Mining User Feedback For Software Engineering: Use Cases and Reference Architecture

Jacek Dąbrowski*†, Emmanuel Letier*, Anna Perini† and Angelo Susi†
*University College London, London, UK
{j.dabrowski, e.letier}@cs.ucl.ac.uk
†Fondazione Bruno Kessler, Trento, Italy
{dabrowski, perini, susi}@fbk.eu

***Abstract*—App reviews can provide valuable information about user needs but analyzing them manually is challenging due to their large quantity and noisy nature. To overcome this problem, a variety of app review mining techniques have been proposed. So far, however, research in this area has paid little attention to the software engineering use cases of the mining techniques. This limits the understanding of their usefulness, applications and desired future developments. We address this problem by elaborating a reference model relating app review mining techniques to specific software engineering activities. In this paper, we present a unified description of software engineering use cases for mining app reviews and define a reference architecture realizing these use cases through a combination of natural language processing and data mining techniques. The use cases provide a novel systematic exposition of the envisioned applications and benefits of app review mining for software engineers. The reference architecture synthesises the diversity of research to realise these benefits and provide a general framework guiding the development and evaluation of future research and tools.**

***Index Terms*—Requirements Engineering, Mining User Feedback, Software Engineering, Reference Architecture, Software Analytics**

I. Introduction

App reviews provide a rich source of information that can help software engineers to maintain and evolve their products [13], [59], [73]. Exploiting this information, however, is challenging due to the large number of reviews [29], [36], [56], [73] and the difficulty in extracting actionable information from short informal texts [43], [56].

A variety of app review mining techniques have been proposed to address the problem [43], [44], [59], [73]; for example, to classify reviews by topics [29], [38], [56], [61], to extract information like bug descriptions [23], [45], or to analyze user opinions [30], [39]. Research in this area has grown rapidly [23], [59], resulting in a large number of scientific publications (at least 182 between 2012 and 2020) [23].

A few studies take the perspective of practitioners and analyse the state of practice in exploiting user feedback for software maintenance and evolution [23], [85]. For instance, studies interviewed practitioners to understand how user feedback is captured and used by software engineers for specific tasks, such as verification and validation [46]; and they found that many practitioners still use a manual approach for collecting feedback and that the exploitation of feedback in their software development process is still not systematic [85]. Other interviews with practitioners highlight the importance that software developers give to user reviews in app stores to guide the design and evolution of their apps [13], [84].

These studies show that a gap still exists between the perspectives of researchers and developers on how to leverage online user feedback for software engineering. Understanding which tools and techniques should be combined, and how, to include user feedback mining into software development in a systematic way is still an open problem.

In this perspective paper, we present a study that aims to address the problem by elaborating a unified description of software engineering use cases for mining app reviews and by defining a reference architecture that realizes 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 the realize specific software engineering use cases. The use cases describe the usage scenarios of this architecture for software engineering purposes. More specifically, we study the following research questions:
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?

We answer these questions by analysing an existing data set collected for a previous systematic literature review (SLR) [23]. To answer RQ3, we have extended the SLR data set with additional information about commercial app review mining tools. The novelty of this paper with respect to the SLR is the study of three new research questions (RQ1-RQ3) by applying systematic information modelling and knowledge synthesis steps detailed in Section III. This study generates three novel outputs: (i) a synthesis of the software engineering uses cases for app review mining, (ii) a reference architecture for realizing these use cases, and (iii) a mapping between app review mining tools and the reference architecture components. This mapping covers publicly available tools in research papers as well as those from commercial organisations. The use cases shed new light on the applications and benefits of app review mining techniques for software engineers, notably by relating use cases to software engineering goals. The reference

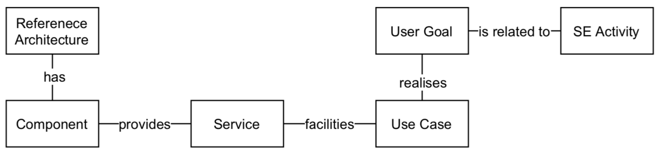

Fig. 1: The main concepts that we use to describe our reference architecture.

architecture synthesises the diversity of research to realise these goals 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 gaps in research and commercial tools.

