Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Requirements Intelligence: On the Analysis of User Feedback

Dissertation with the aim of achieving a doctoral degree **(Dr. rer. nat.)**
at the Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Applied Software Technology
Universität Hamburg

submitted by

Christoph Stanik

from Lüneburg

Hamburg, 2020

Day of the oral defense	:	May 12, 2020
Head of the examination commission	:	Prof. Dr. Timo Gerkmann
		Prof. Dr. Peter Kling (deputy)
Evaluators of the dissertation	:	1. Prof. Dr. Walid Maalej
		2. Ph.D. Xavier Franch

To my dear mother.

Acknowledgments

The first person I would like to thank is Walid Maalej. I am deeply grateful that you accepted me as your student and gave me the opportunity to pursue my PhD. Throughout my PhD, you have always supported me with advice and helped me grow as a researcher and an independent, critical thinking person. Through you, I could experience what it means to work in exciting, multinational, and multicultural projects and to travel around the world.

I was fortunate to have worked with many wonderful colleagues at the Applied Software Technology group with whom I share many memorable experiences. It was a great pleasure to work with and to learn from everyone in the group. In particular, I want to thank Daniel Martens, who constantly motivated me to finish my thesis. I also want to thank Marlo Häring, who supported me throughout the whole time of my PhD and helped me advance my research whenever needed.

I further owe special thanks to four of my closest friends, Christiane Frede, Hyeonju (Judy) Bae, Sandra Schmalzbauer, and Asad Sajid. Your constant support, encouragement, and company in the last years made this long journey possible.

Finally, I would like to thank my dear family, who supported me with their endless patience and understanding. Above all, I would like to thank my mother, who has always inspired and motivated me with her strength and willpower. I would also like to thank my two nieces, Fiona and Hannah, whose smiles have always cheered me up.

Abstract

Traditionally, software requirements engineering involved users through workshops, interviews, and observations in the early software development phases. Although beneficial to software teams, these approaches are challenging to carry out continuously and can involve only a limited number of users. In recent years, requirements stakeholders started analyzing explicit user feedback, such as written app reviews, and implicit user feedback like app usage data as continuous sources for requirements-related information. Yet, research highlights that stakeholders rarely use explicit and implicit user feedback in their decision-making process because they receive it in large and unfiltered amounts, making a manual analysis unfeasible. As user satisfaction is crucial for the success of an app, stakeholders need automated approaches for analyzing user feedback to understand user needs and to guide their decision-making. In an interview study, we found that stakeholders need to know how their apps perform, to identify innovative features efficiently, and to understand reported issues and bugs.

This dissertation introduces requirements intelligence, a framework that continuously collects, preprocesses, filters, as well as transforms and matches explicit and implicit user feedback to requirements. The framework aims to generate insights for stakeholders in an integrated interactive visualization. The core enablers for requirements intelligence include two main analysis activities on explicit and implicit feedback: Feedback filtering and feedback to requirements analysis. Feedback filtering is the activity that identifies requirements-relevant feedback, such as problem reports, inquiries, and feature requests. Feedback to requirements extracts the software features users discuss and matches them with the features as documented on, e.g., app pages. We developed and empirically evaluated supervised machine learning approaches for both feedback types and activities. Our approaches rely on crowdsourcing studies for training machine

learning models and benchmarking experiments to identify the optimal machine learning models.

Based on our requirements intelligence framework, we iteratively developed the prototype *feed.ai*. We evaluated *feed.ai* with a total of 15 stakeholders from a major telecommunication company for 12 months. We found that the stakeholders agreed with 92% of the automated filtering results indicating high accuracy. The stakeholders found requirements intelligence beneficial for departments working with user feedback like customer care, marketing, and technology innovation. In a final survey, ten stakeholders anonymously rated *feed.ai*'s functionality, on average, with 4.1/5 and its usability with 4.3/5. They further reported that *feed.ai* helped them to reduce 70% of their time spent on analyzing user feedback, indicating a high effectiveness of our approach.

Kurzfassung

Traditionell werden Nutzer im Anforderungsmanagements in Workshops, Interviews und Beobachtungen in den frühen Projektphasen eingebunden. Obwohl das Anforderungsmanagement von der Einbeziehung der Nutzer mit diesen Ansätzen profitiert, ist es eine große Herausforderung, diese kontinuierlich durchzuführen. Nur eine begrenzte Anzahl an Nutzern können auf diese Weise eingebunden werden. In den letzten Jahren haben Stakeholder deswegen damit begonnen explizites Nutzerfeedback, wie z.B. App-Reviews und implizites Nutzerfeedback, wie z.B. Nutzungsdaten von Apps zu betrachten. Beide Arten von Feedback dienen als kontinuierliche Quellen, die wertvolle Informationen für das Anforderungsmanagement bereitstellen. Die Forschung zeigt jedoch, dass Stakeholder nur selten explizites und implizites Nutzerfeedback in ihren Entscheidungsprozess berücksichtigen. Der Grund ist, dass Nutzerfeedback in großen und ungefilterten Mengen auftritt, weshalb eine manuelle Analyse erschwert ist. Da die Zufriedenheit der Nutzer aber über den Erfolg einer App entscheidet, benötigen Stakeholder automatisierte Ansätze für die Analyse des Nutzerfeedbacks, die sie dabei unterstützen, die Bedürfnisse der Nutzer zu verstehen und Entscheidungen zu treffen. Durch eine Interviewstudie konnten wir belegen, dass Stakeholder wissen müssen, wie ihre App wahrgenommen wird, welche innovativen Funktionen Nutzer diskutieren und wie sie gemeldete Fehler reproduziert werden können.

Diese Dissertation führt ein Requirements Intelligence Framework ein, das explizites und implizites Nutzerfeedback kontinuierlich sammelt, verarbeitet, filtert, sowie transformiert und vergleicht. Ziel des Frameworks ist es, Stakeholdern neue Erkenntnisse in einer integrierten interaktiven Visualisierung aufzuzeigen. Die Analysen des expliziten und impliziten Feedbacks sehen dabei zwei Aktivitäten vor. Die Aktivität „Feedback filtern“ identifiziert anforderungsrelevantes Feedback, wie zum Beispiel Problemberichte, Anfragen und Wünsche für neue Funktionen. Die Aktivität „Feedback zu Anforderungen“ identifiziert

Software-Funktionen die Nutzer beschreiben und gleicht diese mit den von Stakeholdern dokumentierten Funktionen ab. Wir haben beide Aktivitäten durch maschinelles Lernen umgesetzt und empirisch evaluiert. Hierbei basieren unsere Lösungsansätze für beide Aktivitäten auf Crowdsourcing-Studien, mit denen wir die maschinellen Ansätze trainiert haben. Außerdem haben wir durch die Anwendung von Benchmarks identifiziert, wie wir die maschinellen Ansätze optimal konfigurieren können.

Basierend auf dem Requirements Intelligence Framework und unseren Ansätzen haben wir dann den Prototypen *feed.ai* iterativ entwickelt. Wir haben *feed.ai* über 12 Monate zusammen mit insgesamt 15 Stakeholdern eines großen Telekommunikationsunternehmen evaluiert. Wir fanden heraus, dass die Stakeholder den Ergebnissen des automatisierten Filteransatzes zu 92% zustimmten. Die Stakeholder berichteten außerdem darüber, dass die Abteilungen am meisten von *feed.ai* profitieren können, die mit Nutzerfeedback arbeiten, z.B. das Marketing oder das Customer-Relationship. In einer abschließenden Befragung bewerteten zehn Stakeholder die Funktionalität von *feed.ai* mit 4,1/5 und die Benutzerfreundlichkeit mit 4,3/5. Weiterhin konnten sie mit *feed.ai* 70% ihrer Zeit für die Analyse des Nutzerfeedbacks einsparen.

Contents

Chapter 1

Introduction

A painter should begin every canvas with a wash of black, because all things in nature are dark except where exposed by the light.

Leonardo da Vinci

1.1 Problem Statement

Requirements define a potential new system that must be capable of solving a specific problem or need [114]. These requirements are defined by Stakeholders—a group of people that are involved in deciding the system requirements—such as users, customers, developers, and project managers [85]. Stakeholders typically express their needs for that system in natural language, although that language may be ambiguous as each stakeholder has a different perspective and background [60]. Requirements engineering is the process of formulating, documenting, and systematically maintaining software requirements [131]. However, as technical issues may arise during the development or the needs of stakeholders may evolve, requirements engineering is not a single-phase process. On the contrary, requirements engineering is a continuous process that must adapt to change [60]. Requirements elicitation is the step that identifies requirements from stakeholders. Originally, requirements elicitation included users in the software development life cycle by, e.g., performing face-to-face interviews, workshops, and A/B testing [179, 198, 261, 269]. Although these approaches are helpful in early development phases, they only have limited value when trying to continuously identify user needs because they can only involve few users and are only carried out from time to time [47, 148]. In this work, we distinguish between the term stakeholder (e.g., developer and project manager) and the term user, who is the end-user of the system. In the following, we illustrate the risk of not continuously involving users in the area of mobile app development.

As of today, it is hard to imagine a business or a service that does not have any app support—may it be a native, a cross-platform, or a responsive web app. However, the market for mobile apps is highly competitive and dynamic. In the third quarter of 2018, the two leading app stores, the Google Play Store and the Apple AppStore, count over 4 million apps [239]. About 94% apps in the Google Play Store and about 88% apps in the Apple AppStore are freely available [238]. With that many free apps and services such as one-click installations from a desktop computer, the hurdle to download and test any app is low. As a side-effect, the total number of app downloads count as high as hundreds of billions over the last years. The number of existing apps and the number of their downloads are only a glimpse toward the market and its competition created in recent years.

App development organizations face many, but two major challenges to be successful or even survive in the market. One major challenge is visibility—or, in other terms, a high rank in search results. When users search for a new app, they see many alternatives. For example, when searching for "email client" in the Google Play Store, the store returns and shows 250 apps[1]. As comparing all the results of a search query is tedious, users usually only pay attention to the first few visible results [116, 226, 233]. The second major challenge is user satisfaction. Users who are unsatisfied with certain aspects of an app are likely to look for alternatives [14, 74, 256]. Therefore, dissatisfaction can lead to the fall of even previously popular and successful apps [139, 256]. In this environment, continuously monitoring and understanding the changing needs and habits of users is indispensable for the successful survival and evolution of the app. Stakeholders can retrieve and analyze user satisfaction in explicit and implicit user feedback [119, 153, 184].

Explicit user feedback is written feedback about the users' perception and opinion of an app. It is an essential source of information for stakeholders for two reasons. First, besides users considering only the first few apps in the search

[1]as of the 15th of September

results, they also consider the reviews of other users when comparing alternatives [139]. App stores allow users to review apps with a freely-written text and by attaching a rating of one to five stars. Second, user feedback is a helpful resource for stakeholders to understand the users' opinions as it usually contains information such as user experience, problem reports, and feature requests [99, 187]. Apps with more and highly rated reviews get a better ranking in the app stores leading to more visibility and an increased potential in sales and download numbers [73]. User reviews are not a scarce source. As of September 2019, the app *WhatsApp* [254] counts more than 90 million app reviews considering both major app stores. In general, popular apps such as *Facebook*'s main app [66], receive about 4,000 reviews daily [187]. Therefore, if stakeholders collect user feedback over a long period, they can not only learn the users' current opinion but also understand how it developed over time. Alas, the amount of user reviews makes a manual analysis for stakeholders unfeasible. Nevertheless, their number and rich content make them a promising candidate for automated analysis. By addressing user feedback in the development life cycle, stakeholders can improve user's satisfaction with the app and therefore foster more and highly rated reviews [12, 107].

