	33	8	39	18	14	21	28	20	181
	Avg. sentence length	15.15	9.00	10.87	14.67	10.29	9.29	9.61	17.75	12.08
	No. paragraphs	8	2	13	6	6	7	9	12	63
Features	No. features	18	7	24	10	10	19	21	15	124
	Min. feature length	1	1	2	2	2	1	1	1	1
	Max. feature length	4	4	5	3	5	4	4	5	5
	Avg. feature length	2.06	2.14	2.50	2.50	3.00	2.11	2.33	2.40	2.07
	No. single-word features	4	2	0	0	0	3	3	1	13
	No. multi-word features	14	5	24	10	10	16	18	14	111
Agr.	Fleiss' Kappa	0.82	0.88	0.75	0.68	0.67	0.84	0.70	0.89	0.78

Table 5.3: The set of query for the empirical evaluation.

App Name	Id	Query	App Name	Id	Query
	1	Create shortcuts		13	Play playlist on shuffle mode
Evernote	2	Write notes	Spotify	14	Create playlist of songs
	3	Annotate documents		15	Offline listening
	4	Chat		16	Edit a photo
Facebook	5	Share	Photo Editor	17	Photo filters
	6	Watch videos		18	Build photo collage
	7	Bid item		19	Twitter Moment
eBay	8	Search for offer on item	Twitter	20	Follow people on Twitter
	9	List items for sale		21	Write a Tweet
	10	Rate movies		22	Notifications
Netflix	11	Search for titles	WhatsApp	22	WhatsApp call
	12	Watch movies		24	Send messages

in finding feature-related reviews [284]. We opted for this method as it is commonly used by researchers for evaluating approaches addressing the information retrieval problem [284, 292]; in particular, when the assessment of their results is limited due to the large size of evaluation dataset. The problem of finding feature-related reviews can be seen as an instance of the general information retrieval problem.

In this method, the top-n reviews (with n = 20) from the rankings obtained by the evaluated approaches are merged into a pool, duplicates are removed, and the reviews are presented in a random order to human-coders annotating their relevance with regards to input queries. We selected the top-20 reviews as this level of the pool's depth is recommended in the information retrieval literature [167]; it reduces the number of documents that human-coders need to annotate and enables to calculate stable values of evaluation metrics. Figure 5.1 illus-

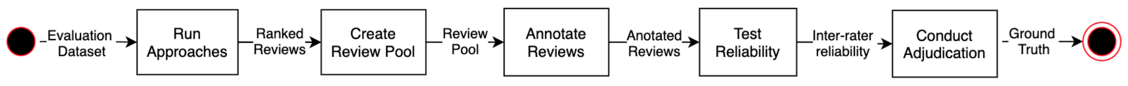


Figure 5.1: The Method for Review Pool Creation.

trates the overview of the procedure. We first inputted the selected approaches with a query and a review sample. We then obtained ranked results of top-n reviews from each evaluated approach (n = 20 in our study). We merged their results into a pool of unique reviews and removed duplicates. The review pool was next presented in a random order to the coders who annotated it with the query set. Each assessor judged the relevance of the reviews with respect to input query; a review was classified as relevant if it refers to the queried feature, and irrelevant otherwise. We measured their inter-rater agreement to assess the task was understandable, unambiguous, and could be replicated. We employed Fleiss' Kappa to this end as it is suitable for evaluating inter-rater reliability between two or more coders for categorical items annotations such as a review's relevance [283, 284]. The quality of the annotation was always at least at substantial level [283]. Therefore, the coders discussed the minor differences in their annotations, adjudicated them and provided an annotated dataset, comprising of a query and a pool of respectively annotated reviews. We repeated the method for each query and corresponding new review sample. Having the annotated reviews for all the queries, we obtained the ground truth that we used to assess the performance of the approaches. Table 5.4 reports the statistics of the ground truth. These statistics concern annotated reviews and inter-rater reliability measures. The number of reviews is reported in relation to a concrete query, indicating relevant reviews (query-related) and non-relevant (remaining). In total, 1,113 reviews have been annotated with respect to 24 queries. Among 1,113 annotated reviews, 512 of them are relevant and 601 are non-relevant. On average, 46 reviews have been annotated per a query. The number of relevant reviews ranges between 2 and 54, with the mean of 21 reviews; whereas the number of non-relevant reviews pear query is between 4 and 49, with the mean of 25 reviews. The inter-rater agreement indicates the substantial reliability of the dataset [283, 284].

5.4.3 Evaluation Metrics

We used precision@n, average precision and relative recall metrics [284, 292] to answer RQ3. We used them because searching for feature-related reviews is an instance of information retrieval problem [284]. Precision@n indicates the percentage of the top-n retrieved reviews

Table 5.4: Statistics of the ground truth, indicating no. reviews in relation to concrete queries.

	App Name											
		Evernote			Facebook			eBay			Netflix	
Query Id	1	2	3	4	5	6	7	8	9	10	11	12
No. reviews	47	51	45	46	45	36	48	48	48	55	42	58
No. relevant	2	40	4	23	24	31	39	25	21	6	16	54
No. non-relevant	45	11	41	23	21	5	9	23	27	49	26	4
Fleiss' Kappa	1.00	0.84	1.00	1.00	1.00	1.00	1.00	0.92	1.00	1.00	0.88	0.88

	App Name												
		Evernote			Photo Editor			Twitter			WhatsApp		**Overall**
Query Id	13	14	15	16	17	18	19	20	21	22	23	24	24
No. reviews	46	43	47	40	45	48	47	47	49	39	48	45	1,113
No. relevant	21	13	16	31	15	9	2	18	15	13	43	31	512
No. non-relevant	25	30	31	9	30	39	45	29	34	26	5	14	601
Fleiss' Kappa	1.00	1.00	1.00	1.00	1.00	1.00	0.66	1.00	0.95	0.89	1.00	0.83	0.95

that are relevant [292]. The metric is useful for assessing the searching task in finding the most relevant documents at a given rank. Average precision summarizes the ranking of top-n retrieved reviews by averaging the precision@n values from the rank positions where a relevant review was retrieved [284]. The metric is based on the ranking of all the relevant reviews, but their value depends heavily on the highly ranked relevant reviews. It is thus a suitable measure for evaluating the task of finding as many relevant reviews as possible while still reflecting the intuition that the top-ranked reviews are the most important. Relative recall is the proportion of known relevant reviews that has been retrieved in top-n results [292]. The metrics provide partial knowledge about the completeness of retrieved results. To determine whether retrieved reviews are relevant, we compared them with the annotated ones in the ground truth. We also used it to determine the known relevant reviews.

5.5 Results

In answering RQ3, we report the effectiveness of Lucene, MARAM and SAFE in finding feature-related reviews. To this end, we compared the top-20 retrieved reviews to our ground truth. We selected this level of depth as it is typically recommended in the literature [284, 292]. Table 5.5 reports precision@20 (P@20), average precision (AP) and relative recall (RR) for each approach (best in bold). The results show the approaches achieved substantially different performance; for example, P@20 and RR for Lucene was over twice as high as for SAFE. Lucene generally scored the best results given all three metrics; whereas MARAM achieved

Table 5.5: RQ3. Results of the evaluated approaches for finding feature-related reviews.

	Lucene			**MARAM**			**SAFE**		
App Name	P@20	AP	RR	P@20	AP	RR	P@20	AP	RR
Evernote	**0.42**	**0.83**	**0.67**	0.32	0.41	0.47	0.25	0.61	0.36
Facebook	**0.97**	**0.99**	**0.75**	0.85	0.98	0.66	0.32	0.51	0.22
eBay	0.73	0.82	0.52	**0.82**	**0.90**	**0.59**	0.37	0.62	0.25
Netflix	**0.57**	0.70	**0.62**	0.47	**0.82**	0.32	0.37	0.74	0.19
Spotify	**0.58**	**0.85**	**0.70**	0.42	0.45	0.49	0.23	0.32	0.29
Photo Editor	**0.68**	**0.99**	**0.80**	0.33	0.68	0.24	0.38	0.27	0.38
Twitter	**0.47**	0.66	**0.70**	0.32	**0.77**	0.39	0.12	0.44	0.29
Twitter	**0.85**	**0.99**	**0.68**	0.70	0.94	0.56	0.53	0.75	0.33
Mean	**0.66**	**0.85**	**0.68**	0.53	0.74	0.46	0.32	0.53	0.29

the second best results. Lucene achieved the precision of 0.66 in returning relevant results among the top-20 retrieved reviews; MARAM, on average, scored P@20 of 0.53. While looking at their average precision, we can also observe the respective approaches showed the effectiveness of 0.85 and 0.74 in ranking most relevant reviews in the top of their results. Whereas, relative recall suggests the approaches retrieved a substantial portion of the relevant reviews: Lucene reached relative recall of 0.68 while MARAM scored 0.46. Conversely, we observe that SAFE achieved the lowest performance given all three metrics. On average, one-third of the reviews returned by SAFE is relevant to queried features; the average precision of 0.53 indicates these reviews were scattered over the ranked results. In our experiment, SAFE's relative recall ranges from 0.19 to 0.38. This result can be attributed to the poor effectiveness of the tool in performing feature extraction (see Sec. 4.5); the outcome of this task affect the input of the actual task of finding feature-related reviews (see Sec. 5.2).

5.6 Discussion

The results shed a new light on the effectiveness of the evaluated approaches. Our findings suggest the approaches can be useful for practical applications.

A) Finding Feature-Related Reviews

In our experiment, the three approaches achieved diverse effectiveness. The results suggest SAFE achieves lower performance than previously reported. The approach was reported with 0.70 precision and 0.56 recall [9]. Our results show the approach achieves 0.32 precision@20, 0.53 average precision and 0.29 recall rate. We argue the discrepancy between the original results and ours cannot be attributed to the evaluation metrics. We rather hypothesize the original evaluation procedure led to over-optimistic results. The limitations of the procedure were questioned recently [31, 217]. The procedures relied on a subjective judgment [9, 217], biased and small evaluation dataset [31, 217]. We have taken great care to overcome the limitations in our study; we employed much larger and diverse dataset; then followed rigorous evaluation procedure. We thus argue the strength of evidence is generally larger in our study.