II. Terminology

Figure 1 illustrates the main concepts used in the paper to describe the use cases and reference architecture for app review mining. 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 [14]. In this study, we view 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 [22]; 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 [53]. 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 [19].

III. 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 an existing dataset collected for a previous systematic literature review [23] 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 [48]. 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 activities, 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 [28]. Table I lists the data items used in this paper; we have selected these data items because they contain information to answer RQ1–RQ3. The 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 synthetized directly. In particular, the SLR classified the collected information for the software engineering activity (F1) as one of 14 standard software engineering activities as defined in the software engineering body of knowledge [19]; and the app review analysis (F3) as one of 9 broad types identified from the previous surveys on intelligent mining techniques [83] and text analytics [62], [78], [79]. The classification categories have been systematically derived from 182 publications using content analysis [18]. 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. The SLR data set does not contain explicit data items listing the software engineering uses cases and reference architecture components for app review mining tools.

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 [81], [82]; the method supports a creative process in which different pieces of information are combined to form a new artifact [82]. 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

TABLE I: Data items from a previous systematic literature review [23] used to answer RQ1-RQ3.

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) [19].
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.

the process and the service viewpoints using ArchiMate modelling language [89]; 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 describes the 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 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 [31].
iii) Knowledge Synthesis. We used the models derived from the information modeling step to define the use cases (RQ1) and the reference architecture (RQ2) [34]; 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 (the 'What'), a goal of a use case (the '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 interactions with an app analysis tool; each developer's step corresponds 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 [71]; we therefore examined the developer's steps in the process view of a model to determine what type of information a developer is interested 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 is interested in 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 fine-grained use cases; each fine-grained use case describes 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 for each group. We combined a group of fine-grained use cases into a coarse-grained use case 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 [17]. 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 [33]; 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 partial implementations of this architecture already exist (RQ3). As previous studies (e.g. [49], [65]), we validated our reference architecture using the matrix traceability method [21]; 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 [47]. To identify research tools and their features, we used information F3 ('app review analysis') and F4 ('replication package') from the SLR data set [23]. To identify commercial tools, we used popular software comparison platforms: TrustRadius [9] and G2 [7]. 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 [14].

IV. 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. The use cases are related to 14 software engineering activities from the Software Engineering Body of Knowledge [19]. The following sections describe 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') [53]. We give references to the papers justifying the relevance of the use cases for software engineering activities.

A. 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 features of an app; search for reviews referring to a concrete opinion of their interest; and to summarize these reviews to present their main points.

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 [13], [66]. Examining reviews providing users justifications for these opinions can help to identify problems with app features [29], [30], requested modification [45] or new requirements [29].
- *Requirements Specification:* Reviews justifying negative opinions can be used as information for requirements

specification or ad-hoc documentation [52], [57], [66]; 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 [39], [45], [57]. Developers may compare how often these opinions appear, for how long they have been made, and whether their frequency is increasing or decreasing [29], [30].
- *Validation by Users:* Knowing what features users like or dislike can indicate user acceptance of these features [30], [39]. Examining reviews of these opinions may help to know what users say about these features [13], [23].

B. 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 [56], whereas those requesting changes or reporting problems may indicate potential perfective and corrective modifications [66], [86].
- *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 [56], [57]. Reviews, describing steps to reproduce bugs, can be useful to complement crash reports of stack traces which could be difficult to understand standalone [37], [70].
- *Test Design:* Analyzing reviews reporting problems may describe scenarios in which an unusual situation emerged or there was lack of workarounds [57]. The information can be used to design test cases capturing such exception and exercising the scenarios [38], [57].
- *Requirements Prioritization, Requested Modification Prioritization, Test Prioritization:* User requests may help developers prioritize their work when added with statistics [64], [73]. 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 [29].
- *Validation by Users:* Identifying and quantifying reviews reporting problems can help during public beta testing before an official release [13], [56]. If the number of problems is high, or the problems concern core features, a new release can be postponed [13], [29]. Analyzing the reviews can help to know what users say about these problems [13], [30].
- *Help Desk:* Responding to reviews may answer app users questions or assist in troubleshooting [41], [60]. Such responses may also inform users about addressing their requests (e.g., new modifications or fixing problems) [60].
- *Impact Analysis:* Reviews with change requests can be linked to source code and indicate code snippets requiring modifications [20], [67], [68]. 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 [20], [67], [68].