Implicit user feedback collects the usage data of apps and reports on the real behavior of the user. State of the art approaches either log low-level execution events (e.g., stack traces and method calls to augment crash reports) [108], or collect and visualize general high-level usage information (such as the app activation time, location, and device) [27, 251, 259]. However, if we combine usage data with context information such as the location, the name of the current view, and version of the currently running app, stakeholders can use that information to gain insights into app usage behavior. These insights stretch from simple ideas such as when and how often the app is used to more advanced insights such as what are the steps to reproduce for crashing and non-crashing bugs [22, 157].

However, identifying and understanding user needs and satisfaction is a difficult task which, among others, faces the following challenges:

1. **High amount of unstructured and noisy user feedback.** Pagano and Maalej [187] analyzed the 25 top free and 25 top paid apps from each of the 22 categories in Apple AppStore (as of 2012)—1,100 apps in total. When considering these top apps, they receive, on average, 22 reviews per day.

Free apps receive a significantly higher amount of reviews (∼37 reviews/-day) than paid apps (∼7 reviews/day); popular apps such as Facebook receive about 4,000 reviews each day. Further, when looking into user feedback apps receive on Twitter, Guzman et al. [99] show that popular apps receive about 31,000 daily user feedbacks on average. Such numbers make it difficult for stakeholders, particularly those with a popular app in the market, to employ a manual analysis on user feedback to understand their users' needs [98]. Pagano and Maalej [187], as well as Guzman et al. [99] show that more than 70% of the user feedback is not related to requirements engineering and therefore considered as noise.

2. **Diversity of feedback channels and sources.** Users can express feedback explicitly and implicitly. For explicit user feedback, app reviews on app distribution platforms are the most crucial source for requirements engineering as their rating and amount have a direct impact on the rank and, therefore, the visibility of the app [73, 139]. Nevertheless, there are other channels, such as Twitter or forums, where users give informative feedback [99, 176, 180, 257]. Similar to the findings of Pagano and Maalej [187], who analyzed app stores, Guzman et al. [100] show that about 42% of tweets addressing an app are improvement requests and are, therefore, relevant information for stakeholders. In contrast, implicit user feedback considers app usage data that represents the real behavior of the users. Usage data is a series of event-based data triggered by user interactions with a device that we can enrich with context data. We can gather this type of feedback with hardware sensors such as the GPS signal or software sensors that can track information like clicks and scrolling [117]. Therefore, while explicit user feedback is natural language, implicit user feedback is a series of events. As a consequence, we need different analysis methods for both types of feedback to cope with the diversity of the data.

3. **Different stakeholders need different types of feedback analysis.** Research shows that stakeholders long for automated analysis of user feedback [59, 115, 119, 151, 184, 234, 249] as they see many opportunities. These opportunities encompass, for example, less manual effort, quicker identification and resolution of bugs, and getting inspiration for new fea-

tures. However, software projects often have diverse stakeholders and roles, such as developers, project managers, product owners, and users. When analyzing user feedback to integrate the analysis results in the development life cycle, each stakeholder may have different information needs. In a published study [151], we interviewed nine stakeholders covering different roles, such as software developers, requirements engineers, and project managers, to understand how we should present user feedback. All of the interviewees agreed that they need help to filter noisy user feedback (e.g., "great app."), but they have different needs when it comes to the representation of detailed analysis results. For project managers, it is more important to see a few charts summarizing the overall performance of the software and to compare it with, e.g., previous releases or versions on different platforms (e.g., Android vs. iOS). Stakeholders in the requirements engineering process want inspiration for the release plan by understanding requests for new features but also want to know what to improve based on problem reports. Software developers, on the other hand, are more interested in the details of each app review, including information like a description of a problem attached with details about the affected hardware and software version.

1.2 Objectives and Contribution

In this work, we aim to support stakeholders to understand their users' needs and in their decision-making process by automatically analyzing user feedback. We analyze explicit and implicit user feedback to provide analytics insights into users' opinions and needs. For this, we introduce requirements intelligence, a framework about the analyses of both feedback types, and an integrated interactive visualization. Based on qualitative studies, we identified stakeholder needs for automated analysis of user feedback and developed the feed.ai prototype. We evaluated feed.ai with a major telecommunication company for over 12 months. In the following, we summarize our contributions in more detail:

Requirements intelligence. As to the best of our knowledge, there is no standard definition of requirements intelligence we attempt to define and scope that term in Chapter 4. For defining the term, we look into the related work of requirements engineering, software analytics, and business intelligence & analytics, which we consider closely related fields. We then draw parallels to requirements intelligence and distinguish more clearly between these fields.

Besides defining the term, we also introduce a requirements intelligence framework. The core enablers for the framework are the analysis of explicit and implicit user feedback, which we realized with machine learning approaches. As the analysis of both feedback types follows a similar machine learning approach, we contribute with the description of our machine learning pipeline. The framework includes three activities for each type of user feedback. The first activity is about data collection and preprocessing. Part of this activity is getting a quantitative and qualitative understanding of the data. The second activity, feedback filtering, extracts requirements-relevant user feedback for stakeholders. The third activity, feedback to requirements, extracts features users write about and use, and matches them with requirements (features) stakeholders document. Finally, stakeholders can get analytical insights into user feedback in the integrated interactive visualization that combines the analysis results of both feedback types.

In Chapter 3, we motivate the requirements intelligence framework with two qualitative studies that explore stakeholder needs for analyzing user feedback. We summarize related work and report on an interview study, both investigating user feedback usefulness and challenges for stakeholders for including user feedback in their work. The studies encompass, in total, the opinions and experiences of 90 stakeholders who generally agree that they need an automated analysis of user feedback to improve their app continuously and to stay competitive in the market.

Automated analysis of explicit user feedback. As stated earlier, popular apps receive written user feedback thousandfold daily. As this feedback can come from different platforms, in different languages, in diverse quality, and as it is usually noisy, stakeholders need automated approaches to get access to the feedback that is relevant for them.

Chapter 5 concerns the second requirements intelligence activity, feedback filtering, and presents an approach that filters user feedback from app stores and twitter in the English and Italian languages. In total, we collected more than six million app reviews and tweets, out of which we sampled about 6,000 English app reviews, 10,000 English tweets, and 15,000 Italian tweets. Participants in a crowdsourcing study then labeled the sampled data into the categories problem report, inquiry, and irrelevant. We then applied traditional machine learning and deep learning to find the best configuration for the automated categorization. Our study shows that there is no single best approach for the categorization and that if stakeholders want to support multiple languages, they have to put more effort into machine learning feature extraction. After checking the results against a test set, we achieved F1 scores of up to .89 with a ROC AUC score of .86, confirming the validity of the results.

Chapter 6 addresses the third requirements intelligence activity, which maps the requirements-relevant user feedback to features. In this chapter, we present an approach that identifies the features users discuss in feedback and the features stakeholders document on app pages. In a final step, the approach matches the features users discuss with those stakeholders document to generate indicators, for example, for popular and missing features. For identifying the stakeholder documented features, we created an evaluation set of ten app descriptions covering 197 manually extracted features in total. For identifying features users discuss, we selected the 400 most recent reviews from five apps in our data set and manually extracted 244 features. We compared our automated approach to state of the art and found that for the extraction from app descriptions, we achieve an F1 score of .46. For extracting features users address in app reviews, we achieve an F1 score of .35. Both approaches improve the state of the art.

Automated analysis of implicit user feedback. Besides the written opinion of users, they can also provide feedback in the form of their usage behavior. Implicit user feedback analyzes user interactions with a device and the usage context. In contrast to explicit user feedback, implicit user feedback does not report on the subjective perception of users but their real behavior. Therefore, this feedback type can provide truthful and rich insights into how users use apps. However, already a few users can generate a million interaction events, making a manual analysis of this feedback unfeasible.

Chapter 7 concerns the second requirements intelligence activity, feedback filtering, and presents a study that filters user implicit feedback for the usage context. The approach focuses on a rather hard problem for identifying the usage context, which is an automated identification of private and professional device usage. More straightforward context filters can be, e.g., identifying the user's location, which some APIs like the Google Maps API already provide. We performed a crowd-study with 18 participants, who labeled their mobile device usage for two weeks. The participants generated more than 88,000 context events and 6,486 labeled sessions. We then applied machine learning based on the collected and labeled context data to find the best configuration for the classification into private and professional device usage. We further experimented with a minimization of the context event types to foster user privacy. For the within-users analysis, we achieved an F1 score of .94 using a simple Decision Tree classifier. In our between-users analysis, we achieved an average F1 score of .95 using all context events and .87 using the minimized number of context types.

Chapter 8 addresses the third requirements intelligence activity, which maps the requirements-relevant user feedback to features. The approach aims to identify the app features user use automatically based only on their interactions with the device. For that, we ran a crowd-based study for 18 days, including 55 participants, to get a labeled dataset for a machine learning-based approach. We found that depending on the app and the number of labeled data, we can achieve an average F1 score of .70 with an ROC AUC of .61 for the within-app analysis. For some apps, we reached a score of about .85 for both metrics. In the between-apps analysis, we combined the labeled data of multiple apps to check if the feature usage is similar. Our results go up to an F1 score of .86 and an ROC AUC of .91 for the feature *listen to music*. The average scores are similar to those of the within-app analysis. We further analyzed the most significant machine learning features to check whether humans can understand how the classifiers came to their solutions. Our results show that with a few human-readable machine learning features, we can reach explainable and accurate conclusions (e.g., frequent text edits reveal writing a message).

Integrated interactive visualization. The integrated interactive visualization, as part of the requirements intelligence framework, is the interface for stakeholders to get analytics insights from user feedback. In Chapter 9, we combine our contributions into the web-based prototype feed.ai that is an implementation of the requirements intelligence framework. We detail on the architecture and the dynamic models of feed.ai. For the prototype, we contribute with an open-source implementation that interested parties can use and extend. In total, the prototype consists of more than a dozen microservices, for which we provide the source code, technical documentation, endpoint description, and give hints for developers who want to work on the microservices.

Chapter 10 evaluates the prototype. One aim of this work is to provide a prototype that stakeholders can use, which makes sense in their workflow. For this, we cooperated with a major Italian telecommunication company that used the prototype for 12 months. We developed the prototype iteratively and evaluated it in five iterations by either using surveys, interviews, or performing phone meetings. In a final survey, the telecommunication company invited ten stakeholders from different departments, such as marketing and customer care, to evaluate the latest iteration of the prototype. The stakeholders found the prototype useful for their workflow and highlighted that it was easy to use and supported them to get meaningful insights. They further reported that *feed.ai* helped them to reduce 70% of their time spent on analyzing user feedback.

1.3 Scope

We discuss the scope of this work by describing the following topics.

Domain focus. In this work, we support stakeholders in requirements engineering in involving users by analyzing user feedback. We do not support traditional requirements engineering activities and methods for involving users, although we introduce them briefly to differentiate ourselves from them.