Our findings shed a new light on the usefulness of MARAM and Lucene. Though they have never been empirically evaluated for finding feature-related reviews (see Chapter 2), the approaches achieved much better performance than SAFE. We took great caution in implementing MARAM as described in the original study as rigorously as possible; similarly, we implemented Lucene following their documentation to make the most of it [290]. The most striking result to emerge from the data, however, is that Lucene, a standard search engine library, achieved the best effectiveness. Importantly, the overall run-time of this tool, including the time of all the intermediate steps like indexing reviews, text processing and responding to a query was between 2 and 5 seconds. In contrast, it took between 3 and 5 minutes for MARAM and up to 2 hours for SAFE to complete their execution. This observation suggests the purpose-built tools would be of limited use in the practical settings where the number of reviews is substantially larger than in our experiment (1,250 reviews per app); popular apps like WhatsApp can receive hundreds of thousands reviews per month. Practitioners would not accept an app review analytics tool whose searching feature needs several days for generating the output. In our opinion, these results also provide useful evidence to researchers aiming at developing new mining techniques about the opportunity to exploit existing searching techniques; and pay more attention to the efficiency of their approaches. We believe our new annotated dataset and the evaluation procedure will help to improve the quality of these techniques evaluation.

B) Implication on Requirement Engineering Practices

The results suggest Lucene tool can be useful for requirements engineering use cases. Supposing requirements engineers have been tasked to detail requirements about concrete features, the tool could help them to quickly identify what users have been saying about the features. Though the engineers would need to filter-out a few unrelated reviews, the results indicate most of them, in particular, those highly ranked one would be of their interest. This cheap elicitation technique might be not sufficient in itself, but can be of great value when used in combination with other techniques e.g., for classifying user feedback [293]. Integrating the techniques could help the engineers to further distill reviews reporting problems about these features, requesting improvements or discussing their quality attribute. As for the usefulness of the tool for requirements prioritization, the engineers would seek for the information about the concrete number of feature-related comments that are negative or report a certain type of requests. Future studies thus should investigate the tool's performance for outputting the set of reviews. It would require studying the cut-off value of the tool's similarity measure to discriminate feature-related reviews from unrelated ones.

5.7 Threats To Validity

Internal Validity. The main threat is that the annotation of both: app description and reviews was done manually with a certain level of subjectivity and reliability. To mitigate the threat, we employed a systematic procedure to create our evaluation dataset. We prepared an annotation guideline with definitions and examples. We then evaluated the quality of the annotation task using standard inter-rater agreement metrics [283, 284]. Another limitation concerns the lack of evaluation of 'the absolute' precision and recall. We only estimated their 'relative' values using the annotated sample of app reviews that the tools had outputted; we did not annotate the complete set of collected app reviews to prepare the evaluation dataset. Identifying relevant app reviews to a particular query from a collection of thousands or millions of app reviews, like in our dataset, is a non-trivial task. A single human-coder, for instance, would require 83 hours to judge the collection of 10,000 reviews for a single query (30 sec/review). Precision and recall can thus be estimated only. We used the standard pooling method to prepare our annotated dataset and estimate the metrics [167]. We admit the use of this method cannot provide the complete knowledge about 'the absolute' precision and recall [284]; and their actual values remain unknown. Estimating their 'relative' values is however still useful; it helps to benchmark the tools and to obtain the partial knowledge about their performance.

External Validity. To mitigate the threat, we selected apps belonging to diverse categories and different app stores. Their descriptions as well as user feedback has diverse characteristics: different length, varied vocabulary and refereed to different app features. We, however, admit that our dataset comprises a tiny fraction of all the apps in the app market. Though its size is much greater compared to previous studies [9, 110], we are also exposed to sampling bias [61].

Construct Validity. A potential threat to our study is the extent to which our feature operationalization reflects the actual construct under study [286]. To reduce this threat, we defined the conceptual meaning of the feature referring to the requirement engineering literature [274, 287]; we then chose the standard variable representing the concept in a textual document: a bounded textual phrase referring to a feature mentioned in an app review. We further confirmed the meaning and the operationalization are understandable to external requirements engineering and natural language processing scholars. To ensure the opera-

Table 5.6: The differences between our study and previous empirical evaluations for RQ3.

Criterion	**Our Study**	**SAFE** [9]	**MARAM** [13]	**JaDa**^{2} [110]
No. Approaches	**3**	1	0	1
Validation	**Yes**	**Yes**	No	**Yes**
Dataset Relased	**Yes**	**Yes**	-	No
Ground Truth	**Yes**	No	-	No
Np. Apps	**8**	5	-	1
No. Reviews	**1,113**	-	-	200
No. Queries	20	**178**	-	20

tionalization was reliable, we checked the inter-rater agreement of human-coders annotating sample reviews with the concept was of sufficient quality.

5.8 Related Works

More than 182 papers in the area of app review analysis have been published in the last decade (see Chapter 2), but only three of them investigated the use of techniques for task of searching for feature-related reviews [9, 13, 110]. Previous works however had different objective than in this study: they focused on developing new approaches for finding feature-related reviews, whereas this study focuses on a more extensive evaluation of these approaches and their comparison to a general purpose searching technique [290]. We here discuss the difference between our study and related works based on aspects concerning an empirical evaluation. Table 5.6 shows the differences between our study and the previous works, pointing out the different criteria guiding this discussion.

Our empirical study evaluated and compared the effectiveness of three approaches: SAFE [9], Lucene [290] and MARAM [13]. The related works focused only on proposing and/or evaluating their searching approaches without bench-marking them with the existing techniques (e.g., [290]). Similarly like SAFE's authors, we also released our dataset publicly. Most importantly, we elaborated and facilitated the first dataset, consisting of annotated reviews with queries. Previous studies either evaluated their approach without such dataset [9], or have not facilitated it for the public scrutiny [110]. The scale of the empirical evaluation also favours our study; in particular in terms of the number of app reviews; In our previous work [110] we elaborated only 200 reviews for a single app with 20 exemplary queries.

5.9 Conclusion

Searching for app reviews related to a specific feature can be useful to guide requirement engineering activities such as user validation [7, 135], requirements elicitation [7, 135], or requirement prioritization [7]. However, the performance of app review mining techniques and their ability to support these tasks in practice are still unknown. We have presented an empirical study aimed at evaluating techniques searching for feature-related reviews.

In the study, we have evaluated three approaches in searching for feature-related reviews: Lucene [290], a search engine library relying on vector space model and Okapi BM25 similarity; MARAM [13] adopting Jaccard Similarity; and SAFE, exploiting part-of-speech parsing and semantic similarity measure [9]. With human-coders' help, we elaborated a novel evaluation dataset, including 1,113 annotated reviews with respect to 24 queries (app features). We used the dataset to evaluate the approaches in searching for feature-related reviews. The findings showed Lucene, a standard searching tool, provides better performance than state-of-the-art techniques developed for app review analysis. We concluded the tool provides promising accuracy, and could be potentially used to support requirements elicitation. We suggest future studies focus on extending existing techniques; and pay more attention to the efficiency of their approaches.

We hope our novel annotated dataset [291] and the evaluation methods will contribute to improving the quality of app review mining techniques.

Chapter 6

Conclusions and Future Works

App review analysis has become an active research area in the software engineering community; app reviews are a rich source of information that can guide different activities in software development life cycle. Unfortunately, analysing these app reviews manually is challenging due to their large number and the difficulty in extracting actionable information from the short unstructured text. A large body of previous research has studied what type of information can be found in app review and how to extract such information automatically. Yet little is known about how analysing app reviews support software engineering activities. The overall thesis objective was to study this problem, while addressing the limitations of state-of-the-art. The thesis provides several contributions to advance this research; and recommends future research directions summarized bellow.

6.1 Summary of Contributions

The major contributions of this thesis are summarised as follows.

6.1.1 A Literature Survey on App Review Analysis for Software Engineering

In Chapter 2, we have presented a systematic literature review of research on app review analysis since the birth of the field in 2012 until December 2020. They key findings from our analysis of 182 papers are:

1. The literature suggests app review analysis can support 14 different types of software engineering activities related to requirement, design, testing and maintenance.

2. 9 broad types of app review analyses are used to support software engineering activities: (1) information extraction; (2) classification; (3) clustering; (4) search and information retrieval; (5) sentiment analysis; (6) content analysis; (7) recommendation; (8) summarization and (9) visualization.

3. The literature employs 4 broad types of techniques to realise app review analysis: (1)

manual analysis; (2) natural language processing; (3) machine learning and (4) statistical analysis.

4. Software engineers find app review analysis useful; it may reduce their workload, ease their activities, and support decision-making. The accuracy of mining techniques is promising, yet little is known to what extent it is sufficient for practical use.

5. Existing literature provides little discussion about software engineering use cases of their mining approaches. This challenges the understanding of their usefulness and their application.

6. Research in this area of app review analysis is currently of inconsistent quality in terms of an evaluation method and reproducibility; most papers did not make available their app review analysis tools and evaluation datasets.

6.1.2 SE Use Cases and Reference Architecture for Mining App Reviews

In Chapter 3, we synthesised software engineering use cases for mining app reviews that have been envisioned in the literature; and defined a novel reference architecture realising these use cases through a combination of machine learning and natural language processing techniques. We then validated the feasibility of the architecture by mapping their components to the implementations of available tools. The key contributions of this study are:

1. The use cases summarize the different ways in which app review analysis can be used in practice and what software engineering activities they support. These use cases can be used by software engineers to understand the potential value of app review analysis tools, they can also be used by developers of app review analysis tools to improve their tools and by researchers to evaluate app review analysis techniques with respect to specific software engineering use cases and goals.