C. Use Case 3: Analyze Non-Functional Requirements

Description (What): The app developer's goal is to analyze non-functional requirements (NFRs). 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 [44], [55]. Understanding what user says about these attributes may help to elicit new requirements, identify issues, or serve as inspiration for designing test scenarios or acceptance criteria [38], [38], [57].
- *Requirements Categorization:* Reviews discussing NFRs can be labeled with quality attributes they discuss (e.g., performance, or usability) [38], [44], [55].
- *Requirements Specification, Design Rationale Capture:* Reviews discussing NFRs can serve as ad-hoc documentations of user requirements [44], [56], [57]. Such specification may be later used by software engineer to justify reasons why certain decisions has been made e.g., why certain mobile phone model has been supported, or why certain security protection mechanism has been implemented [51].
- *Requirements Prioritization, Requested Modification Prioritization:* Added with statistics, reviews reporting NFRs may help development team to prioritize their works [13], [52], [57]. Comparing how frequently NFRs are reported, for how long do user reported them and whether these numbers are increasing or decreasing may indicate users relative importance of these requirements [51], [73].

V. 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')

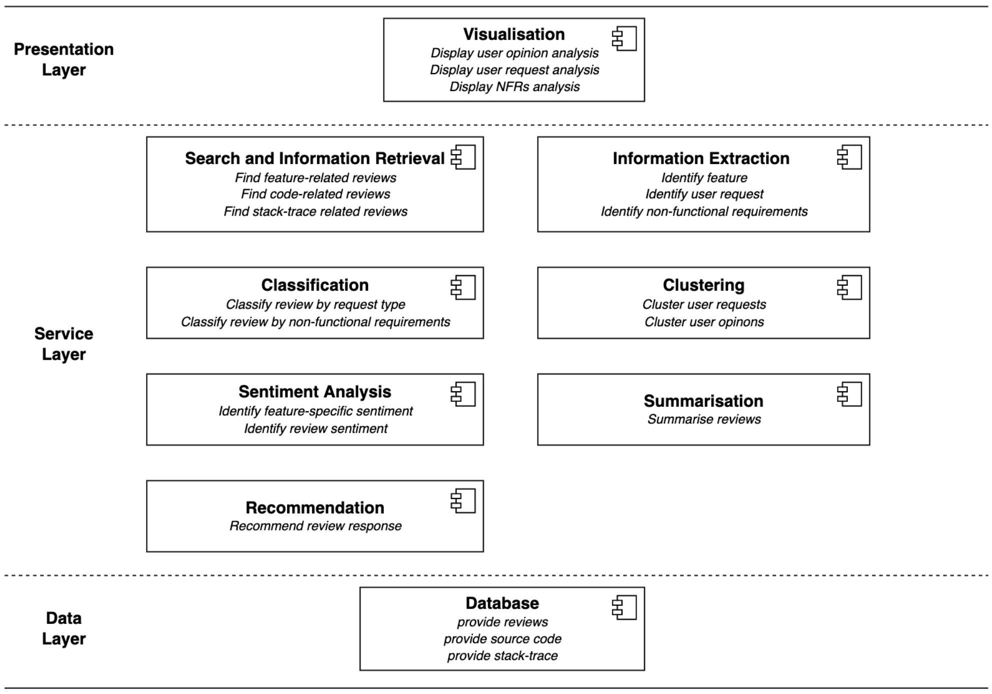
Presentation Layer
Visualisation
Display user opinion analysis
Display user request analysis
Display NFRs analysis
Service Layer
Search and Information Retrieval
Find feature-related reviews
Find code-related reviews
Find stack-trace related reviews
Information Extraction
Identify feature
Identify user request
Identify non-functional requirements
Classification
Classify review by request type
Classify review by non-functional requirements
Clustering
Cluster user requests
Cluster user 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