We further decided to focus on the app development domain for our analysis activities to show one coherent approach. However, we evaluated our developed prototype in the telecommunication domain, as we had access to the stakeholders of a cooperation partner for 12 months.

Requirements documentation. Our user feedback analysis focuses on software features and, therefore, functional requirements. We simplify the reading of this work by using both terms interchangeably.

One of our objectives is to match features users address in feedback with the features stakeholders documented. As we focus on the app development domain, we only include app pages (i.e., the app description) as a requirements documentation source. We do not cover other sources for requirements like requirements specifications, issue trackers, or media and news articles.

User feedback. For the explicit user feedback analysis, we cover app reviews and tweets. Therefore, we include representatives for the native review platforms of apps, as well as a representative from social media. We did not include other sources of explicit user feedback such as emails, Facebook, Reddit, or news media comments. Further, we do not consider other types of explicit user feedback, such as oral feedback (e.g., phone calls, interviews, and brainstorming).

In particular global organizations receive explicit user feedback in multiple languages. The challenge with different languages is that the effort for their analysis may differ. Therefore, we decided to cover English, which is the most analyzed and one of the most spoken languages, as well as Italian, because we cooperated with a major Italian telecommunication company. We do not cover other languages in this work.

Implicit user feedback is about the interactions with a device and the usage context. We can collect and analyze that feedback from many different platforms, such as smartphones, tablets, TVs, desktops, and web browsers. We decided to focus on mobile devices as it allows us to align the analysis results for both feedback types (i.e., app reviews for explicit user feedback). We further scope the implicit user feedback analyses on feedback collected from the Android platform, which is less restrictive than, e.g., iOS, and because of its high market share.

1.4 Structure

We structured the remainder of this work into the following three parts.

Part I *Problem.* The idea of this part is to motivate and scope the problem this work addresses in detail.

In Chapter 2, we first introduce the foundations that discuss the challenges stakeholders in requirements engineering face when involving users. In the same chapter, we also describe the definitions and conceptual models of explicit and implicit user feedback.

Chapter 3 presents two qualitative studies that identify stakeholder needs for analyzing user feedback.

In Chapter 4, we propose requirements intelligence, a framework that addresses stakeholders' needs to support their decision-making process based on user feedback analytics.

Part II *Core Enablers for Requirements Intelligence.* The second part of this work is about the analyses of explicit and implicit user feedback, which are the core enablers for requirements intelligence. Requirements intelligence defines the two activities *feedback filtering* and *feedback to requirements* for both the explicit and implicit user feedback analysis.

Explicit user feedback analysis.

In Chapter 5, we introduce an approach to filter explicit user feedback for the categories irrelevant, problem report, and inquiry.

Chapter 6 presents an approach that extracts features (requirements) from user feedback and app pages and matches them.

Implicit user feedback analysis.

In Chapter 7, we describe an approach that filters implicit user feedback based on the usage context of an app.

Chapter 8, reports on an approach that helps stakeholders to learn which features users use.

Part III *Evaluation.* Here we focus on implementing and evaluating a requirements intelligence prototype based on the findings of Part I and Part II.

In Chapter 9, we report on feed.ai, our requirements intelligence prototype. We discuss its technical details like the architecture, including the dynamic models, and detail on the integrated interactive visualization, which is part of the requirements intelligence framework.

Chapter 10 discusses our 12-month evaluation of feed.ai with a major telecommunication company.

Chapter 11 concludes the work by summaries our contributions and by stating ideas for future work.

Part I

Problem

Chapter 2

Foundation

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.

Richard Feynman

We summarize the foundation and the two core concepts of this work. First, we introduce requirements engineering in Section 2.1, where we define the term requirements, the requirements engineering process and its activities, and user involvement. Then, we introduce explicit user feedback in Section 2.2. It is one of the main concepts and core enablers for requirements intelligence. We define the term, discuss the conceptual model of explicit feedback, and conclude with a summary of platforms that we consider in this work for collecting explicit feedback. After that, we follow the same structure for implicit user feedback in Section 2.3. We define the term, discuss the conceptual model of implicit feedback, and conclude with the platform we consider in this work to collect implicit feedback. Eventually, we summarize our conclusions of the chapter in Section 2.4.

2.1 Requirements Engineering

In this work, we aim at supporting the decision-making process of stakeholders by providing analytical insights into user feedback. Users are, among other roles, one specific type of a stakeholder that impacts the development of a system such as software. As requirements engineering is the process of deciding what to build, and stakeholders are those making these decisions, it is the field on which this work has the biggest effect. Therefore, the goal of this section is to provide a brief introduction to the field of requirements engineering. For this, we start by defining the terms *requirement*, *stakeholder*, and *requirements engineering.* We follow by explaining the activities in the requirements engineering process in Section 2.1.1. Then, we discuss user involvement in requirements engineering and explain why it plays a crucial role in the success of software apps in Section 2.1.2.

2.1.1 Definition

Requirements define what a potentially new system must be capable of to solve a specific problem or need [114]. Literature separates requirements into functional and non-functional requirements [31, 43, 131, 154, 212]. Functional requirements are "[...] a function that a system or system component must be able to perform." [114] as the IEEE standard states. In the literature, there is a broad consensus about this term, as Glinz [84] found. For example, Glinz shows literature that provides similar definitions, such as requirements are "what the product must do" [206] or "what the system should do" [229]. In the Book “Managing Requirements Knowledge”, Maalej and Thurimella state that the term requirement is similar to **features**, but that it has a larger scope and more technical focus [154]. For non-functional requirements, there is no agreed-on definition, as Glinz [84] showed in 2007. In this work, we follow the definition given by Glinz, which is “A non-functional requirement is an attribute of or a constraint on a system” [84].

Stakeholders are a group of people that are involved in deciding the requirements of a system. Typical stakeholders are, among others, customers, developers, project managers, and **users** [85]. The decision-making process of stakeholders is often about choosing between alternative requirements that can apply to solve a specific issue [154, 196]. Stakeholders have various backgrounds as they can come from, e.g., business, marketing, law, project management, design, and development, and therefore, have diverse roles and tasks [154]. Stakeholders typically express their needs for a system in natural language, although that language may be ambiguous as the stakeholders’ perspectives and backgrounds differ [60].

Requirements engineering is the process of formulating, documenting, and systematically maintaining software requirements [131]. Similarily, Sawyer, Sommerville, and Viller define requirements engineering as "Requirements engineering is concerned with the discovery of required properties (the requirements) and their transformation into a form which will serve as the basis for development of a product which will exhibit those properties." [219]. However, as technical issues may arise during the development or the **needs of stakeholders may evolve**, requirements engineering is not a single-phase process. On the contrary, requirements engineering is a continuous and iterative process that must adapt to change [26, 60, 219]. The literature highlights the importance of change management [154] as requirements engineering is a complex process that might not get all of the necessary requirements right from the start of a project. Reasons for this are, among others, that clients are unsure what the final product should be, stakeholders have tacit knowledge, or any event or wrong decision during the project may impact the solution. Poorly managed requirements lead to risks in the project, such as exceeding the budget or failing the project, making requirements engineering one of the most critical processes in projects. Boehm and Basili [25] found that identifying and fixing a problem in the software after its deployment often costs 100 times more than identifying it during the requirements and design phase [25]. Second, the authors also show that projects spend about 40-50% of effort in avoidable reworking, which again highlights the importance of requirements engineering.

Requirements engineering is a process composed of activities. Kotonya and Sommerville define the activities of the requirements engineering process as requirements elicitation, requirements analysis and negotiation, requirements documentation, and requirements validation [131]. Ramesh and Cao [202], as well as Martin et al. [160] highlight that the initially proposed requirements engineering process is linear, which is in contrast to the idea that requirements engineering must adapt to change. In a later publication, Sawyer, Sommerville, and Viller [219] introduced potential improvements to the requirements engineering process, suggesting a cyclic representation of the activities. We cite the authors' suggestion, which determined the following three activities as part of the cyclic representation [219].

1. **Requirements elicitation**. Given a statement of organisational needs and other inputs, different requirements sources (stakeholders, domain experts,

operating regulations etc.) are consulted to understand the problem and the application domain. The resulting requirements may be incomplete, vaguely expressed and unstructured.

2. **Requirements analysis and validation**. The requirements discovered during the elicitation phase are integrated and analysed. This is designed to identify problems such as missing information, inconsistencies and requirements conflicts.

3. **Requirements negotiation**. Problems discovered during analysis need to be resolved. The analysts and stakeholders clarify their understanding and consider possible solutions. This may require negotiation to establish the necessary trade-offs. The elicitation of further requirements information and the initiation of a further cycle may be necessary.

Besides these cyclic activities, the authors suggest adding **requirements management** as a cross-section activity that also stretches over the whole development process. It is the activity that is responsible for handling both new emerging requirements and general changes to exiting requirements. Requirements management ensures requirements traceability and the enforcement of the change. The overall goal of representing the requirements engineering process cyclic is to cope with the three challenges of the difficulty of elicitation, changes to requirements, and the limitations of time and costs [219]. The literature agreed on the importance of change management in requirements engineering, which led to several publications supporting a cyclic representation of the process [26, 111, 199]. Yet, the suggested cyclic representation is difficult to include in modern agile project management and development as the activities do not map to the agile concept [202].

Agile requirements engineering is the consequence of the non-sequential process of requirements engineering in modern agile projects. Its goal is to be highly adaptive to change in agile projects, which are iterative by nature. Ramesh, Cao, and Baskerville [202] present an empirical study discussing how the four requirements activities *requirements elicitation*, *requirements analysis and negotiation*, *requirements documentation*, and *requirements validation* of Kotonya and Sommerville [131] can be used in agile projects. Ramesh et al. [202] suggest the following. Perform *requirements elicitation* iteratively with face-to-face meetings with stakeholders instead of collecting requirements once before the start of the development. Similarly, *requirements analysis and negotiation* are also iterative and should be carried out with face-to-face meetings, constant planning, and extreme prioritization. The authors further suggest that *requirements documentation* is not a formal process but that requirements are documented informally as, for example, a **list of features** or stories. *Requirements validation* includes the **users** of the system to validate if the requirements reflect the **users' current needs**. Table 2.1 summarizes and compares the agile and traditional requirements engineering process.

Table 2.1: Traditional and agile approach for requirements engineering (RE) activities (taken from Ramesh et al. [202]).

RE activities	Traditional RE	Agile RE	Agile practices to support RE activities
Requirements elicitation	Discovering all the requirements upfront	Iterative: requirements evolve over time and are discovered throughout the development process	Iterative RE, face-to-face communication
Requirements analysis and negotiation	Focus on resolving conflicts	Focus on refining, changing and prioritizing requirements iteratively	Iterative RE, face-to-face communication, constant planning, extreme prioritization
Requirements documentation	Formal documentation contains detailed requirements	No formal documentation	Face-to-face communication
Requirements validation	The consistency and completeness of requirements document	Focus on ascertaining whether the requirements reflect current user needs	Review meetings, face-to-face communication

Conclusion 1. Requirements define what a potential system must be capable of to solve a specific problem or need. Functional requirements are sometimes also called the features of the system. Stakeholders, for example, the users of the

system, decide about the requirements. Requirements engineering is a crucial process in projects as poorly managed requirements lead to major risks such as exceeded budgets or the failure of a project. These risks include, for example, requirements not known prior to the project or tacit knowledge of stakeholders. Projects can cope with the risks if the requirements engineering process can adapt to change. Therefore, researchers suggested a cyclic representation of the process, which later led to agile requirements engineering, which better integrates into modern agile projects. Agile requirements engineering constantly involves the users to validate if the requirements match the users' needs. Still, the research discussed suggests interacting with users in physical face-to-face meetings, which lead to significant issues as we elaborate in the following.