2. The reference architecture communicates how integrating different components helps to realise intended use cases and satisfy stakeholders‘ goals. The model can help to identify and reuse common components for a typical app review mining tool and explain their novelty and contribution within that framework.

3. The mapping between reference architecture components and features of existing, publicly available tools shows the feasibility of this architecture; it helps to identify what app review analysis has already been implemented in commercial tools; and provides

evidence of their perceived usefulness in practice; it can also help to identify techniques currently absent from commercial tools and indicate opportunities for commercialisation.

6.1.3 Empirical Study of Opinion Mining Approaches

In Chapter 4, we conducted an empirical study of three opinion mining approaches; we evaluated and compared SAFE [9] relying on part-of-speech parsing, GuMa [62] adopting a collocation-based algorithm, and ReUS [275] exploiting a syntactic dependency-based parser. We assessed their effectiveness in performing feature extraction and sentiment analysis using a new annotated dataset. The key contributions of this study are:

1. An empirical study extending previous evaluations of opinion mining approaches; it follows more rigorous and systematic evaluation methods compared to the previous works.
2. A comparison of three opinion mining approaches; the results show feature extraction and sentiment analysis techniques are not yet effective enough for a practical application as they have lower precision and recall than reported in their initial studies.
3. A new dataset consisting of 1,000 reviews annotated which 1,521 opinions; pairs of features and users‘ associated sentiment were manually identified and labeled; the dataset can help to improve the quality of app review mining techniques.

6.1.4 Empirical Study of Approaches Searching for Feature-Related Reviews

In Chapter 4, we have empirically evaluated three approaches addressing the problem of finding feature-specific reviews: Lucene, MARAM and SAFE. We evaluated the approaches using a new annotated dataset that we created for this study; we first tasked human-annotators to form a set of queries (i.e., app features) based on descriptions of selected apps. We then fed the approaches with these queries and a set of reviews. The annotators then assessed top-n outputted reviews from each approach with respect to the queries. The key contributions of this study are:

1. An empirical study evaluating approaches searching for app reviews related to a specific feature. Differently than previous works, the study employs standard methods for information retrieval system evaluation; and assesses two approaches that have not been previously evaluated in this context.
2. A comparison of three approaches searching for app reviews related to a specific feature; the results reveal SAFE achieves lower performance than originally reported; the other tools, which have never been evaluated, are more effective. The results suggest Lucene, a standard search engine library, can be useful for requirements engineering.

3. A new dataset consisting of 1,113 app reviews annotated with 24 features; it can help future works in evaluating approaches searching for app reviews related to a specific feature.

6.2 Summary of Future Works

This section summarizes future work identified in the previous chapters.

6.2.1 Conduct More Replication and Comparison Studies

Replication studies are invaluable in the empirical software engineering [294]; a new knowledge coming from experimental results need to be verified. The literature survey, in Chapter 2, however found replication studies are still rare in app review analysis research; only two replication studies have been published, including our study [4, 31]. The thesis partially addressed the problem by extending previous evaluations of opinion mining (see Chapter 4) and searching techniques (see Chapter 5). The effectiveness of several important types of app review analysis has not been yet confirmed, but comes from individual studies e.g., for summarising reviews [215]; extracting NFRs [5]; or finding review-code links [103]. The strength of their results is limited; and additional replication studies can advance our knowledge on their actual performance. We recommend future works to re-use publicly available datasets and tools identified in our literature survey (see Chapter 2); and focus on addressing the limitations of their original studies; future works can for example extend the original evaluations by benchmarking different techniques; or elaborating new more rigorous evaluation procedures and evaluation datasets. Not only such an extension will bring out the novelty of these studies, but it will also make them very useful for SE community.

6.2.2 Evaluate Run-time Efficiency of App Review Mining Techniques

Run-time efficiency is a crucial quality of information systems [295]. Existing literature, however, does not study how much time app review mining tools and techniques need to facilitate their intended analyses; nor how the time changes when they are inputted with growing number of reviews (see Chapter 2). Consequently, understanding their usefulness in the industrial settings is limited. Our experiments recorded opinion mining and searching techniques require substantial time to analyse a relatively small size of reviews; for example, feature extraction tools like SAFE required from 2 to 6 hours to identify features in 1,200 reviews; similarly, it took between several dozen minutes up to several hours for the searching tools to find feature-related reviews. Popular apps like WhatsApp or Instagram however receive more than 5,000 reviews per days. Our observation thus brings the usefulness of these techniques into question. We recommend future studies to evaluate run-time efficiency of existing app review mining techniques; as well as to take this criterion into account when developing new ones.

6.2.3 Evaluate Analytics Tools Against Software Engineering's Concerns

The literature survey, in Chapter 2, has found that most empirical evaluations assess app review mining techniques in terms of ML effectiveness metrics (e.g., precision and recall); how the techniques address software engineering concerns has not been main focus of the studies. It is therefore important to evaluate the techniques in terms of software engineering concerns: Do the techniques improve the quality of, for instance, the requirements elicitation or requirements prioritisation? Do the techniques save time to perform SE activities? etc. Evaluating the techniques with respect to software engineering concerns is more challenging but crucial to make sure research the effort is aligned with real stakeholders' goals. Such an evaluation will employ quantitative and qualitative research methods to reduce the current uncertainty about potential impacts of app review mining techniques on software engineering activities.

6.2.4 Improve Accuracy of Opinion Mining Techniques

Opinion mining techniques are inaccurate in feature extraction and feature-specific sentiment analysis (see Chapter 4). Majority of extracted features are false positives; and more than a half of true positives are unidentified. To improve their accuracy, future works can build a dictionary with a list of already known features (so-called Gazetteer) and integrate it with the techniques. Such dictionary could be used to verify their output; or lookup for unidentified features. Future works can also experiment with supervised deep learning techniques showing splendid accuracy over unsupervised ones [296]. To avoid the problem of preparing a new training dataset, one can train a supervised technique on an existing dataset from different domains; then exploit domain adaptation techniques to tune the techniques to the target domain. Regarding feature-specific sentiment analysis, the accuracy of the evaluated techniques, in particular for negative sentiment, is low. Similarly as previous studies [297], our result analysis suggests the proposed techniques are not adopted to SE domain [297]. Future works can adopt the techniques for the language specificity of app reviews; or experiment with existing tools that have been SE-specific tuned [297, 298].

6.2.5 Improve Accuracy of Technique Searching for Feature-Related Reviews

The results, in Chapter 5, suggest future works could improve the accuracy of approaches searching for feature-related reviews. The performance of standard tools like Lucene is promising and could be boosted using query expansion or relevance feedback methods [167]. App reviews frequently refer to the queried app features using different words; query expansion could help to retrieve such reviews, and thus improve the recall rate, by enriching an initial query with a semantically similar words. Relevance feedback, on the other hand, engages a tool's user in the retrieval process; the user could give feedback on the relevance of initially retrieved documents; such feedback would be then used to improve the final result.

6.2.6 Using and Evolving the Use Cases and Reference Architecture

In Chapter 3, we hypothesized the use cases and the architecture are useful for practitioners to communicate the benefits of mining app reviews and how these benefits can be realised. Future studies can validate this hypothesis using a combination of qualitative and quantitative methods. These works could for example, interview practitioners to investigate who is the actual beneficiary of the use cases? Do the use cases satisfy their goals? What are their perceived usefulness and importance? Are these use case complete? Not only would it help to validate the practical aspect of the use cases; but more importantly it will provide a new knowledge helping to scope and justify the future research directions in app review analysis research, as well as provide insights to their commercializion. As of the reference architecture, the model can help researchers to identify techniques currently absent from commercial tools (see Table 3.5); future work can study whether these techniques are not aligned with real stakeholder‘s needs or that their performance is not yet sufficient to be useful in practice. We also anticipate the future research will extend existing tools with new features that have not been yet envisioned in the literature like (e.g., prioritizing user requirements or generating a report of bug reports); and improve the automaton of the overall app review analysis process; existing tools are missing a sufficient automation as their outputs still require a manual inspection to distill target information (e.g., user requirements). Our model can help scholars to identify these gaps; and communicate their novel contributions within this model.

Bibliography

[1] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, 43(9):817–847, 2017.

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

[3] Simon van Oordt and Emitza Guzman. On the Role of User Feedback in Software Evolution: a Practitioners' Perspective. In *2021 29th IEEE International Requirements Engineering Conference (RE)*, pages 221–231, September 2021.

[4] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. Mining user opinions to support requirement engineering: An empirical study. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, volume 12127 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 2020.

[5] E. C. Groen, S. Kopczyska, M. P. Hauer, T. D. Krafft, and J. Doerr. Users the hidden software product quality experts?: A study on how app users report quality aspects in online reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 80–89, Sep. 2017.

[6] Marlo Haering, Christoph Stanik, and Walid Maalej. Automatically matching bug reports with related app reviews. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 970–981, 2021.

[7] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman. App store effects on software engineering practices. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

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

[9] T. Johann, C. Stanik, A. M. A. B., and W. Maalej. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 21–30, Sep. 2017.

[10] Felwah Alqahtani and Rita Orji. Usability issues in mental health applications. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization*, UMAP19 Adjunct, page 343348, New York, NY, USA, 2019. ACM.

[11] Daniel Franzmann, Arvid Eichner, and Roland Holten. How mobile app design overhauls can be disastrous in terms of user perception: The case of snapchat. *Trans. Soc. Comput.*, 3(4), September 2020.

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

[13] Claudia Iacob, Shamal Faily, and Rachel Harrison. Maram: Tool support for mobile app review management. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, MobiCASE16, page 4250. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

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

[15] Walid Maalej, Zijad Kurtanovic, Hadeer Nabil, and Christoph Stanik. On the automatic classification of app reviews. *Requir. Eng.*, 21(3):311–331, 2016.

[16] Cuiyun Gao, Jichuan Zeng, Michael R. Lyu, and Irwin King. Online app review analysis for identifying emerging issues. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE 18, page 4858, New York, NY, USA, 2018. ACM.