Fig. 2: Reference architecture for mining app reviews: components illustrated by rectangles; their name written in bold; services they facilitate written in italics.

as described in the modeling and knowledge synthesis in the research methodology (see Sec. III). Figure 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 present examples of how they can be wired to realize the software engineering use cases. For simplicity, we present the realisation of each use case in isolation. The architectural components can, however, be wired in more complex configurations and facilitate, for example, the realisation of a combination of these use cases.

A. Architectural Components and Services

Visualization Component. The visualisation component aids developers in interpreting mined information from reviews. This component provides three services, each generating analytics dashboard per review mining use case: *Display user opinion 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.

Search and Information Retrieval Component. This component provides three services: *Find feature-related reviews* searches for reviews discussing a specific feature of interest. An example of a retrieved review for a queried feature "add reservations" is "Please, improve adding reservations"; *Find code-related 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 related 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 keyboard".

Information Extraction Component. Relevant information can be located at different places within a review (e.g., in the middle of a sentence). The information extraction component locates relevant 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. The classification component organises reviews into predefined categories. The component provides two services classifying reviews based on different categorization schemas. *Classify reviews by request types* classifies 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 mentioned in reviews. An example review classified as "performance" is "The uploading file works very slow, any improvements?".

Clustering Component. The clustering component organizes reviews, their sentences or textual snippets into groups (called clusters) whose members share some similarity, 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. This component provides the service *summarize reviews* 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. Fix the problem with sending messages".

Recommendation Component. This component provides the service *suggest review response*. The service identifies reviews that should be responded to 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. The database stores review mining related data and provide the data to other architectural components. This 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.

B. The Realisation of Software Engineering Use Cases

In the previous section, we have presented the static view of the reference architecture by specifying its components and services. We now present its dynamic view [63]; we present an example of how the architecture could facilitate the use cases defined in Section IV. As for other reference architectures in other domains [32], [65], 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. IV). 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 II presents what components and in what orders realise the intended developer's use of a system in each scenario. Figure 3 illustrates dataflows between components in these scenarios.

TABLE II: 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.*

Use Case 2: Analyse User Request. We demonstrate the main scenario where: (i) an app developer identifies what user requests have been made in reviews, then (ii) an 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 responds 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 III presents what components and in what orders realise the intended developer's use of a system in each scenario. Figure 4 illustrates dataflows between components in these scenarios.

Use Case 3: Analyse Non-functional Requirements. We

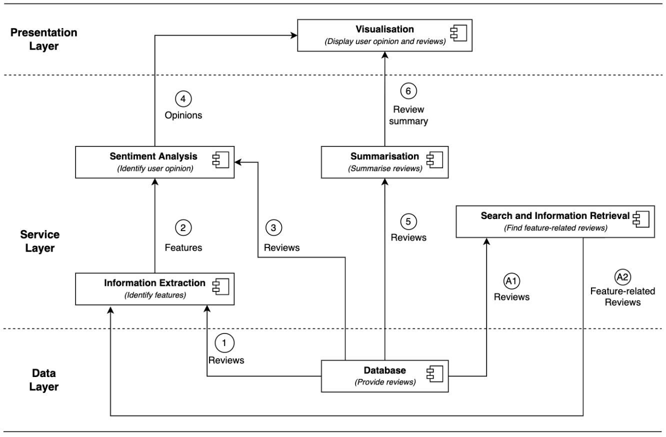

Fig. 3: Dataflow between components for Analyse User Opinion Use Case

TABLE III: 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) 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.

demonstrate the main scenario in which: (i) an app developer classifies reviews by the types of NFRs they convey, (ii) an app developer identifies user sentiment per each NFR category, and then (iii) an 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 IV presents what components and in what orders realise the intended developer's use of a system in this scenario. Figure 5 illustrates dataflows between components.