2.1.2 User Involvement

We introduced requirements engineering, its activities, and concluded that it is an iterative and continuous process. Stakeholders decide about the features/requirements of a software system. To understand if stakeholders build the right system, they have to validate the requirements with users frequently. The users in this context are the end-users that are interacting with the system. Users are often an underestimated stakeholder, although probably the most crucial for the project's success because they decide about its rise and fall [132, 139, 140, 256]. The objective of user involvement is to identify and address user needs, such as the features they need, as early and frequently as possible, for continuously improving the software [44, 63, 132]. User requirements describe how the software can help users achieving their goals and how it satisfies their needs in the context of use [132]. Therefore, when involving users, we have to understand their needs and their context [44, 132].

Traditional user involvement. Projects that involved users in their requirements engineering process typically elicited user needs by performing workshops, surveys, observations, or interviews [179, 198, 221, 261, 269]. For example, workshops and semi-structured interviews are methods that can help understanding user needs as they allow to interact with the users and allow for asking follow-up questions. Observations, on the other hand, are a method that aids in identifying the context of use and how the users interact with the system. Although research shows that these methods are helpful, they usually apply in the early stages of the project and are challenging to perform continuously [15, 47, 265]. There are local challenges of the methods, such as time limitations in observations, no in-depth responses in surveys, or emphasizing the opinion of extroverts in workshops [221]. To overcome the local challenges, research suggests using various elicitation methods [221]. Besides the local challenges, there are also global challenges that apply to multiple requirements elicitation methods. These challenges are, for example, reaching a representative sample of users [47] and time and budget constraints to perform them repeatedly [221].

Modern user involvement. With the rising popularity of the internet, many organizations started publishing their software products online, either on their own website or app distribution platforms like app stores. In parallel, users discovered internet platforms, such as forums and blogs, to write and discuss their opinions online. Nowadays, app distribution platforms provide feedback mechanisms on which users also write their opinion about software products by, e.g., reviewing and rating them [74, 187]. Besides writing reviews on app stores, users also discuss software products on social media channels like Twitter [99, 257]. Research shows that the explicit feedback stated on these platforms is valuable for stakeholders as users vividly report problems they have and features they wish [78, 99, 112, 187, 191]. Involving users by including user feedback in the requirements engineering process helps to increase user satisfaction and, consequently, influences the product's success positively [74, 107, 140, 190]. **We call the written opinion of users explicit user feedback**.

Kujala et al. [132] and Bettenburg et al. [22] highlight that requirements engineering should also consider the context of use. If the deployed software collects information about the context, such as the operating system version or the app version, stakeholders can use that information to, e.g., understand if only a specific version is affected. Martens and Maalej [157] show that users sometimes also state context information in explicit user feedback. When also collecting the interactions with the software, we can create an understanding of under which circumstances users provide feedback and which steps within the software led to either a crashing or non-crashing bug [148, 153]. Studies also show that context and interaction data help improve usability by conducting usability testing [54, 55, 146, 150, 153]. **We call the context and interaction data implicit user feedback**.

Research shows that combining explicit user feedback from app distribution platforms and social media channels like Twitter help to understand users better [176]. Further, leveraging the combination of explicit and implicit user feedback can lead to either new requirements or requirements of better quality [119, 153, 184]. The strength of explicit and implicit user feedback is that it comes continuously after the deployment of the app, covering a wider range of users than traditional methods can. We also cover the user perception with explicit user feedback and the real usage behavior and context with implicit user feedback.

Although continuously involving users by analyzing explicit and implicit feedback is a valuable source of information, it comes with challenges. Here, we want to highlight two frequently stated challenges. First, users write their opinion about popular software thousandfold a day [99, 187]. Similarly, collecting implicit user feedback generates a million data points with only a few users [184]. These amounts of feedback make a manual analysis unfeasible [99, 184, 187]. Second, the majority of user feedback is rather uninformative and of low quality [180]. Regarding explicit user feedback, such uninformative feedback is often spam, praise for the app, insulting comments, or a repetition of the star rating in words [109, 187]. As a consequence, practitioners seek automated support to filter the feedback and to identify the requirements-relevant information [151, 234, 249].

Conclusion 2. User involvement plays a critical role in requirements engineering because user satisfaction decides about the rise and fall of the software. Successfully involving users in requirements engineering means to understand user needs and the context of use. Involving users with only traditional requirements elicitation methods limits the positive impact users can have on the app. Traditional methods are hardly representing the whole userbase and are too costly and time-consuming to be employed continuously. Projects that carefully include explicit user feedback from app distribution platforms and social media can foster the success of the app. Users use these platforms to state their opinion, including the problems they face and the features they wish. Besides analyzing the written opinion of users, research shows that requirements engineering must also consider implicit user feedback. Again, traditional methods to understand the context of use, such as observations, come with many challenges. To continuously understand how and in which context users use the app, stakeholders need to involve users by analyzing their behavior from within the software. So far, stakeholders do not have automated tool support for analyzing explicit and implicit user feedback continuously. That gap makes it unfeasible to include users continuously on a large scale and misses many opportunities to foster the project's success.

In the next sections, we detail on explicit and implicit user feedback by defining the terms and by introducing their conceptual models.

2.2 Explicit User Feedback

The goal of this section is threefold. First, we define the term *Explicit User Feedback*. Then, we introduce its conceptual model and detail the concepts. Finally, we describe the platforms providing explicit user feedback that we consider in this work.

2.2.1 Definition

Originally, requirements engineering considered users in the software development lifecycle by, e.g., interviewing them face-to-face, performing workshops, and A/B testing [179, 198, 261, 269]. Although these approaches are helpful in the early phases of development processes, such as the design phase, they only have a limited value after the deployment of the software. As soon as the software is available on the market, the userbase might become diverse and enormous, making it challenging to identify the changing needs of most users with, e.g., physical workshops [47, 265].

However, with the rise of social media and app distribution platforms, there are plenty of options for users to discuss, review, and rate software apps online. Research states that users use these platforms actively for this purpose. Pagano and Maalej [187] highlight that popular apps receive about 4,000 reviews daily. Iacob and Harrison found that free apps receive, on average, 45.5 daily reviews [112]. Further, Guzman et al. [99] show that popular apps receive, on average, about 31,000 daily tweets. Besides users providing such an amount of feedback, that feedback contains valuable information for stakeholders, as users tend to use these platforms to report bugs, request new features, or describe their overall experience with the app [4, 99, 187]. The following shows an example for a bug reported by a user in an app review:

> "[...] I loved SoundHound before the last update. Now I can't view my history. I've tried several times [...] I just give up. Back to using Shazam. I hope this gets fixed."

This written feedback not only reports the problem faced (cannot open the history view) but also states that the user considers the app of a competitor, though still hoping that the issue gets fixed. In this example, we learn about the issue itself and the importance of the feature for the user [134]. Identifying such feedback helps stakeholders to resolve them quickly [107].

The following shows an example of a feature request in an app review:

> "You can't even view you sorted contact groups. That needs to change. What's the point? "

Here, the user addresses a feature (sort contact groups) that seems to be without value if not used by a not yet implemented feature (view sorted contact groups). Stakeholders can take such feedback as an inspiration for new requirements and may add them to one of the following releases of the app [175, 249]. Besides the high-level identification of problems and feature requests, research shows that identifying the concrete features users address in their feedback helps stakeholders to improve their requirements [48, 218]. If stakeholders match the features addressed by the users with their internal or publicly documented requirements, they can, for example, better identify popular or missing features.

Stakeholders understood that user feedback contains valuable information and started to manually and automatically extract that information to improve their apps during the full development lifecycle [119, 151, 153]. If stakeholders use this feedback, they can increase the overall satisfaction with the app and therefore foster its success in the market [74, 140, 256].

Online user feedback has the advantage that it comes continuously. Therefore, organizations can analyze the performance of their apps from the perspective of their users, either after each release, daily, weekly, or according to any time period. The disadvantage of such feedback is that users are typically no technical experts and are often vague about encountered problems. They do not know what technical information a developer needs to work on the reported issue [157]. Another disadvantage is that reviews are opinions about the *perceived* experience and, therefore, prone to emotions, which leads to often uninformative feedback such as: “I hate this app” [101, 158, 187]. Therefore, the major challenge in analyzing explicit user feedback is to filter for requirements-related information.

Definition 2.1: Explicit User Feedback

Explicit User Feedback is the written opinion of a user. In its base-form, it is freely-written text addressing any aspect of a software or service provided by an entity (e.g., company or single developer). It may have a numerical rating attached that expresses the users’ overall satisfaction and experience.

Conclusion 3. Explicit user feedback is a continuous source of valuable, requirements-related information. Users report problems and feature requests that, if addressed by stakeholders, can improve the overall satisfaction with the app. Users address features in their feedback that stakeholders can match with the requirements of the app to understand if users, e.g., address existing features.

2.2.2 Conceptual Model

We describe the overall conceptual model and detail its key concepts.

Figure 2.1 shows the conceptual model for explicit user feedback. The *Explicit Feedback* concept is the core of the model. A user can write explicit feedback directed at exactly one particular app. Apps typically have a description that reports on the features (requirements) they offer. The feedback can come from different sources, such as social media or app stores. Feedback generally contains a submission date, is written in a particular language, and has a freely-written *Body Text*. Usually, feedback platforms allow users and developers to communicate with each other by replying to feedback. Depending on the feedback platform, it may also contain a picture (e.g., in tweets) or a video (e.g., in Amazon reviews and tweets). A review posted on any app store must contain a star rating (one to five stars). If being more specific about the platform, the Google Play Store