[17] Cuiyun Gao, Wujie Zheng, Yuetang Deng, David Lo, Jichuan Zeng, Michael R. Lyu, and Irwin King. Emerging app issue identification from user feedback: Experience on wechat. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP 19, page 279288. IEEE Press, 2019.

[18] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 14–24, May 2016.

[19] Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, and Barbara Paech. How do practitioners capture and utilize user feedback during continuous software engineering? In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 153–164. IEEE, 2019.

[20] Eduard C. Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, and Melanie Stade. The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 34(2):44–52, 2017.

[21] App Annie. `https://www.appannie.com/`, 2020. Accessed: 2020-07-01.

[22] Mohammadali Tavakoli, Liping Zhao, Atefeh Heydari, and Goran Nenadi. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. *Expert Systems with Applications*, 113:186 – 199, 2018.

[23] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol-Valeriu Chioasca, and Riza T. Batista-Navarro. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv.*, 54(3), apr 2021.

[24] Sachiko Lim, Aron Henriksson, and Jelena Zdravkovic. Data-driven requirements elicitation: A systematic literature review. *SN Computer Science*, 2, 02 2021.

[25] Necmiye Genc-Nayebi and Alain Abran. A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software*, 125:207 – 219, 2017.

[26] Assem Al-Hawari, Hassan Najadat, and Raed Shatnawi. Classification of application reviews into software maintenance tasks using data mining techniques. *Software Quality Journal*, Aug 2020.

[27] Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, and Irwin King. Infar: Insight extraction from app reviews. In *Proceedings of the 2018 26th ACM Joint*

Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, page 904907, New York, NY, USA, 2018. ACM.

[28] Fatemeh Hemmatian and Mohammad Karim Sohrabi. A survey on classification techniques for opinion mining and sentiment analysis. *Artificial Intelligence Review*, Dec 2017.

[29] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 767778, New York, NY, USA, 2014. ACM.

[30] Mario Sänger, Ulf Leser, and Roman Klinger. Fine-grained opinion mining from mobile app reviews with word embedding features. In *Natural Language Processing and Information Systems - 22nd International Conference on Applications of Natural Language to Information Systems, NLDB 2017, Liège, Belgium, June 21-23, 2017, Proceedings*, pages 3–14, 2017.

[31] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Is the SAFE approach too simple for app feature extraction? A replication study. In *Requirements Engineering: Foundation for Software Quality - 25th International Working Conference, REFSQ 2019, Essen, Germany, March 18-21, 2019, Proceedings*, pages 21–36, 2019.

[32] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. Analysing app reviews for software engineering: a systematic literature review. *Empir. Softw. Eng.*, 27(2):43, 2022.

[33] Jessica Clement. Number of apps available in leading app stores as of 1st quarter 2020. `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`, 2020. Accessed: 2020-07-01.

[34] App Store. Ratings, Reviews, and Responses. `https://developer.apple.com/app-store/ratings-and-reviews/`, 2021. Accessed: 2021-06-01.

[35] Rifat Ara Shams, Waqar Hussain, Gillian Oliver, Arif Nurwidyantoro, Harsha Perera, and Jon Whittle. Society-oriented applications development: Investigating users' values from bangladeshi agriculture mobile applications. In *Proceedings of the ACM/IEEE*

42nd International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS '20, page 5362, New York, NY, USA, 2020. Association for Computing Machinery.

[36] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. A little bird told me: Mining tweets for requirements and software evolution. In Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech, editors, *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, pages 11–20. IEEE Computer Society, 2017.

[37] Javed Khan, Yuchen Xie, Lin Liu, and Lijie Wen. Analysis of requirements-related arguments in user forums. 11 2019.

[38] Ally Nyamawe, Hui Liu, Nan Niu, Qasim Umer, and Zhendong Niu. Automated recommendation of software refactorings based on feature requests. pages 187–198, 09 2019.

[39] Ehsan Noei and Kelly Lyons. A survey of utilizing user-reviews posted on google play store. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, CASCON 19, page 5463, USA, 2019. IBM Corp.

[40] Barbara A. Kitchenham. Procedures for performing systematic reviews. 2004.

[41] Pierre Bourque, Robert Dupuis, Alain Abran, James Moore, and Leonard Tripp. The guide to the software engineering body of knowledge. *IEEE Software*, 16:35–44, 12 1999.

[42] D. Moher, A. Liberati, Jennifer Tetzlaff, and Douglas Altman. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *Br Med J*, 8:336–341, 01 2009.

[43] M. Bujang and N. Baharum. Guidelines of the minimum sample size requirements for kappa agreement test. *Epidemiology, biostatistics, and public health*, 14, 2017.

[44] Anthony J Viera and Joanne Mills Garrett. Understanding interobserver agreement: the kappa statistic. *Family medicine*, 37 5:360–3, 2005.

[45] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on*

Evaluation and Assessment in Software Engineering, EASE 14, New York, NY, USA, 2014. ACM.

[46] Wiley. Step by step guide to reviewing a manuscript. `https://authorservices.wiley.com/Reviewers/journal-reviewers/how-to-perform-a-peer-review/step-by-step-guide-to-reviewing-a-manuscript.html`, April 2022.

[47] Nancy Ide and James Pustejovsky, editors. *Handbook of Linguistic Annotation*. Springer Netherlands, 2017.

[48] M. Graham, Anthony T. Milanowski, and Jackson Miller. Measuring and promoting inter-rater agreement of teacher and principal performance ratings. 2012.

[49] Martin Bauer. Content analysis. an introduction to its methodology by klaus krippendorff from words to numbers. narrative, data and social science by roberto franzosi. *The British Journal of Sociology*, 58:329 – 331, 07 2007.

[50] Mario Cannataro and Carmela Comito. A data mining ontology for grid programming. In *Proc. 1st Int. Workshop on semantics in Peer-to-Peer And Grid Computing, In Conjunction with WWW2003*, pages 113–134, 2003.

[51] Gary Miner, John Elder, Thomas Hill, Robert Nisbet, Dursun Delen, and Andrew Fast. *Practical Text Mining and Statistical Analysis for Non-Structured Text Data Applications*. Academic Press, Inc., USA, 1st edition, 2012.

[52] Vivek Singh. South Asian University - Department of Computer Science. `http://www.sau.int/research-themes/text-analytics.html`, 2021. Accessed: 2021-06-01.

[53] TIBCO Software. What is Text Analytics? `http://www.tibco.com/reference-center/what-is-text-analytics`, 2021. Accessed: 2021-06-01.

[54] Jacek Dąbrowski. Supplementary material for System Literature Review: Analysing App Reviews for Software Engineering. `https://github.com/jsdabrowski/SLR-SE/`, February 2021.

[55] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 241–244. ACM, 2012.

[56] Laura V. G. Carreño and Kristina Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE 13, page 582591. IEEE Press, 2013.

[57] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 13, page 12761284, New York, NY, USA, 2013. ACM.

[58] Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 41–44. IEEE Press, 2013.

[59] Emitza Guzman, Muhammad El-Halaby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE 15, page 771776. IEEE Press, 2015.

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

[61] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. The app sampling problem for app store mining. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR 15, page 123133. IEEE Press, 2015.

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

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

[64] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE 15, page 749459. IEEE Press, 2015.

[65] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 499510, New York, NY, USA, 2016. ACM.

[66] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, Jan 2016.

[67] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E. Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, 21(3):1067–1106, 2016.

[68] Ying Wang, Liwei Zheng, and Ning Li. Rom: A requirement opinions mining method preliminary try based on software review data. In *Proceedings of the 2020 4th International Conference on Management Engineering, Software Engineering and Service Sciences*, ICMSS 2020, page 2633, New York, NY, USA, 2020. Association for Computing Machinery.

[69] Y. Wang, H. Wang, and H. Fang. Extracting user-reported mobile application defects from online reviews. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 422–429, Nov 2017.

[70] Yuanchun Li, Baoxiong Jia, Yao Guo, and Xiangqun Chen. Mining user reviews for mobile app comparisons. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3), September 2017.

[71] Hui Guo and Munindar P. Singh. Caspar: Extracting and synthesizing user stories of problems from app reviews. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 628–640, Oct 2020.

[72] Grant Williams, Miroslav Tushev, Fahimeh Ebrahimi, and Anas Mahmoud. Modeling user concerns in sharing economy: the case of food delivery apps. *Autom. Softw. Eng.*, 27(3):229–263, 2020.

[73] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu. Paid: Prioritizing app issues for developers by tracking user reviews over versions. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 35–45, Nov 2015.

[74] Jeungmin Oh, Daehoon Kim, Uichin Lee, Jae-Gil Lee, and Junehwa Song. Facilitating developer-user interactions with mobile app review digests. In *CHI 13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA 13, page 18091814, New York, NY, USA, 2013. ACM.

[75] Andrea Di Sorbo, Giovanni Grano, Corrado Aaron Visaggio, and Sebastiano Panichella. Investigating the criticality of user-reported issues through their relations with app rating. *Journal of Software: Evolution and Process*, 33(3):e2316, 2020. e2316 smr.2316.

[76] Rishi Chandy and Haijie Gu. Identifying spam in the ios app store. In *Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 56–59. ACM, 2012.

[77] Daniel Martens and Walid Maalej. Towards understanding and detecting fake reviews in app stores. *Empirical Software Engineering*, 24(6):3316–3355, 2019.

[78] Yu Zhou, Yanqi Su, Taolue Chen, Zhiqiu Huang, Harald C. Gall, and Sebastiano Panichella. User review-based change file localization for mobile applications. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[79] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Corrado A. Visaggio, and Gerardo Canfora. Surf: Summarizer of user reviews feedback. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C 17, page 5558. IEEE Press, 2017.