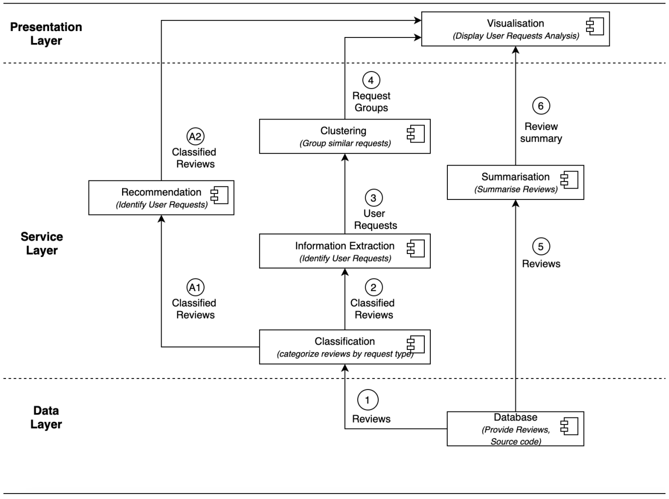

Fig. 4: Dataflow between architectural components for User Request Use Case

TABLE IV: 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.

VI. Validation

We now answer RQ3 (what partial implementations of the reference architecture already exist?) to validate the feasibility of the reference architecture [77]. Following previous work [49], [65], we validate our reference architecture using traceability matrix method [21]; 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 the research literature and commercial websites (see Sect. III). We then related these features to the components of the architecture.

Table V 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

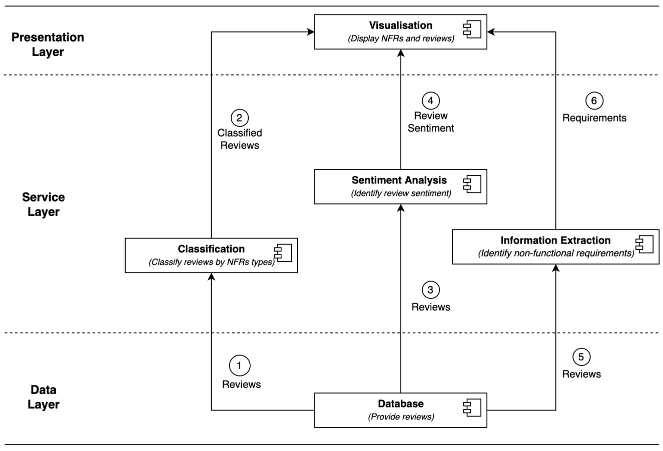

Fig. 5: Dataflow between architectural components for Analyse NFR Use Case

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 commercial 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 commonly implemented service is *Classify reviews by request type*, present in 19 tools. The least commonly 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. Two commercial tools: Appbot [1] and Applysis [4] implement 9 services - the largest number. Whereas seven research prototypes implement only a single service of the architecture [24], [25], [36], [40], [75], [75], [80]. 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 V 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.

VII. 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 V 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 scientific evaluation and 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. IV, which in turn would enable researchers to study how to improve existing techniques or to develop new techniques to better support the use cases.

Table V 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.

VIII. 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 [34]. 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 [23]. We then used standard principles to design and represent our reference architecture and use cases [34], [63].

Another limitation concerns the architecture validation using the traceability matrix method. To gather information about features of research and commercial tools, we have not used the tools neither verified their actual features. We instead relied on information reported in the literature and their vendors' websites. It is not uncommon for tools' authors or vendors to overstate the capabilities of their software.