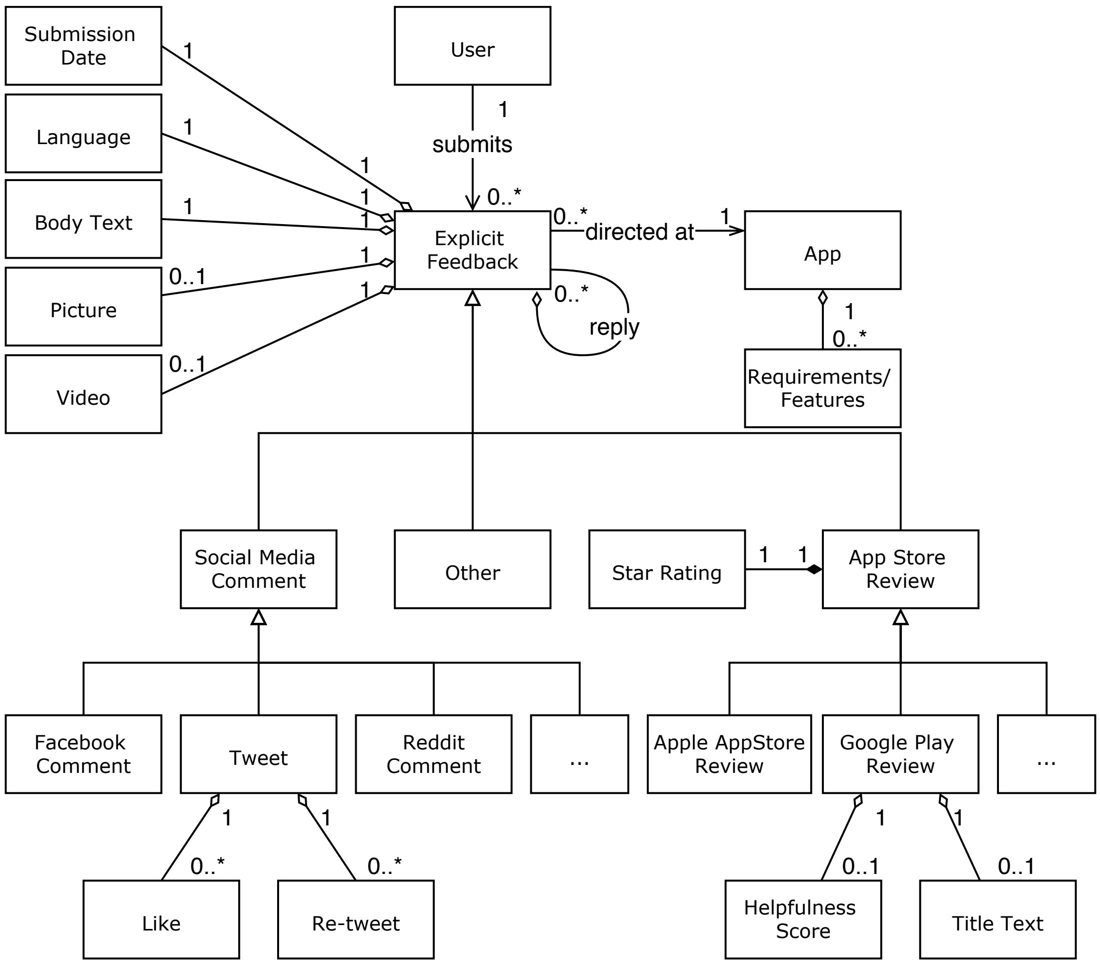


Figure 2.1: Conceptual model for explicit user feedback.

also allows writing a *Title Text* for the review. Further, other users can rate a review as helpful, whereas it may also contain a *Helpfulness score*. Social media, on the other hand, was not originally designed as a platform for explicit user feedback, but users express their opinions about apps on these platforms, too. Tweets are the concept on Twitter in which users write their feedback. Tweets do not contain a star rating but a *Body Text*, a *Submission Date*, potentially *Replies*, a *Picture*, a *Video*, as well as a counter for how often other users liked the Tweet or re-tweeted it.

Conclusion 4. The conceptual model of explicit user feedback highlights that there are different platforms on which users can submit feedback. The content of the feedback varies with the platform. The ability to write a freely-written text (*Body Text*) is a shared feature among all platforms. Therefore, this work focuses on analyzing the written feedback of users, ignoring additional concepts such as videos or pictures.

Conclusion 5. The conceptual model of explicit user feedback shows that we distinguish between social media comments, app store reviews, and other platforms. Although the conceptual model gives several examples for these platforms, we decided to focus on Twitter and app stores (Apple AppStore and Google Play Store). We decided to focus on only some platforms, as each platform needs a custom analysis because the language and the (meta) data available differs. We cover representatives for the native review platforms of apps, as well as a representative from social media.

2.2.3 Platforms for Explicit User Feedback

We decided to include the two app stores (Apple AppStore and Google Play Store) and Twitter as the platforms for user feedback in this work. As the conceptual model for explicit user feedback shows, the data available on these platforms differ. Therefore, we explain these platforms in more detail.

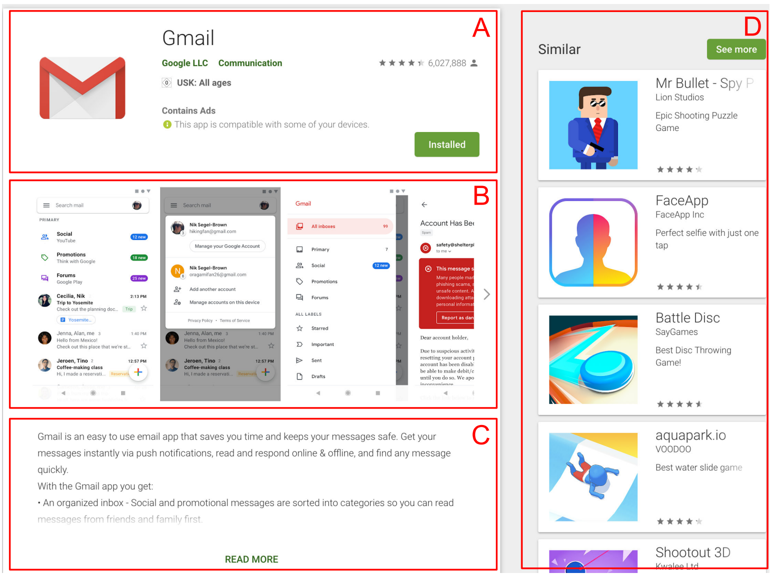
Google Play Store

One of the major app stores is the Google Play Store [88]. It is the primary app distribution platform for Android and contains about 2.5 million apps [239]. Each app has a dedicated app page, which contains the features (requirements) it provides, as well as some describing information. The combination of Figure 2.2, 2.3, and 2.4 show an example of a complete app page. In this section, we chose to create the figures from the web view of the Google Play Store. Although the visualization is different on mobile devices, both versions contain the same information.

The top part of an app page contains descriptive information, as shown in Figure 2.2. Figure 2.2-A contains meta-information such as the name of the app, its average rating, the number of ratings, the USK, as well as a status indicating if the user already installed the app (otherwise, the user can install it via this button). In Figure 2.2-B, stakeholders can upload an introductory video, as well as screenshots showcasing the app. Figure 2.2-C shows the description of the app, which usually describes the features it offers. Stakeholders can fill the description of their app with free-written text and minimal visual supportive elements such as bullet points. Features are often presented with bullet points [118]. The following is an excerpt from the app description of *Wunderlist* [258] detailing the features it provides:

- Create all the lists you need and access them from your phone, tablet and computer
- Easily share lists and collaborate with family, friends and colleagues
- Start conversations about your to-dos
- Attach photos, PDFs, presentations and more
- Share the work and delegate to-dos
- Setting a Reminder ensures you never forget important deadlines (or birthday gifts) ever again
- Organize your projects for home, work and everywhere in between with Folders

On the right side of an app page (Figure 2.2-D), Google automatically generates a list of similar apps.

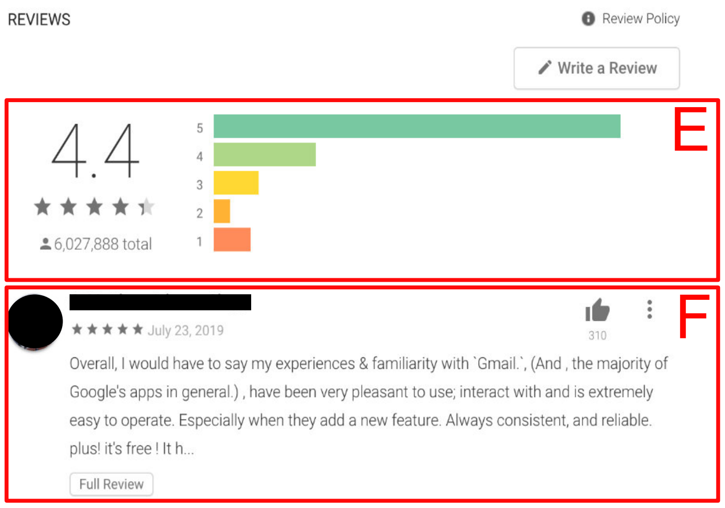


Gmail is an easy to use email app that saves you time and keeps your messages safe. Get your messages instantly via push notifications, read and respond online & offline, and find any message quickly.
With the Gmail app you get:
• An organized inbox - Social and promotional messages are sorted into categories so you can read messages from friends and family first.

Figure 2.2: App page—descriptive information on the app Gmail found on the Google Play Store.

When scrolling below the app description, the user can read app reviews of other users. Figure 2.3 gives an example of that part. On top (Figure 2.3-E), it shows the overall average rating, the total number of ratings, as well as the rating distribution. Below that (Figure 2.3-F), the user can see a list of user reviews. A user review usually contains an avatar, the username of the review authors, the helpfulness score (on the right), as well as the written body text. Previously, the Google Play Store also allowed users to write a title associated with their reviews, but they removed that feature. However, as the title is still part of older reviews, they are often part of scientific studies.

Sometimes, stakeholders reply to user feedback, which would also be visible in this part. On app distribution platforms, stakeholders can give exactly one reply per review, but both the user and the stakeholder can update their text at any time. The consequence is that these platforms do not record and show the whole history of the conversation between the user and the stakeholder but only the most recent snapshot of it. Replies on app distribution platforms have been proven as a powerful tool, as users tend to update their review and rating when their issues got recognized and addressed. Stakeholders often use replies to either ask for more details or to promise a fix in the future [107].



[redacted]

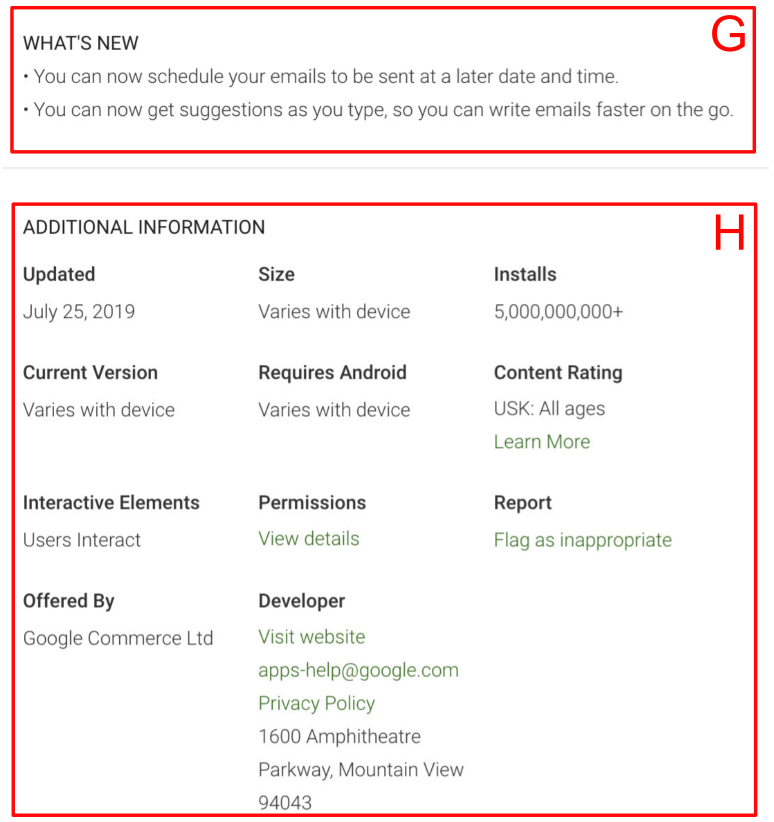
[redacted]
★★★★★ July 23, 2019

Overall, I would have to say my experiences & familiarity with `Gmail.`, (And , the majority of Google's apps in general.) , have been very pleasant to use; interact with and is extremely easy to operate. Especially when they add a new feature. Always consistent, and reliable. plus! it's free ! It h...