[80] Martijn van Vliet, Eduard C. Groen, Fabiano Dalpiaz, and Sjaak Brinkkemper. Identifying and classifying user requirements in online feedback via crowdsourcing. In Nazim H. Madhavji, Liliana Pasquale, Alessio Ferrari, and Stefania Gnesi, editors, *Requirements Engineering: Foundation for Software Quality - 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24-27, 2020, Proceedings [REFSQ 2020 was postponed]*, volume 12045 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2020.

[81] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 10231027, New York, NY, USA, 2016. ACM.

[82] Kamonphop Srisopha, Chukiat Phonsom, Mingzhe Li, Daniel Link, and Barry Boehm. On building an automatic identification of country-specific feature requests in mobile app reviews: Possibilities and challenges. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 494498, New York, NY, USA, 2020. Association for Computing Machinery.

[83] S. Mujahid, G. Sierra, R. Abdalkareem, E. Shihab, and W. Shang. Examining user complaints of wearable apps: A case study on android wear. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 96–99, May 2017.

[84] A. Ciurumelea, S. Panichella, and H. C. Gall. Poster: Automated user reviews analyser. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 317–318, May 2018.

[85] Shuyue Li, Jiaqi Guo, Ming Fan, Jian-Guang Lou, Qinghua Zheng, and Ting Liu. Automated bug reproduction from user reviews for android applications. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 51–60, Oct 2020.

[86] Chuanqi Tao, Hongjing Guo, and Zhiqiu Huang. Identifying security issues for mobile applications based on user review summarization. *Information and Software Technology*, 122:106290, 2020.

[87] Hammad Khalid. On identifying user complaints of ios apps. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1474–1476. IEEE Press, 2013.

[88] E. Bakiu and E. Guzman. Which feature is unusable? detecting usability and user experience issues from user reviews. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 182–187, Sep. 2017.

[89] Iván Tactuk Mercado, Nuthan Munaiah, and Andrew Meneely. The impact of cross-platform development approaches for mobile applications from the users perspective. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, page 4349, New York, NY, USA, 2016. ACM.

[90] Peihan Wen and Mo Chen. A new analysis method for user reviews of mobile fitness apps. In Masaaki Kurosu, editor, *Human-Computer Interaction. Human Values and*

Quality of Life - Thematic Area, HCI 2020, Held as Part of the 22nd International Conference, HCII 2020, Copenhagen, Denmark, July 19-24, 2020, Proceedings, Part III, volume 12183 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2020.

[91] Lei Cen, Luo Si, Ninghui Li, and Hongxia Jin. User comment analysis for android apps and cspi detection with comment expansion. In *Proceeding of the 1st International Workshop on Privacy-Preserving IR (PIR)*, pages 25–30, 2014.

[92] Chong Wang, Fan Zhang, Peng Liang, Maya Daneva, and Marten van Sinderen. Can app changelogs improve requirements classification from app reviews? an exploratory study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM 18, New York, NY, USA, 2018. ACM.

[93] Hui Yang and Peng Liang. Identification and classification of requirements from app user reviews. In *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, pages 7–12, 2015.

[94] R. Deocadez, R. Harrison, and D. Rodriguez. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 367–371, Sep. 2017.

[95] Mengmeng Lu and Peng Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE17, page 344353, New York, NY, USA, 2017. ACM.

[96] T. Wang, P. Liang, and M. Lu. What aspects do non-functional requirements in app user reviews describe? an exploratory and comparative study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 494–503, Dec 2018.

[97] Z. Kurtanovi and W. Maalej. Mining user rationale from software reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 61–70, Sep. 2017.

[98] Zijad Kurtanovic and Walid Maalej. On user rationale in software engineering. *Requir. Eng.*, 23(3):357–379, 2018.

[99] Anang Kunaefi and Masayoshi Aritsugi. Characterizing user decision based on argumentative reviews. In *7th IEEE/ACM International Conference on Big Data Computing,*

Applications and Technologies, BDCAT 2020, Leicester, United Kingdom, December 7-10, 2020, pages 161–170. IEEE, 2020.

[100] E. Guzman, P. Bhuvanagiri, and B. Bruegge. Fave: Visualizing user feedback for software evolution. In *2014 Second IEEE Working Conference on Software Visualization*, pages 167–171, Sep. 2014.

[101] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Xiaoyu Wang. Analyzing reviews guided by app descriptions for the software development and evolution. *Journal of Software: Evolution and Process*, 30(12):e2112, 2018. e2112 JSME-17-0184.R2.

[102] E. Guzman, O. Aly, and B. Bruegge. Retrieving diverse opinions from app reviews. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, Oct 2015.

[103] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald Gall, Filomena Ferrucci, and Andrea De Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE 17, page 106117. IEEE Press, 2017.

[104] Zhenlian Peng, Jian Wang, Keqing He, and Mingdong Tang. An approach of extracting feature requests from app reviews. In *Collaborate Computing: Networking, Applications and Worksharing - 12th International Conference, CollaborateCom 2016, Beijing, China, November 10-11, 2016, Proceedings*, pages 312–323, 2016.

[105] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Phrase-based extraction of user opinions in mobile app reviews. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 726–731, 2016.

[106] Runyu Chen, Qili Wang, and Wei Xu. Mining user requirements to facilitate mobile app quality upgrades with big data. *Electronic Commerce Research and Applications*, 38:100889, 2019.

[107] Jianmao Xiao, Shizhan Chen, Qiang He, Hongyue Wu, Zhiyong Feng, and Xiao Xue. Detecting user significant intention via sentiment-preference correlation analysis for continuous app improvement. In Eleanna Kafeza, Boualem Benatallah, Fabio Martinelli, Hakim Hacid, Athman Bouguettaya, and Hamid Motahari, editors, *Service-Oriented Computing - 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates,*

December 14-17, 2020, Proceedings, volume 12571 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2020.

[108] Mohammad Abdul Hadi and Fatemeh H Fard. Aobtm: Adaptive online biterm topic modeling for version sensitive short-texts analysis. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 593–604, Sep. 2020.

[109] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Tool support for analyzing mobile app reviews. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 789–794, 2015.

[110] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. Finding and analyzing app reviews related to specific features: A research preview. In *Requirements Engineering: Foundation for Software Quality - 25th International Working Conference, REFSQ 2019, Essen, Germany, March 18-21, 2019, Proceedings*, pages 183–189, 2019.

[111] Tong Li, Fan Zhang, and Dan Wang. Automatic user preferences elicitation: A data-driven approach. In *Requirements Engineering: Foundation for Software Quality - 24th International Working Conference, REFSQ 2018, Utrecht, The Netherlands, March 19-22, 2018, Proceedings*, pages 324–331, 2018.

[112] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software*, 137:143 – 162, 2018.

[113] L. Pelloni, G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall. Becloma: Augmenting stack traces with user review information. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 522–526, March 2018.

[114] Ehsan Noei, Feng Zhang, Shaohua Wang, and Ying Zou. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering*, 24(4):1964–1996, 2019.

[115] Lili Wei, Yepang Liu, and Shing-Chi Cheung. Oasis: Prioritizing static analysis warnings for android apps based on app user reviews. In *Proceedings of the 2017 11th Joint*

Meeting on Foundations of Software Engineering, ESEC/FSE 2017, page 672682, New York, NY, USA, 2017. ACM.

[116] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall. Analyzing reviews and code of mobile apps for better release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 91–102, Feb 2017.

[117] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall. Exploring the integration of user feedback in automated testing of android applications. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 72–83, March 2018.

[118] Aman Yadav and Fatemeh H. Fard. Semantic analysis of issues on google play and twitter. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 308–309, Oct 2020.

[119] Aman Yadav, Rishab Sharma, and Fatemeh Hendijani Fard. A semantic-based framework for analyzing app users' feedback. In Kostas Kontogiannis, Foutse Khomh, Alexander Chatzigeorgiou, Marios-Eleftherios Fokaefs, and Minghui Zhou, editors, *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pages 572–576. IEEE, 2020.

[120] Emanuel Oehri and Emitza Guzman. Same same but different: Finding similar user feedback across multiple platforms and languages. In Travis D. Breaux, Andrea Zisman, Samuel Fricker, and Martin Glinz, editors, *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, pages 44–54. IEEE, 2020.

[121] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Shanquan Gao. Combining goal model with reviews for supporting the evolution of apps. *IET Softw.*, 14(1):39–49, 2020.

[122] Shanquan Gao, Lei Liu, Yuzhou Liu, Huaxiao Liu, and Yihui Wang. Updating the goal model with user reviews for the evolution of an app. *Journal of Software: Evolution and Process*, 32(8):e2257, 2020. e2257 JSME-19-0105.R2.

[123] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Xinglong Yin. App store mining for iterative domain analysis: Combine app descriptions with user reviews. *Software: Practice and Experience*, 49(6):1013–1040, 2019. SPE-19-0009.R1.

[124] Daniel Martens and Timo Johann. On the emotion of users in app reviews. In *Proceedings of the 2nd International Workshop on Emotion Awareness in Software Engineering*, SEmotion 17, page 814. IEEE Press, 2017.

[125] D. Martens and W. Maalej. Release early, release often, and watch your users' emotions: Lessons from emotional patterns. *IEEE Software*, 36(5):32–37, Sep. 2019.

[126] Kamonphop Srisopha, Devendra Swami, Daniel Link, and Barry Boehm. How features in ios app store reviews can predict developer responses. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, page 336341, New York, NY, USA, 2020. Association for Computing Machinery.

[127] Haroon Malik, Elhadi M. Shakshuki, and Wook-Sung Yoo. Comparing mobile apps by identifying hot features. *Future Generation Computer Systems*, 2018.

[128] Inthuja Gunaratnam and D.N. Wickramarachchi. Computational model for rating mobile applications based on feature extraction. In *2020 2nd International Conference on Advancements in Computing (ICAC)*, volume 1, pages 180–185, Dec 2020.

[129] R. A. Masrury, Fannisa, and A. Alamsyah. Analyzing tourism mobile applications perceived quality using sentiment analysis and topic modeling. In *2019 7th International Conference on Information and Communication Technology (ICoICT)*, pages 1–6, July 2019.