Another threat to validity concerns the usefulness and the completeness of the identified use cases. Regarding their use-

TABLE V: 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-related reviews	Find stack-trace related reviews	Identify features	Identify user requests	Identify NFRs	Classify reviews by request types	Classify reviews by NFRs	Cluster user requests	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 [45]				✓			✓											2
		CLAP [73], [86]										✓		✓						2
		MARK [87]				✓			✓											2
		ChangeAdvisor [68]					✓					✓		✓						3
		SURF [26], [27]		✓								✓		✓		✓				4
		MARC [42]–[44]									✓	✓	✓			✓				4
		Ardoc [69]										✓						✓		2
		URR [20]					✓					✓				✓				3
		IDEA [35]		✓						✓				✓						3
		RRGen [36]																	✓	1
		OASIS [88]					✓					✓								2
		AOBTM [40]												✓						1
		BECLoMA [70]						✓						✓						2
		Deshpande et al. [24]												✓						1
		Dhinakaran et al. [25]										✓								1
		Scoccia et al. [75]										✓								1
		Shah et al. [76]										✓								1
		Grano et al. [37]						✓				✓								2
		Stanik et al. [80]										✓								1
		Ali et al. [12]										✓						✓		2
	Commercial	Appbot [1]	✓	✓					✓	✓		✓		✓	✓		✓	✓		9
		RankMyApp [8]	✓						✓			✓		✓	✓		✓	✓		7
		WonderFlow [10]	✓	✓	✓				✓			✓	✓				✓			7
		AppAnie [15]	✓						✓								✓			3
		Applysis [4]	✓	✓					✓	✓		✓		✓	✓		✓	✓		9
		AppRadar [5]		✓														✓	✓	3
		AppFollow [3]	✓	✓						✓		✓		✓				✓	✓	7
		AppFigures [2]	✓			✓												✓		3
		Apptopia [6]	✓	✓								✓		✓				✓		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.

fulness, 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 [13].

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 generalized the work upon a large set of studies, including in total 182 papers published between 2012 and 2020 that have been searched and selected using a systematic procedure [23]. 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) [54], [90]. We thus argue the result of our study might to some degree extend these domains.

IX. Related Work

Multiple reference architectures have been proposed for different types of systems and domains, e.g. for big data systems [32], [49], [58], [65], [72], control systems for self-driving vehicles [74], intelligent systems for unmanned vehicle [11], or industrial Internet-of-Things systems [50]. 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 [16]. The use cases are presented using a textual description [11], [32], [49], [72], a use case diagram [74] or a class diagram showing the hierarchy of user requirements [58]. However, 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., [49], [72]). In contrast, our use cases communicate what the users's goals are and how these goals can be satisfied when using a system. Like our study, other reference architecture for other domains present the static and dynamic views of the archtitecture (e.g., [58], [65]). 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 surveyed the literature and collected information to model their reference architecture and use cases [32], [58], [65], [74]). Unlike our work, their methodology was not systematic and lacking details how specific steps were conducted (e.g., [34], [63]). These works, for example, did not report how they selected the relevant literature (e.g., [32]); how many publications they used (e.g., [58], [65]); nor what information they extracted (e.g., [74]). It is also not clear how studies used the collected information to construct their reference models. In contrast, we followed a systematic method to construct an empirically-grounded reference architecture [34]; 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 [50], [74]; interview with stakeholders to assess their usefulness [32]; or by mapping architectural components to available implementations to demonstrate their feasibility [32], [58], [65], [72]. 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 [34].

In summary, our study differs from 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 [23]. The research methodology for elaborating and validating this architecture is more systematic and rigorous compared to previous studies.

X. 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. Previous literature paid little attention to software engineering use cases of their approaches.

To address the problem, we presented a study consolidating the knowledge from a large body of app review analysis 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 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.

The use cases bring to light the benefits and usage of mining app reviews for software engineers. The reference architecture consolidates the diversity of research to gain these benefits and provides a general framework directing the development and evaluation of future research and tools. The mapping shows evidence of the usefulness of app review analysis tools in practice and identifies new gaps in the research and practice.