Figure 2.3: App page—user reviews on the app Gmail found on the Google Play Store.

User reviews on app stores are a potential source for the requirements engineering process, as they contain valuable information, as described earlier. Additionally, users also use reviews to address app features directly. As a consequence, we have available the app page, which documents the features and the reviews that discuss them [3, 118, 126, 217].

The bottom of an app page (see Figure 2.4) presents additional meta-information. In Figure 2.4-G, stakeholders can add information about what changed in the recent update of the app. That field usually addresses the features that were previously broken or are newly added. Figure 2.4-H contains information about the app and the organization, such as the date of the last update, the estimated number of installations, and the address of the organization.



WHAT'S NEW

• You can now schedule your emails to be sent at a later date and time.

• You can now get suggestions as you type, so you can write emails faster on the go.

ADDITIONAL INFORMATION

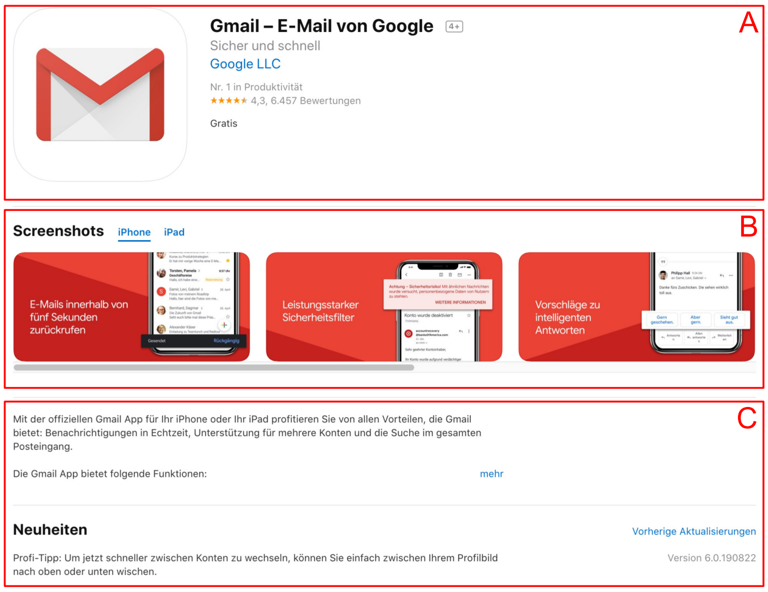
Updated	**Size**	**Installs**
July 25, 2019	Varies with device	5,000,000,000+
Current Version	**Requires Android**	**Content Rating**
Varies with device	Varies with device	USK: All ages Learn More
Interactive Elements	**Permissions**	**Report**
Users Interact	View details	Flag as inappropriate
Offered By	**Developer**	
Google Commerce Ltd	Visit website apps-help@google.com Privacy Policy 1600 Amphitheatre Parkway, Mountain View 94043	

Figure 2.4: App page—additional meta-information on the app Gmail found on the Google Play Store.

Conclusion 6. App pages contain two important types of information for stakeholders. First, they document and advertise the features, and therefore the requirements, of the app. Second, they contain user reviews that address either the existing or missing features. Stakeholders can use that information to identify, for example, popular (often discussed) features, problematic features, or feature requests. Further, they can also use the reviews to update the description of the app to create awareness for certain features. As stakeholders can reply to user reviews, they can influence the review and the rating of the user if they address their concerns.

Apple App Store

The Apple AppStore is the second largest app distribution platform and the main platform for iOS devices, containing about 1.8 million apps [239]. Similar to the Google Play Store, the Apple AppStore also has dedicated app pages. Both app stores contain very similar information for users. Nonetheless, we detail on the information available in the Apple AppStore by presenting the same app example as for the Google Play Store. The combination of Figure 2.5, Figure 2.6, and Figure 2.7 represents one complete app page as it is visible to users.


A
B
C
Screenshots
iPhone
iPad
E-Mails innerhalb von fünf Sekunden zurückrufen
Gesendet
Rückgängig
Leistungsstarker Sicherheitsfilter
Konto wurde deaktiviert
WEITERE INFORMATIONEN
Vorschläge zu intelligenten Antworten
Gern geschehen.
Aber gern.
Sieht gut aus.
mehr
Vorherige Aktualisierungen

Gmail – E-Mail von Google 4+

Sicher und schnell

Google LLC

Nr. 1 in Produktivität

★★★★★ 4,3, 6.457 Bewertungen

Gratis

Mit der offiziellen Gmail App für Ihr iPhone oder Ihr iPad profitieren Sie von allen Vorteilen, die Gmail bietet: Benachrichtigungen in Echtzeit, Unterstützung für mehrere Konten und die Suche im gesamten Posteingang.

Die Gmail App bietet folgende Funktionen:

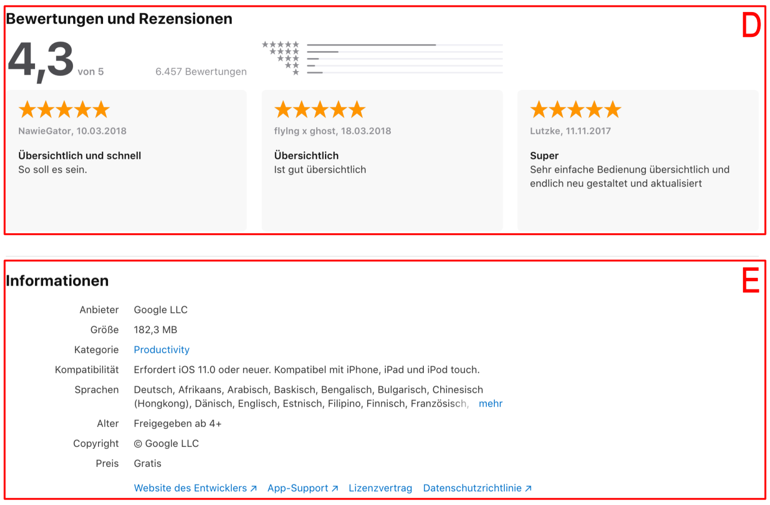
Neuheiten

Profi-Tipp: Um jetzt schneller zwischen Konten zu wechseln, können Sie einfach zwischen Ihrem Profilbild nach oben oder unten wischen.

Version 6.0.190822

Figure 2.5: App page—descriptive information on the app Gmail found on the Apple AppStore.

Figure 2.5 shows the top part of an app page. In Figure 2.5-A, we see the app name, its average rating, the total number of ratings, its price, as well as the name of the organization developing the app. In Figure 2.5-B, stakeholders can showcase screenshots to advertise specific parts of the app. Also, similar to the Google Play Store, the Apple AppStore contains a description of the app, as well as release information for the most recent update (see Figure 2.5-C). Organizations use this part to describe and showcase the features of the app to give users an impression of what they can expect from it. The app description is a freely-written text, which can use simple text structures like bullet point listings.



Bewertungen und Rezensionen

4,3 von 5 6.457 Bewertungen

★★★★★
NawieGator, 10.03.2018
Übersichtlich und schnell
So soll es sein.

★★★★★
flying x ghost, 18.03.2018
Übersichtlich
Ist gut übersichtlich

★★★★★
Lutzke, 11.11.2017
Super
Sehr einfache Bedienung übersichtlich und endlich neu gestaltet und aktualisiert

Informationen

Anbieter	Google LLC
Größe	182,3 MB
Kategorie	Productivity
Kompatibilität	Erfordert iOS 11.0 oder neuer. Kompatibel mit iPhone, iPad und iPod touch.
Sprachen	Deutsch, Afrikaans, Arabisch, Baskisch, Bengalisch, Bulgarisch, Chinesisch (Hongkong), Dänisch, Englisch, Estnisch, Filipino, Finnisch, Französisch, mehr
Alter	Freigegeben ab 4+
Copyright	© Google LLC
Preis	Gratis

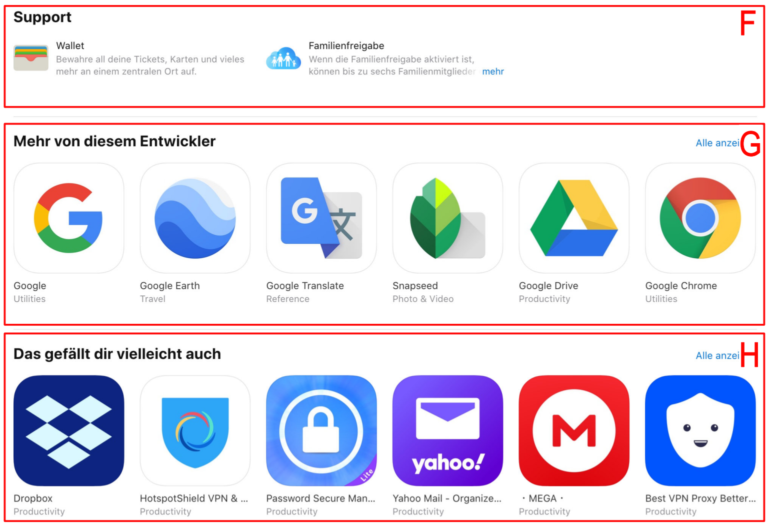
Website des Entwicklers ↗ App-Support ↗ Lizenzvertrag Datenschutzrichtlinie ↗

Figure 2.6: App page—user reviews and meta-information on the app Gmail found on the Apple AppStore.

Figure 2.6 shows the average rating, the distribution of the ratings, the total number of ratings, as well as written reviews by the users (Figure 2.6-D). The displayed information is, again, similar to what the Google Play Store shows. The main difference is the style of presentations. Figure 2.6-E shows general information of the app and the organization developing the app, such as the size of the app, its category, as well as the address of the organization. Users can utilize this part to check whether their devices are compatible with the app or to visit the website of the organization to get more information.

On the bottom of the app page, the Apple AppStore shows which third-party features of Apple it supports (Figure 2.7-F) like the *wallet* feature to store tickets or *family sharing* options. Figure 2.7-G contains some other apps from the same organization and allows users to browse their full list of apps. In Figure 2.7-H, users can see similar and other apps recommended by Apple that they might be interested in using.

Conclusion 7. Similar to the Google Play Store, the Apple AppStore also provides app pages that contain, among others, the features the app provides and user reviews that discuss the app and its features. As a consequence, we can use app pages as features (requirements) documentation for apps and analyze the user reviews concerning the features.



Bewahre all deine Tickets, Karten und vieles mehr an einem zentralen Ort auf.

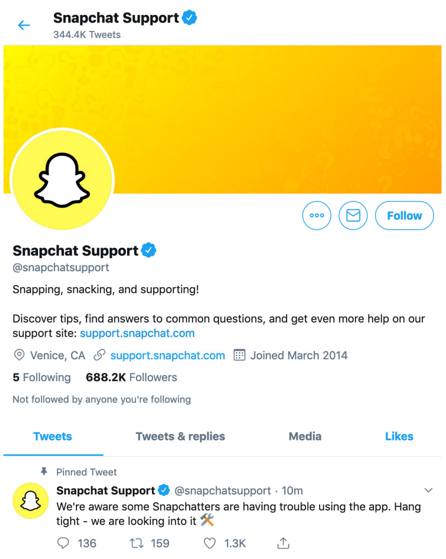
Wenn die Familienfreigabe aktiviert ist, können bis zu sechs Familienmitglieder

Figure 2.7: App page—references on the app Gmail found on the Apple AppStore.