[130] Johannes Huebner, Remo Manuel Frey, Christian Ammendola, Elgar Fleisch, and Alexander Ilic. What people like in mobile finance apps: An analysis of user reviews. In *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia, MUM 2018, Cairo, Egypt, November 25-28, 2018*, pages 293–304, 2018.

[131] Fabiano Dalpiaz and Micaela Parente. RE-SWOT: from user feedback to requirements via competitor analysis. In *Requirements Engineering: Foundation for Software Quality - 25th International Working Conference, REFSQ 2019, Essen, Germany, March 18-21, 2019, Proceedings*, pages 55–70, 2019.

[132] Mariaclaudia Nicolai, Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. Healthcare android apps: a tale of the customers' perspective. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 27, 2019*, pages 33–39, 2019.

[133] Ting-Peng Liang, Xin Li, Chin-Tsung Yang, and Mengyue Wang. What in consumer reviews affects the sales of mobile apps: A multifacet sentiment analysis approach. *International Journal of Electronic Commerce*, 20(2):236–260, 2015.

[134] Raymond P. L. Buse and Thomas Zimmermann. Information needs for software development analytics. In *34th International Conference on Software Engineering*, pages 987–996, 2012.

[135] Andrew Begel and Thomas Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *36th International Conference on Software Engineering*, pages 12–13, 2014.

[136] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and Kon Mouzakis. A preliminary analysis of vocabulary in mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 245–248. ACM, 2012.

[137] G. Williams and A. Mahmoud. Modeling user concerns in the app store: A case study on the rise and fall of yik yak. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 64–75, Aug 2018.

[138] Kanimozhi Kalaichelavan, Haroon Malik, Narman Husnu, and Sreehari Sreenath. What do people complain about drone apps? a large-scale empirical study of google play store reviews. *Procedia Computer Science*, 170:547–554, 2020. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[139] Huiyi Wang, Liu Wang, and Haoyu Wang. Market-level analysis of government-backed covid-19 contact tracing apps. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ASE ’20, page 7984, New York, NY, USA, 2020. Association for Computing Machinery.

[140] Claudia Iacob, Varsha Veerappa, and Rachel Harrison. What are you complaining about?: A study of online reviews of mobile applications. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*, pages 29:1–29:6. British Computer Society, 2013.

[141] S. Hassan, C. Bezemer, and A. E. Hassan. Studying bad updates of top free-to-download apps in the google play store. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[142] K. Srisopha, C. Phonsom, K. Lin, and B. Boehm. Same app, different countries: A preliminary user reviews study on most downloaded ios apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 76–80, Sep. 2019.

[143] E. Guzman and A. Paredes Rojas. Gender and user feedback: An exploratory study. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 381–385, Sep. 2019.

[144] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. End users' perception of hybrid mobile apps in the google play store. In *Proceedings of the 4th International Conference on Mobile Services (MS)*. IEEE, 2015.

[145] Mohamed Ali, Mona Erfani Joorabchi, and Ali Mesbah. Same app, different app stores: A comparative study. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft 17, page 7990. IEEE Press, 2017.

[146] Hanyang Hu, Cor-Paul Bezemer, and Ahmed E. Hassan. Studying the consistency of star ratings and the complaints in 1 & 2-star user reviews for top free cross-platform android and ios apps. *Empirical Software Engineering*, 23(6):3442–3475, 2018.

[147] Hanyang Hu, Shaowei Wang, Cor-Paul Bezemer, and Ahmed E. Hassan. Studying the consistency of star ratings and reviews of popular free hybrid android and ios apps. *Empirical Software Engineering*, 24(1):7–32, 2019.

[148] Stuart McIlroy, Weiyi Shang, Nasir Ali, and Ahmed Hassan. Is it worth responding to reviews? a case study of the top free apps in the google play store. *IEEE Software*, PP, 2015.

[149] Safwat Hassan, Chakkrit Tantithamthavorn, Cor-Paul Bezemer, and Ahmed E. Hassan. Studying the dialogue between users and developers of free apps in the google play store. *Empirical Software Engineering*, 23(3):1275–1312, 2018.

[150] Y. Man, C. Gao, M. R. Lyu, and J. Jiang. Experience report: Understanding cross-platform app issues from user reviews. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 138–149, Oct 2016.

[151] S. Scalabrino, G. Bavota, B. Russo, M. D. Penta, and R. Oliveto. Listening to the crowd for the release planning of mobile apps. *IEEE Transactions on Software Engineering*, 45(1):68–86, Jan 2019.

[152] Swetha Keertipati, Bastin Tony Roy Savarimuthu, and Sherlock A. Licorish. Approaches for prioritizing feature improvements extracted from app reviews. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE 16, New York, NY, USA, 2016. ACM.

[153] G. Tong, B. Guo, O. Yi, and Y. Zhiwen. Mining and analyzing user feedback from app reviews: An econometric approach. In *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 841–848, Oct 2018.

[154] Sherlock A. Licorish, Bastin Tony Roy Savarimuthu, and Swetha Keertipati. 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*, EASE17, page 108117, New York, NY, USA, 2017. ACM.

[155] Jianzhang Zhang, Yinglin Wang, and Tian Xie. Software feature refinement prioritization based on online user review mining. *Information and Software Technology*, 108:30 – 34, 2019.

[156] Hammad Khalid, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. Prioritizing the devices to test your app on: a case study of android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 610–620, 2014.

[157] G. Greenheld, B. T. R. Savarimuthu, and S. A. Licorish. Automating developers' responses to app reviews. In *2018 25th Australasian Software Engineering Conference (ASWEC)*, pages 66–70, Nov 2018.

[158] C. Gao, J. Zeng, X. Xia, D. Lo, M. R. Lyu, and I. King. Automating app review response generation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 163–175, Nov 2019.

[159] Phong Minh Vu, Tam The Nguyen, and Tung Thanh Nguyen. Why do app reviews get responded: A preliminary study of the relationship between reviews and responses in mobile apps. In *Proceedings of the 2019 ACM Southeast Conference*, ACM SE 19, page 237240, New York, NY, USA, 2019. ACM.

[160] Claudia Iacob, Rachel Harrison, and Shamal Faily. Online reviews as first class artifacts in mobile app development. In *Proceedings of the 5th International Conference on Mobile Computing, Applications, and Services. MobiCASE '13*, 2013.

[161] K. Phetrungnapha and T. Senivongse. Classification of mobile application user reviews for generating tickets on issue tracking system. In *2019 12th International Conference on Information Communication Technology and System (ICTS)*, pages 229–234, July 2019.

[162] Cuiyun Gao, Hui Xu, Junjie Hu, and Yangfan Zhou. Ar-tracker: Track the dynamics of mobile apps via user review mining. In *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE '15*, pages 284–290, 2015.

[163] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Using app reviews for competitive analysis: Tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA 2019, page 4046, New York, NY, USA, 2019. ACM.

[164] Mario Sänger, Ulf Leser, Steffen Kemmerer, Peter Adolphs, and Roman Klinger. SCARE - the sentiment corpus of app reviews with fine-grained annotations in German. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016.

[165] C. Stanik, M. Haering, and W. Maalej. Classifying multilingual user feedback using traditional machine learning and deep learning. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 220–226, Sep. 2019.

[166] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009.

[167] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[168] Nishant Jha and Anas Mahmoud. MARC: A mobile application review classifier. In *Joint Proceedings of REFSQ-2017 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017), Essen, Germany, February 27, 2017*, 2017.

[169] Jianmao Xiao. Ospaci: Online sentiment-preference analysis of user reviews for continues app improvement. In Sami Yangui, Athman Bouguettaya, Xiao Xue, Noura Faci,

Walid Gaaloul, Qi Yu, Zhangbing Zhou, Nathalie Hernandez, and Elisa Yumi Nakagawa, editors, *Service-Oriented Computing - ICSOC 2019 Workshops - WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28-31, 2019, Revised Selected Papers*, volume 12019 of *Lecture Notes in Computer Science*, pages 273–279. Springer, 2019.

[170] A. Puspaningrum, D. Siahaan, and C. Fatichah. Mobile app review labeling using lda similarity and term frequency-inverse cluster frequency (tf-icf). In *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 365–370, July 2018.

[171] Rui Song, Tong Li, and Zhiming Ding. Automatically identifying requirements-oriented reviews using a top-down feature extraction approach. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 450–454, Dec 2020.

[172] Saurabh Malgaonkar, Sherlock A. Licorish, and Bastin Tony Roy Savarimuthu. Towards automated taxonomy generation for grouping app reviews: A preliminary empirical study. In Martin J. Shepperd, Fernando Brito e Abreu, Alberto Rodrigues da Silva, and Ricardo Pérez-Castillo, editors, *Quality of Information and Communications Technology - 13th International Conference, QUATIC 2020, Faro, Portugal, September 9-11, 2020, Proceedings*, volume 1266 of *Communications in Computer and Information Science*, pages 120–134. Springer, 2020.

[173] M. D. Kafil Uddin, Qiang He, Jun Han, and Caslon Chua. App competition matters: How to identify your competitor apps? In *2020 IEEE International Conference on Services Computing, SCC 2020, Beijing, China, November 7-11, 2020*, pages 370–377. IEEE, 2020.

[174] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[175] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah. App review analysis via active learning: Reducing supervision effort without compromising classification accuracy. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 170–181, Aug 2018.

[176] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simplifying the classification of app reviews using only lexical features. In *Software Technologies - 13th International Con-*

ference, ICSOFT 2018, Porto, Portugal, July 26-28, 2018, Revised Selected Papers, pages 173–193, 2018.

[177] Kamonphop Srisopha, Daniel Link, Devendra Swami, and Barry Boehm. Learning features that predict developer responses for ios app store reviews. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA, 2020. Association for Computing Machinery.

[178] H. Khalid, M. Nagappan, and A. E. Hassan. Examining the relationship between findbugs warnings and app ratings. *IEEE Software*, 33(4):34–39, July 2016.