References

[1] Appbot. https://appbot.co. Accessed: 2021-03-01.
[2] AppFigures. https://appfigures.com. Accessed: 2021-03-01.
[3] AppFollow. https://appfollow.io. Accessed: 2021-03-01.
[4] Applysis. https://applysis.io. Accessed: 2021-03-01.
[5] AppRadar. https://appradar.com. Accessed: 2021-03-01.
[6] Apptopia. https://apptopia.com. Accessed: 2021-03-01.
[7] G2. https://www.g2.com/categories/mobile-app-analytics. Accessed: 2021-03-18.
[8] RankMyApp. https://apprankcorner.com. Accessed: 2021-03-01.
[9] TrustRadius. https://www.trustradius.com/mobile-analytics\#products. Accessed: 2021-03-18.
[10] WonderFlow. https://www.wonderflow.ai. Accessed: 2021-03-01.
[11] J. S. Albus. 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles. In G. R. Gerhart, C. M. Shoemaker, and D. W. Gage, editors, *Unmanned Ground Vehicle Technology IV*, volume 4715, pages 303 – 310. International Society for Optics and Photonics, SPIE, 2002.
[12] M. Ali, M. E. Joorabchi, and A. 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.
[13] 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.
[14] 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.
[15] App Annie. https://www.appannie.com/, 2020. Accessed: 2020-07-01.
[16] P. Avgeriou. Describing, instantiating and evaluating a reference architecture : A case study. 2003.
[17] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI series in software engineering. Addison-Wesley, 2003.
[18] M. 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.
[19] P. Bourque, R. Dupuis, A. Abran, J. Moore, and L. Tripp. The guide to the software engineering body of knowledge. *IEEE Software*, 16:35–44, 12 1999.
[20] 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.

[21] J. Cleland-Huang, O. Gotel, and A. Zisman. *Software and Systems Traceability*. Springer Publishing Company, Incorporated, 2012.

[22] R. J. Cloutier, G. Muller, D. Verma, R. R. Nilchiani, E. Hole, and M. A. Bone. The concept of reference architectures. *System Engineering*, 13:14–27, 2010.

[23] J. Dąbrowski, E. Letier, A. Perini, and A. Susi. Analysing app reviews for software engineering: a systematic literature review. *Empirical Software Engineering*, 27:43, jan 2022.

[24] 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.

[25] 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.

[26] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. 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.

[27] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. 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.

[28] J. Dąbrowski. Supplementary material for System Literature Review: Analysing App Reviews for Software Engineering. https://github.com/jsdabrowski/SLR-SE/, Feb. 2021.

[29] J. Dąbrowski, E. Letier, A. Perini, and A. 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.

[30] J. Dąbrowski, E. Letier, A. Perini, and A. Susi. Mining user opinions to support requirement engineering: An empirical study. In S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. 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.

[31] J. Dąbrowski, E. Letier, A. Perini, and A. Susi. Models Representing How Mining App Reviews Can Support Software Engineering Activities. Zenodo. https://doi.org/10.5281/zenodo.6587502, May 2022.

[32] V. Elvov. Design of Big Data Reference Architectures for Use Cases in the Insurance Sector. Master's thesis, The Technical University of Munich, 2018.

[33] W. Feng, E. F. Crawley, and O. L. de Weck. *Design Structure Matrix Methods and Applications*, chapter BP Stakeholder Value Network, Example 5.5. MIT Press, 2012.

[34] M. Galster and P. 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.

[35] C. Gao, J. Zeng, M. R. Lyu, and I. 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.

[36] 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.

[37] 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*, pages 72–83, 2018.

[38] 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.

[39] 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.

[40] M. A. Hadi and F. 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.

[41] S. Hassan, C. Tantithamthavorn, C. Bezemer, and A. 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.

[42] N. Jha and A. 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, Essen, Germany, February 27, 2017*, 2017.

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

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

[45] 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.

[46] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. 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.

[47] N. Juristo and O. S. Gómez. *Replication of Software Engineering Experiments*, pages 60–88. Springer Berlin Heidelberg, 2012.

[48] B. A. Kitchenham. Procedures for performing systematic reviews. 2004.

[49] J. Klein, R. Buglak, D. Blockow, T. Wuttke, and B. 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.