Twitter

Twitter describes itself as a platform that is about "what's happening in the world and what people are talking about right now."[247]. It is a social media platform on which users can have a profile (account) and send public messages with hashtags or private direct messages to other users. Some organizations advertise their app on Twitter and provide support via dedicated support profiles. One example is *Snapchat Support* [228]. A profile, such as *Snapchat Support*, has a public name, a unique identifier, a place for a description, and some other meta-information. The meta-information is about the location of the profile owner (in this case, the primary location of the Snapchat organization), a link to their website, the number of profiles followed, and the number of followers (see Figure 2.8a).

Users can send messages to these support profiles, as depicted in Figure 2.8b. The figure shows the conversation between a user of Snapchat and the support



Snapchat Support
344.4K Tweets

Follow

Snapchat Support
@snapchatsupport

Snapping, snacking, and supporting!

Discover tips, find answers to common questions, and get even more help on our support site: support.snapchat.com

Venice, CA support.snapchat.com Joined March 2014

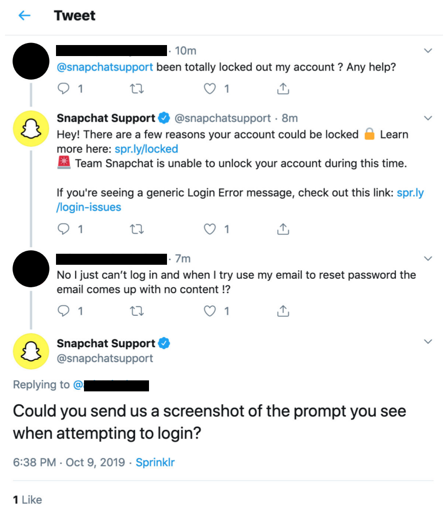
5 Following 688.2K Followers

Not followed by anyone you're following

Tweets | Tweets & replies | Media | Likes

Pinned Tweet
Snapchat Support @snapchatsupport · 10m
We're aware some Snapchatters are having trouble using the app. Hang tight - we are looking into it 🛠
136 159 1.3K

(a) Twitter profile.



Tweet

[redacted] · 10m
@snapchatsupport been totally locked out my account ? Any help?
1 1

Snapchat Support @snapchatsupport · 8m
Hey! There are a few reasons your account could be locked 🔒 Learn more here: spr.ly/locked
🚨 Team Snapchat is unable to unlock your account during this time.

If you're seeing a generic Login Error message, check out this link: spr.ly/login-issues
1 1

[redacted] · 7m
No I just can't log in and when I try use my email to reset password the email comes up with no content !?
1 1

Snapchat Support
@snapchatsupport

Replying to @[redacted]

Could you send us a screenshot of the prompt you see when attempting to login?

6:38 PM · Oct 9, 2019 · Sprinklr

1 Like

(b) Twitter conversation.

Figure 2.8: A profile and conversation example from Snapchat's Twitter support profile.

profile. In that example, the user reports a problem, and the support account replies to that message to offer help. On Twitter, the reply functionality allows the user and the support profile to write several replies, which leads to a conversation of any length. The whole conversation history is publicly available as long as no involved party deleted a message. Twitter conversations follow a tree-like structure. In addition to the two involved parties, any other user can participate and contribute to public conversations. For that, the user may answer the main conversation or starting a sub-conversation by answering a specific tweet. Figure 2.9 illustrates an example of a real Twitter conversation. The figure shows that the support profile sent a public tweet that addressed no particular user. Then, two different users replied to that tweet, each reporting a problem they face. The support profile then replies to each user individually, which initiates two distinct conversations.

Conclusion 8. Organizations that have a support profile on Twitter have the unique opportunity to interact with their users. Usually, users message these profiles in case they encountered a problem with the app and cannot solve it on their own. Another advantage of Twitter is that stakeholders can involve users by starting a conversation with them. As Twitter allows for conversations of any

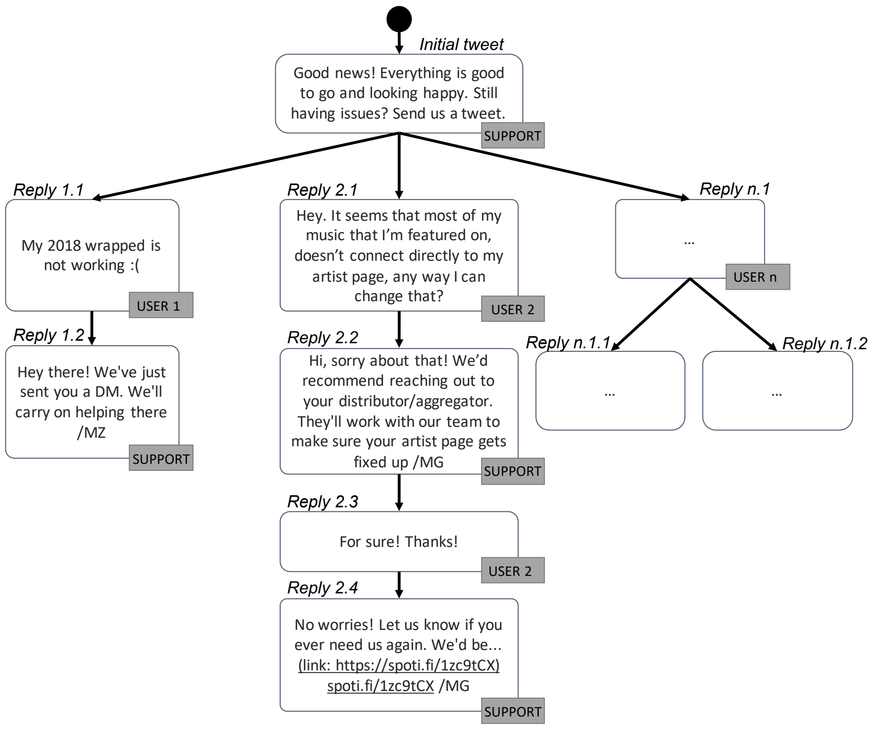

Initial tweet
Good news! Everything is good to go and looking happy. Still having issues? Send us a tweet.
SUPPORT
Reply 1.1
My 2018 wrapped is not working :(
USER 1
Reply 1.2
Hey there! We've just sent you a DM. We'll carry on helping there /MZ
SUPPORT
Reply 2.1
Hey. It seems that most of my music that I'm featured on, doesn't connect directly to my artist page, any way I can change that?
USER 2
Reply 2.2
Hi, sorry about that! We'd recommend reaching out to your distributor/aggregator. They'll work with our team to make sure your artist page gets fixed up /MG
SUPPORT
Reply 2.3
For sure! Thanks!
USER 2
Reply 2.4
No worries! Let us know if you ever need us again. We'd be... (link: https://spoti.fi/1zc9tCX) spoti.fi/1zc9tCX /MG
SUPPORT
Reply n.1
...
USER n
Reply n.1.1
...
Reply n.1.2
...

Figure 2.9: Example of a tweet conversation tree between a support profile and users.

length, stakeholders can ask for details like context information to get a better idea of their users' problems.

2.3 Implicit User Feedback

The goal of this part is threefold. First, we define the term *Implicit User Feedback*. Then, we introduce its conceptual model, including the description of its main concepts. Finally, we discuss the Android operating system as a platform for collecting implicit user feedback.

2.3.1 Definition

As introduced earlier, explicit user feedback is the written opinion of users. Therefore, users submit that feedback type to report their subjective perception of an app and its features. Though that feedback provides insights for stakeholders, such as understanding problems, that feedback is noisy [99, 187], emotional [101, 158], and often misses information about the usage context [157]. For stakeholders, it is crucial to understand the opinion of the users and to know about the usage context [22, 132] and how users interact with a device and app [153]. That is essential because it provides objective feedback that reports the actual usage. Traditionally, stakeholders gathered this feedback with observations. Yet observations have several disadvantages as they do not scale to the extent that covers a representative set of users [120, 201, 263], are hard to perform continuously [47, 265], and may introduce bias because the users feel observed and are not necessarily in their natural environment [13, 221]. Stakeholders have the unique opportunity to use monitoring approaches within their apps. Monitoring is an unobtrusive way of capturing, e.g., event logs in a software system by inserting code in their running system [71]. Requirements monitoring is one particular field that monitors the execution environment of an app or the interactions and the context of the users to, e.g., validate requirements and to identify the changing needs of the users [45, 71, 207]. Two questions that this field addresses are "when do our systems need to change?" and "how can we use monitored data to orchestrate the change?" [71]. By employing monitoring approaches, stakeholders can identify user interactions with a device in the given context.

User interactions, such as clicking, scrolling, and changing views, are an objective way to report how users interact with apps. The advantage is that provides an unbiased view and chain of events that stakeholders can analyze to gain insights into, e.g., the features used [75, 119]. Analyzing user interactions has many benefits, such as:

- Extract steps to reproduce for (non-) crashing bugs [86, 209, 210].
- Identify popular, inefficient, and frustrating navigation paths [53–55].
- Understand session durations and which views users stay longest [16, 21, 27, 55].
- Discover types of users [16, 122, 267].
- Learn usage behavior [68, 122].

However, to fully understand app usage, stakeholders also have to analyze the context of the user and the device itself. Context data such as the location of the user or software specifics, such as the version of an app, are essential information to learn about usage scenarios [22, 40, 157]. For example, if stakeholders want to extract steps to reproduce a problem from the interactions with the app, they also need to know the app version and the platform as they may have a different design and navigation flow [169, 268]. We look at two definitions of context. Maalej defines context as "the set of all events and information, which can be observed and/or interpreted in the course of the work, except those events and pieces of information that constitute the change (i.e. the main output of the work)." [149]. Pagano defines context as "refers to all current and past conditions and events which influence the interaction of a user with an application or the execution of an application on a system" [185]. In this work, we further summarize the aggregation of context data (e.g., location and app version) for one particular user interaction as the usage context. The usage context is a higher abstraction and describes, for example, the device (smartphone vs. smartwatch, including the operating system) or private vs. professional use.

Similar two the two authors, we also distinguish between user interaction and context. As a consequence, we understand implicit user feedback as the aggregation of interaction data with the describing context from both the user and the device.

Definition 2.2: Implicit User Feedback

Implicit User Feedback is the aggregation of user interaction events with a computing device which is enriched with context information about the user and the device for every recorded interaction event.

Conclusion 9. Implicit user feedback is objective feedback that reports on the actual usage of a user with a device. Stakeholders can analyze user interactions with a device and the usage context to understand how their users are using the apps and their features. They can use that information to derive either new requirements or improvements to existing requirements.

2.3.2 Conceptual Model

We describe the overall conceptual model as well as the details for its key concepts.