[179] E. Guzman, L. Oliveira, Y. Steiner, L. C. Wagner, and M. Glinz. User feedback in the app store: A cross-cultural study. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 13–22, May 2018.

[180] Kamonphop Srisopha and Reem Alfayez. Software quality through the eyes of the end-user and static analysis tools: A study on android oss applications. In *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies*, SQUADE 18, page 14, New York, NY, USA, 2018. ACM.

[181] M. Goul, O. Marjanovic, S. Baxley, and K. Vizecky. Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering. In *2012 45th Hawaii International Conference on System Sciences*, pages 4168–4177, Jan 2012.

[182] Dong Sun and Rong Peng. A scenario model aggregation approach for mobile app requirements evolution based on user comments. In *Requirements Engineering in the Big Data Era*, volume 558, pages 75–91. Springer Berlin Heidelberg, 2015.

[183] Faiz Ali Shah, Yevhenii Sabanin, and Dietmar Pfahl. Feature-based evaluation of competing apps. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, page 1521, New York, NY, USA, 2016. ACM.

[184] Z. S. H. Abad, S. D. V. Sims, A. Cheema, M. B. Nasir, and P. Harisinghani. Learn more, pay less! lessons learned from applying the wizard-of-oz technique for exploring mobile app requirements. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 132–138, Sep. 2017.

[185] Li Zhang, Xin-Yue Huang, Jing Jiang, and Ya-Kun Hu. Cslabel: An approach for labelling mobile app reviews. *J. Comput. Sci. Technol.*, 32(6):1076–1089, 2017.

[186] Roger Deocadez, Rachel Harrison, and Daniel Rodriguez. Preliminary study on applying semi-supervised learning to app store analysis. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE17, page 320323, New York, NY, USA, 2017. ACM.

[187] Nishant Jha and Anas Mahmoud. Mining non-functional requirements from app store reviews. *Empirical Software Engineering*, 24(6):3659–3695, 2019.

[188] N. Al Kilani, R. Tailakh, and A. Hanani. 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)*, pages 541–548, Oct 2019.

[189] James Tizard, Tim Rietz, and Kelly Blincoe. Voice of the users: A demographic study of software feedback behaviour. In Travis D. Breaux, Andrea Zisman, Samuel Fricker, and Martin Glinz, editors, *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, pages 55–65. IEEE, 2020.

[190] Tanusree Sharma and Masooda N. Bashir. Privacy apps for smartphones: An assessment of users' preferences and limitations. In Abbas Moallem, editor, *HCI for Cybersecurity, Privacy and Trust - Second International Conference, HCI-CPT 2020, Held as Part of the 22nd HCI International Conference, HCII 2020, Copenhagen, Denmark, July 19-24, 2020, Proceedings*, volume 12210 of *Lecture Notes in Computer Science*, pages 533–546. Springer, 2020.

[191] Vinicius H. S. Durelli, Rafael S. Durelli, Andre T. Endo, Elder Cirilo, Washington Luiz, and Leonardo Rocha. Please please me: Does the presence of test cases influence mobile app users satisfaction? In *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, SBES 18, page 132141, New York, NY, USA, 2018. ACM.

[192] Maria Gomez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier. A recommender system of buggy app checkers for app store moderators. In *2nd ACM International Conference on Mobile Software Engineering and Systems*. IEEE, 2015.

[193] Mubasher Khalid, Usman Shehzaib, and Muhammad Asif. A case of mobile app reviews as a crowdsource. *International Journal of Information Engineering and Electronic Business (IJIEEB)*, 7(5):39, 2015.

[194] Haroon Malik and Elhadi M. Shakshuki. Mining collective opinions for comparison of mobile apps. *Procedia Computer Science*, 94:168 – 175, 2016. The 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops.

[195] S. Muñoz, O. Araque, A. F. Llamas, and C. A. Iglesias. A cognitive agent for mining bugs reports, feature suggestions and sentiment in a mobile application store. In *2018 4th International Conference on Big Data Innovations and Applications (Innovate-Data)*, pages 17–24, Aug 2018.

[196] E. Noei, F. Zhang, and Y. Zou. Too many user-reviews, what should app developers look at first? *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

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

[198] Leonard Hoon, Rajesh Vasa, Gloria Yoanita Martino, Jean-Guy Schneider, and Kon Mouzakis. Awesome! conveying satisfaction on the app store. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI 13, page 229232, New York, NY, USA, 2013. ACM.

[199] Mubasher Khalid, Muhammad Asif, and Usman Shehzaib. Towards improving the quality of mobile app reviews. *International Journal of Information Technology and Computer Science (IJITCS)*, 7(10):35, 2015.

[200] Ivano Malavolta, Stefano Ruberto, Valerio Terragni, and Tommaso Soru. Hybrid mobile apps in the google play store: an exploratory investigation. In *Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems*. ACM, 2015.

[201] Leonard Hoon, MiguelAngel Rodriguez-García, Rajesh Vasa, Rafael Valencia-García, and Jean-Guy Schneider. App reviews: Breaking the user and developer language bar-

rier. In *Trends and Applications in Software Engineering*, volume 405, pages 223–233. Springer International Publishing, 2016.

[202] M. Nagappan and E. Shihab. Mobile app store analytics. In Tim Menzies, Laurie Williams, and Thomas Zimmermann, editors, *Perspectives on Data Science for Software Engineering*, pages 47 – 49. Morgan Kaufmann, Boston, 2016.

[203] Andrew Simmons and Leonard Hoon. Agree to disagree: On labelling helpful app reviews. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction*, OzCHI 16, page 416420, New York, NY, USA, 2016. ACM.

[204] M. Nayebi, H. Cho, H. Farrahi, and G. Ruhe. App store mining is not enough. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 152–154, May 2017.

[205] Giovanni Grano, Andrea Di Sorbo, Francesco Mercaldo, Corrado A. Visaggio, Gerardo Canfora, and Sebastiano Panichella. Android apps and user feedback: A dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA 2017, page 811, New York, NY, USA, 2017. ACM.

[206] Zaoyi Sun, Zhiwei Ji, Pei Zhang, Chuansheng Chen, Xiuying Qian, Xin Du, and Qun Wan. Automatic labeling of mobile apps by the type of psychological needs they satisfy. *Telematics and Informatics*, 34(5):767 – 778, 2017.

[207] Nishant Jha and Anas Mahmoud. Mining user requirements from application store reviews using frame semantics. In *Requirements Engineering: Foundation for Software Quality - 23rd International Working Conference, REFSQ 2017, Essen, Germany, February 27 - March 2, 2017, Proceedings*, pages 273–287, 2017.

[208] Shance Wang, Zhongjie Wang, Xiaofei Xu, and Quan Z. Sheng. App update patterns: How developers act on user reviews in mobile app stores. In *Service-Oriented Computing - 15th International Conference, ICSOC 2017, Malaga, Spain, November 13-16, 2017, Proceedings*, pages 125–141, 2017.

[209] Stuart Mcilroy, Weiyi Shang, Nasir Ali, and Ahmed E. Hassan. User reviews of top mobile apps in apple and google app stores. *Commun. ACM*, 60(11):6267, October 2017.

[210] Maleknaz Nayebi, Henry Cho, and Guenther Ruhe. App store mining is not enough for app improvement. *Empirical Software Engineering*, 23(5):2764–2794, 2018.

[211] G. Deshpande and J. Rokne. User feedback from tweets vs app store reviews: An exploratory study of frequency, timing and content. In *2018 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 15–21, Aug 2018.

[212] Gian Luca Scoccia, Stefano Ruberto, Ivano Malavolta, Marco Autili, and Paola Inverardi. An investigation into android run-time permissions from the end users perspective. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, MOBILESoft 18, page 4555, New York, NY, USA, 2018. ACM.

[213] Ehsan Noei, Daniel Alencar Da Costa, and Ying Zou. Winning the app production rally. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 283294, New York, NY, USA, 2018.

[214] Suhaib Mujahid, Giancarlo Sierra, Rabe Abdalkareem, Emad Shihab, and Weiyi Shang. An empirical study of android wear user complaints. *Empirical Software Engineering*, 23(6):3476–3502, 2018.

[215] Nishant Jha and Anas Mahmoud. Using frame semantics for classifying and summarizing application store reviews. *Empirical Software Engineering*, 23(6):3734–3767, 2018.

[216] P. Weichbroth and A. Baj-Rogowska. Do online reviews reveal mobile application usability and user experience? the case of whatsapp. In *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 747–754, Sep. 2019.

[217] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews. In *International Conference on Software Technologies (ICSOFT)*, 2019.

[218] Walid Maalej, Maleknaz Nayebi, and Guenther Ruhe. Data-driven requirements engineering: An update. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP 19, page 289290. IEEE Press, 2019.

[219] Kendall Bailey, Meiyappan Nagappan, and Danny Dig. Examining user-developer feedback loops in the ios app store. In *52nd Hawaii International Conference on System*

Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019, pages 1–10, 2019.

[220] ISO/IEC 25010. ISO/IEC 25010:2011, systems and software engineering systems and software quality requirements and evaluation (SQuaRE) system and software quality models, 2011.

[221] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[222] B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

[223] Janet E. Burge, John M. Carroll, Raymond McCall, and Ivan Mistrk. *Rationale-Based Software Engineering*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[224] IEEE. IEEE Standard Glossary of Software Engineering Terminology, 1990.

[225] Mona Erfani, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 15–24, 10 2013.

[226] James Pustejovsky and Amber Stubbs. *Natural Language Annotation for Machine Learning - a Guide to Corpus-Building for Applications*. O'Reilly, 2012.

[227] Kevin Hallgren. Computing inter-rater reliability for observational data: An overview and tutorial. *Tutorials in Quantitative Methods for Psychology*, 8:23–34, 07 2012.