[50] H. Koziolek, A. Burger, M. Platenius-Mohr, J. Rückert, and G. 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.

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

[52] 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.

[53] C. 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.

[54] S. Lim, A. Henriksson, and J. Zdravkovic. Data-driven requirements elicitation: A systematic literature review. *SN Computer Science*, 2(1):16, 2021.

[55] M. Lu and P. 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.

[56] W. Maalej, Z. Kurtanovic, H. Nabil, and C. Stanik. On the automatic classification of app reviews. *Requir. Eng.*, 21(3):311–331, 2016.

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

[58] M. Maier. Towards a Big Data Reference Architecture. Master's thesis, Eindhoven University of Technology, 2013.

[59] W. J. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *IEEE Trans. Software Eng.*, 43(9):817–847, 2017.

[60] S. McIlroy, W. Shang, N. Ali, and A. Hassan. Is it worth responding to reviews? a case study of the top free apps in the google play store. *IEEE Software*, PP, 2015.

[61] R. R. Mekala, A. Irfan, E. C. Groen, A. Porter, and M. Lindvall. Classifying user requirements from online feedback in small dataset environments using deep learning. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 139–149, 2021.

[62] G. Miner, J. Elder, T. Hill, R. Nisbet, D. Delen, and A. Fast. *Practical Text Mining and Statistical Analysis for Non-Structured Text Data Applications*. Academic Press, Inc., USA, 1st edition, 2012.

[63] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. 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.

[64] E. Noei, F. Zhang, S. Wang, and Y. Zou. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering*, 24(4):1964–1996, 2019.

[65] P. Pääkkönen and D. Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big Data Res.*, 2(4):166–186, 2015.

[66] 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.

[67] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software*, 137:143 – 162, 2018.

[68] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. 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.

[69] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. 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.

[70] 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, 2018.

[71] F. Provost and T. Fawcett. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. O'Reilly Media, Inc., 1st edition, 2013.

[72] G. M. Sang, L. Xu, and P. T. de Vrieze. A reference architecture for big data systems. *10th International Conference on Software, Knowledge, Information Management & Applications*, pages 370–375, 2016.

[73] 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.

[74] J. Schroeder, D. Holzner, C. Berger, C.-J. Hoel, L. Laine, and A. 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.

[75] G. L. Scoccia, S. Ruberto, I. Malavolta, M. Autili, and P. 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.

[76] F. A. Shah, K. Sirts, and D. Pfahl. Simplifying the classification of app reviews using only lexical features. In *Software Technologies - 13th International Conference, ICSOFT 2018, Porto, Portugal, July 26-28, 2018, Revised Selected Papers*, pages 173–193, 2018.

[77] M. Shaw. Writing good software engineering research papers: Minitutorial. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, page 726736, USA, 2003.

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

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

[80] 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.

[81] A. Tang and A. Aleti. Human reasoning and software design: an analysis. 2010.

[82] A. Tang, A. Aleti, J. Burge, and H. van Vliet. What makes software design effective? *Design Studies*, 31(6):614–640, 2010. Special Issue Studying Professional Software Design.

[83] M. Tavakoli, L. Zhao, A. Heydari, and G. 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.

[84] J. Tizard, T. Rietz, and K. Blincoe. Voice of the users: A demographic study of software feedback behaviour. In T. D. Breaux, A. Zisman, S. Fricker, and M. Glinz, editors, *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, pages 55–65. IEEE, 2020.

[85] S. van Oordt and E. Guzman. On the role of user feedback in software evolution: a practitioners' perspective. In *29th IEEE International Requirements Engineering Conference, RE 2021, Notre Dame, IN, USA, September 20-24, 2021*, pages 221–232. IEEE, 2021.

[86] 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*, pages 14–24, 2016.

[87] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. 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, 2015.

[88] L. Wei, Y. Liu, and S.-C. 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.

[89] G. Wierda. *Mastering ArchiMate Edition III: A Serious Introduction to the ArchiMate Enterprise Architecture Modeling Language*. 2017.

[90] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv.*, 54(3), apr 2021.