[228] Chandler J Cumpston M Li T Page MJ Welch VA Higgins JPT, Thomas J, editor. *Cochrane Handbook for Systematic Reviews of Interventions*. Chichester (UK): John Wiley & Sons, 2nd edition edition, 2019.

[229] Blake Miller, Fridolin Linder, and Walter R. Mebane. Active learning approaches for labeling text: Review and assessment of the performance of active learning approaches. *Political Analysis*, page 120, 2020.

[230] Dave H. Annis. Probability and statistics: The science of uncertainty, michael j. evans and jeffrey s. rosenthal. *The American Statistician*, 59:276–276, 2005.

[231] Daniel M. Berry. Evaluation of tools for hairy requirements and software engineering tasks. In *IEEE 25th International Requirements Engineering Conference Workshops, RE 2017 Workshops, Lisbon, Portugal, September 4-8, 2017*, pages 284–291, 2017.

[232] Daniel Berry. Keynote: Evaluation of NLP tools for hairy RE tasks. In *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018), Utrecht, The Netherlands, March 19, 2018*, 2018.

[233] Domenico Talia. A view of programming scalable data analysis: from clouds to exascale. *Journal of Cloud Computing*, 8(1):4, 2019.

[234] Analytics India Mag. `https://analyticsindiamag.com/challenges-of-implementing-natural-language-processing/`, July 2020. Accessed: 2021-06-01.

[235] Explorium. Understanding and Handling Data and Concept Drift. `https://www.explorium.ai/blog/understanding-and-handling-data-and-concept-drift/`, 2020. Accessed: 2021-06-01.

[236] Barbara A. Kitchenham, Tore Dyba, and Magne Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE 04, page 273281, USA, 2004. IEEE Computer Society.

[237] Paul Ralph, Sebastian Baltes, Domenico Bianculli, Yvonne Dittrich, Michael Felderer, Robert Feldt, Antonio Filieri, Carlo Alberto Furia, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara A. Kitchenham, Romain Robbes, Daniel Méndez, Jefferson Molleri, Diomidis Spinellis, Miroslaw Staron, Klaas-Jan Stol, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, and Sira Vegas. ACM SIGSOFT empirical standards. *CoRR*, abs/2010.03525, 2020.

[238] Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. Use cases and reference architecture for mining app reviews. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*, 2022.

[239] S. Angelov, P. Grefen, and D. Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, pages 141–150, 2009.

[240] Robert J. Cloutier, Gerrit Muller, Dinesh Verma, Roshanak R. Nilchiani, Eirik Hole, and Mary A. Bone. The concept of reference architectures. *System Engineering*, 13:14–27, 2010.

[241] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[242] Antony Tang and Aldeida Aleti. Human reasoning and software design: an analysis. 2010.

[243] Antony Tang, Aldeida Aleti, Janet Burge, and Hans van Vliet. What makes software design effective? *Design Studies*, 31(6):614–640, 2010. Special Issue Studying Professional Software Design.

[244] Gerben Wierda. *Mastering ArchiMate Edition III: A Serious Introduction to the ArchiMate Enterprise Architecture Modeling Language*. R&amp;A, 2017.

[245] Jacek Dąbrowski. Supplementary materials for use cases and reference architectures. `https://github.com/jsdabrowski/RA-Models`, March 2022.

[246] Matthias Galster and Paris Avgeriou. Empirically-grounded reference architectures: A proposal. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, QoSA-ISARCS '11, page 153158, New York, NY, USA, 2011. Association for Computing Machinery.

[247] Foster Provost and Tom Fawcett. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. O'Reilly Media, Inc., 1st edition, 2013.

[248] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI series in software engineering. Addison-Wesley, 2003.

[249] Wen Feng, Edward F. Crawley, and Olivier L. de Weck. *Design Structure Matrix Methods and Applications*, chapter BP Stakeholder Value Network, Example 5.5. MIT Press, 2012.

[250] John Klein, Ross Buglak, David Blockow, Troy Wuttke, and Brenton Cooper. A reference architecture for big data systems in the national security domain. In *2016 IEEE/ACM*

2nd International Workshop on Big Data Software Engineering (BIGDSE), pages 51–57, 2016.

[251] Pekka Pääkkönen and Daniel Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big Data Res.*, 2(4):166–186, 2015.

[252] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman. *Software and Systems Traceability*. Springer Publishing Company, Incorporated, 2012.

[253] Natalia Juristo and Omar S. Gómez. *Replication of Software Engineering Experiments*, pages 60–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[254] TrustRadius. `https://www.trustradius.com/mobile-analytics#products`. Accessed: 2021-03-18.

[255] G2. `https://www.g2.com/categories/mobile-app-analytics`. Accessed: 2021-03-18.

[256] Elisa Y. Nakagawa, Milena Guessi, José C. Maldonado, Daniel Feitosa, and Flavio Oquendo. Consolidating a process for the design, representation, and evaluation of reference architectures. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 143–152, 2014.

[257] Vladimir Elvov. Design of Big Data Reference Architectures for Use Cases in the Insurance Sector. Master's thesis, The Technical University of Munich, 2018.

[258] Mary Shaw. Writing good software engineering research papers: Minitutorial. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, page 726736, USA, 2003. IEEE Computer Society.

[259] Appbot. `https://appbot.co`. Accessed: 2021-03-01.

[260] Applysis. `https://applysis.io`. Accessed: 2021-03-01.

[261] RankMyApp. `https://apprankcorner.com`. Accessed: 2021-03-01.

[262] WonderFlow. `https://www.wonderflow.ai`. Accessed: 2021-03-01.

[263] AppRadar. `https://appradar.com`. Accessed: 2021-03-01.

[264] AppFollow. `https://appfollow.io`. Accessed: 2021-03-01.

[265] AppFigures. `https://appfigures.com`. Accessed: 2021-03-01.

[266] Apptopia. `https://apptopia.com`. Accessed: 2021-03-01.

[267] Sachiko Lim, Aron Henriksson, and Jelena Zdravkovic. Data-driven requirements elicitation: A systematic literature review. *SN Computer Science*, 2(1):16, 2021.

[268] Jan Schroeder, Daniela Holzner, Christian Berger, Carl-Johan Hoel, Leo Laine, and Anders Magnusson. Design and evaluation of a customizable multi-domain reference architecture on top of product lines of self-driving heavy vehicles - an industrial case study. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 189–198, 2015.

[269] James S. Albus. 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles. In Grant R. Gerhart, Chuck M. Shoemaker, and Douglas W. Gage, editors, *Unmanned Ground Vehicle Technology IV*, volume 4715, pages 303 – 310. International Society for Optics and Photonics, SPIE, 2002.

[270] Heiko Koziolek, Andreas Burger, Marie Platenius-Mohr, Julius Rückert, and Gösta Stomberg. Openpnp: A plug-and-produce architecture for the industrial internet of things. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 131–140, 2019.

[271] Paris Avgeriou. Describing, instantiating and evaluating a reference architecture : A case study. 2003.

[272] Markus Maier. Towards a Big Data Reference Architecture. Master's thesis, Eindhoven University of Technology, 2013.

[273] Go Muan Sang, Lai Xu, and Paul Ton de Vrieze. A reference architecture for big data systems. *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, pages 370–375, 2016.

[274] Bing Liu. *Sentiment Analysis and Opinion Mining*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.

[275] Mauro Dragoni, Marco Federici, and Andi Rexha. An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Inf. Process. Manage.*, 56(3):1103–1118, 2019.

[276] Jacek Dąbrowski. Manually annotated dataset and an annotation guideline for CAiSE'20 paper. `https://github.com/jsdabrowski/CAiSE-20/`, November 2019.

[277] Don S. Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[278] Karl E Wiegers and Joy Beatty. *Software Requirements 3*. Microsoft Press, USA, 2013.

[279] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 01 1990.

[280] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[281] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010.

[282] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *38th International Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[283] Hamish Cunningham, Diana Maynard, Valentin Tablan, Cristian Ursu, and Kalina Bontcheva. Developing Language Processing Components with GATE Version 8. *University of Sheffield Department of Computer Science*, November 2014.

[284] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[285] D. M. Berry, J. Cleland-Huang, A. Ferrari, W. Maalej, J. Mylopoulos, and D. Zowghi. Panel: Context-Dependent Evaluation of Tools for NL RE Tasks: Recall vs. Precision, and Beyond. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 570–573, Sep. 2017.

[286] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[287] Afnan A. Al-Subaihin. *Software Engineering in the Age of App Stores: Feature-Based Analyses to Guide Mobile Software Engineers. Doctoral thesis*. PhD thesis, University College London, 2019.

[288] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *40th International Conference on Software Engineering*, pages 94–104, 2018.

[289] Afnan Al-Subaihin. *Software Engineering in the Age of App Stores: Feature-Based Analyses to Guide Mobile Software Engineers*. PhD thesis, University College London, 2019.

[290] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., USA, 2010.

[291] Jacek Dąbrowski. Manually annotated dataset and an annotation guideline for IS'22 paper. `https://github.com/jsdabrowski/IS-22/`, June 2022.

[292] Robert R. Korfhage. *Information storage and retrieval*. John Wiley & Sons, 1997.

[293] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin. A systematic literature review of software requirements prioritization research. *Inf. Softw. Technol.*, 56(6):568–585, June 2014.

[294] Jeff Carver, Natalia Juristo, Maria Baldassarre, and Sira Vegas. Replications of software engineering experiments. *Empirical Software Engineering*, 19, 04 2014.

[295] International Organization for Standardization (ISO). ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

[296] Petr Hajek, Aliaksandr Barushka, and Michal Munk. Opinion mining of consumer reviews using deep neural networks with word-sentiment associations. In Ilias Maglogiannis, Lazaros Iliadis, and Elias Pimenidis, editors, *Artificial Intelligence Applications and Innovations*, pages 419–429, Cham, 2020. Springer International Publishing.

[297] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 94104, New York, NY, USA, 2018. Association for Computing Machinery.

[298] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR ’18, page 364375, New York, NY, USA, 2018. Association for Computing Machinery.