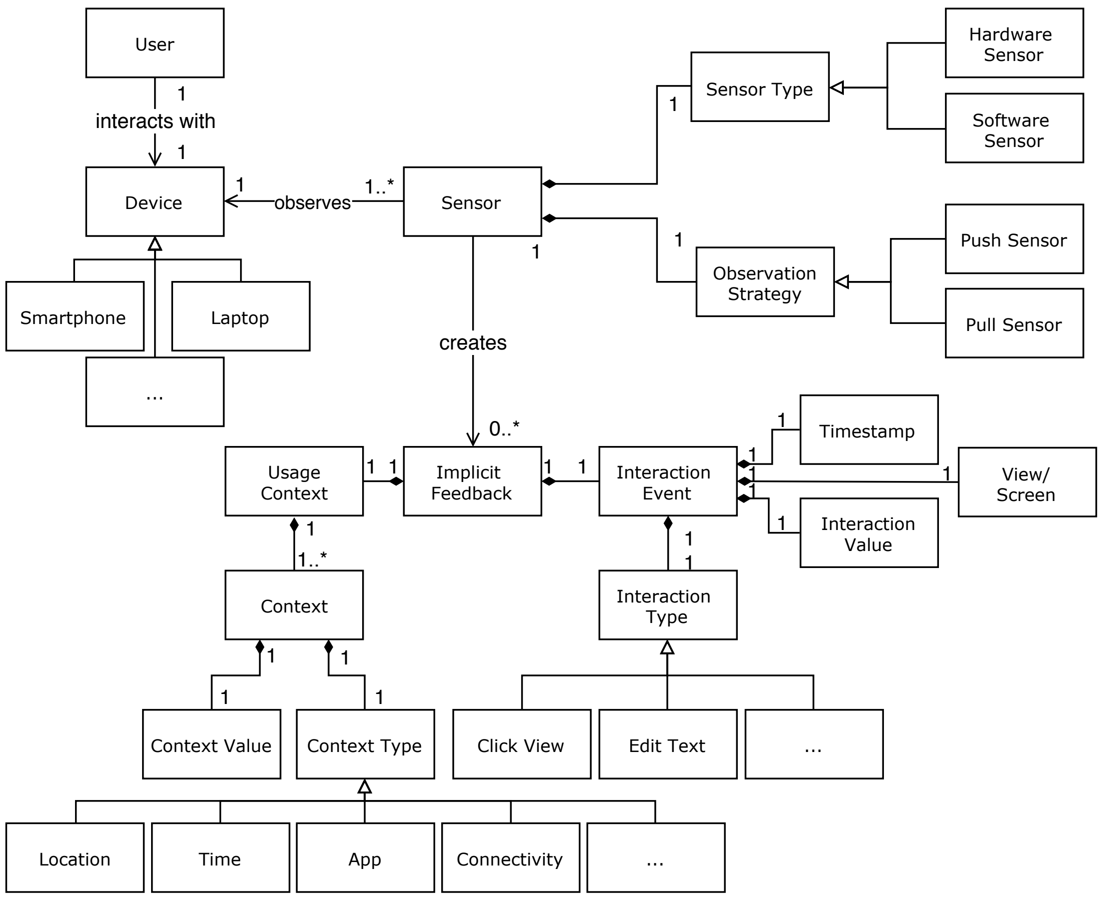


Figure 2.10: Conceptual model for implicit user feedback.

Figure 2.10 shows our conceptual model for implicit user feedback as a domain model. On the top left, the figure shows that the underlying assumption of the model is a user interacting with a device. The device can be anything running software, ranging from, e.g., smartphones, laptops, smartwatches, or smart TVs. In this work, we focus on mobile apps and therefore consider smartphones and tablets (mobile devices).

Sensors observe the interactions with the device and its state. In particular, we distinguish between hardware and software sensors, as well as push and pull sensors, as described in the following.

a) *Hardware and Software Sensors* determine the source of the context event. Hardware sensors provide their data by hardware, such as a built-in GPS sensor or an integrated accelerometer. Software sensors provide data through the operating system (OS). A typical software sensor is, e.g., collecting information about the currently running software like the visible app or the current time and date.

b) *Push and Pull Sensors* describe *how* we observe the changes to the user's and device's state to generate context events. Pull sensors request information from the operating system in a defined interval. The before mentioned example of a software sensor that identifies the currently displayed app is also a pull sensor since we have to ask the operating system for such information. Push sensors, on the other hand, subscribe to certain events of the operating system like a change in the network connection.

User interactions with the device create *Interaction Event*s. Interaction events contain a type, timestamp, the name of the view it appears in, as well as an interaction value. For example, if a user clicks on a send button in an email client, the interaction type is “click”, the interaction value could be “send mail” (the text visible on the button), the view of the event may be called “ComposeMail”, and the timestamp of the event could be “2019-10-30 T 10:45:00 UTC”. For each interaction event a sensor observes, it creates a single *Implicit Feedback*.

Conclusion 10. Sensors can observe user interactions and the usage context. We either need push or pull sensors depending on how the sensor can access the context and interactions (observation strategy). The particular context data and interaction event decide if the sensors can pull or push data. As a consequence, the implementation of a sensor depends on how and what it shall observe.

The implicit feedback created by a sensor contains exactly one interaction event and one *Usage Context*. The usage context is a collection of context data that describes the complete state of the device and the user for one particular interaction event. Therefore, the usage context holds one to many contexts. For example, if a user sends an email (interaction event), the usage context may contain several context information, such as the name of the app, the app version, the location of the user, as well as the operating system. The concept context holds a context value for one particular context type. For example, a context type can be the operating system, while the context value is Android.

Conclusion 11. Usage context describes the whole state of the device and user by aggregating all available context data for one particular interaction event.

2.3.3 Platforms for Implicit User Feedback

We can collect implicit user feedback from any software platform that runs an operating system that either provides APIs for collecting interaction events and context, or that pushes this data (push and pull sensors). In this work, we focus on mobile apps, and therefore, we consider a mobile operating system for collecting implicit user feedback. For several reasons, we consider Android as the operating system for deploying a monitoring app for the data collection. First, it allows us to collect implicit user feedback for any app in the background unobtrusively; besides granting permissions to the monitoring app, the user will not be disturbed by it. Second, Android provides various APIs that allow us to collect a wide range of different context types. Third, the accessibility framework of Android [92] enables us to collect fine-grained user interaction events such as clicking, scrolling, and focussing a view. Fourth, Android's market share ranges around 87% in 2019, allowing us to cover a high proportion of mobile users [113].

In the following, we summarize the interaction types and context types that we can collect on the Android platform. The summarization of these types is not exhaustive for the operating system but lists all the types we analyzed throughout this work.

In contrast to, e.g., app pages, the data available is not directly visible but documented in the API references of the operating system. Table 2.2 shows nine different interaction types that we collect with Android's accessibility framework [92]. For each of the nine interaction types, we added a description to the table that explains when a sensor generates an interaction event for that particular type. The table highlights that we can collect coarse-grained interaction events like clicking, but also fine-grained interactions involving clicking, such as focussing a view or selecting and item in a list.

Table 2.3 gives examples of context types and values we can collect on Android devices. Again, we limited the number of elements in the table to what we analyze in later chapters. The table shows that we can, for example, identify the current foreground app and its version. That information is helpful for understanding in which app user interactions occur. Further, the table shows examples of context types related to connectivity, such as how the device is connected (e.g., mobile internet or Wi-Fi) and the number of available Wi-Fi networks. The combination of these context types and values define the usage context.

Table 2.2: App-independent interaction events in the Android accessibility framework.

#	Interaction Type	Description
1	Click View	Gets fired whenever a participant clicks on any kind of view element, such as a button. This event also covers long pressed clicks
2	Notification	Records the appearance of either a notification (typically shown in the top bar) or toast message (short lasting notification shown on the bottom of the phone).
3	Edit Text	Reports whenever a text within an input field ("EditText" field) is changing.
4	Select Text	Fired when a user is changing the currently selected text.
5	Scroll View	Triggered, whenever the user is scrolling through a list of items or on a web page
6	Change Content	Reported whenever any content of a window changes (e.g., a user archives an email and it gets removed from the list of received emails).
7	Focus View	Triggered when the user or the app sets the focus on a view element, e.g., the app automatically focuses on the input field.
8	Change Window	Fired whenever a section of the user interface changes to something visually distinct (e.g., clicking on a tab that exchanges the main content of the view).
9	Select View	Triggered when a user is selecting an item in a list.

As part of the EU FP7 project MUSES (Multiplatform Usable Endpoint Security) [174], we developed an opensource library to collect interaction events and context data on Android [172]. The overall goal of MUSES was to foster corporate security by reducing the risks introduced by user behavior. The project observes mobile device usage to enforce company security policies. The project's GitHub page [173] contains several repositories for collecting and analyzing implicit user feedback. It also contains repositories for collecting context data on iOS and Windows. These repositories are limited in the data they can collect and are therefore not further considered in this work.

Conclusion 12. Android is the mobile operating system with the highest market share and the highest flexibility for collecting interaction events and context data. On Android, we can collect implicit feedback unobtrusively by deploying an app that collects the feedback in the background.

Table 2.3: Excerpt of app-independent context data accessible on Android.

#	Context Type	Context Value (example)
1	Operating version	Android
2	Operating system version	4.0
3	System language	English
4	Active app identifier	com.google.android.gm
5	Active app version	5.0.1.1
6	Latitude and longitude	[53.551086,9.993682]
7	Date and time	2019-10-30 T 10:45:00 UTC
8	Running background services	[com.waze,de.blitzer.plus,com.whatsapp]
9	Airplane mode	false
10	Connected via Bluetooth	true
11	Network status (off, Wi-Fi, mobile)	mobile
12	No. of available Wi-Fi networks	12
13	Wi-Fi encryption	WPA-PSK-TKIP
14	BSSID of the current connection	d8:c7:c8:44:32:40

2.4 Summary

We summarize the conclusions of this chapter.

Requirements engineering. Requirements define what a potentially new system must be capable of to solve a specific problem or need. Stakeholders, such as customers, developers, and users, define these requirements. They do so in the requirements engineering process that formulates, documents, and systematically maintains requirements. Poorly managed requirements can lead to major risks such as exceeding the budget or failing the project. In requirements engineering, one way of coping with the risks is to include change management. Adapting to change is crucial, as the needs of the users may change or the evolution of the software requires changes. Moreover, user satisfaction decides about the rise and fall of the software in the market. As a consequence, requirements engineering must involve users to identify their changing needs.

User involvement. Traditional approaches, such as workshops and observations, help in understanding features users want and how they use them. However, those approaches are usually part of the early development phases and are challenging to perform continuously with a representative sample of users. Therefore, stakeholders must consider new and modern approaches to involve users.

With the increasing popularity of social media and app distribution platforms, users started to write their opinion about software by reviewing and rating them online (explicit user feedback). Explicit user feedback contains valuable information for requirements engineering, such as encountered problems and requests for new features. Users state their opinion regularly, making explicit user feedback a continuous source for information. Research shows that considering the usage context, in addition to the users' opinion, is vital for user involvement. If the deployed software collects interaction events like clicks and context data, such as the app and operating system version, stakeholders can, e.g., leverage that information to narrow down the location of bugs (implicit user feedback). The combination of explicit and implicit user feedback can lead to either new requirements or requirements of better quality.

Challenges. Including explicit and implicit user feedback in the requirements engineering process comes with challenges. Popular apps receive explicit user feedback thousandfold daily. Similarly, collecting implicit user feedback generates a million interaction events with only a few users. Both feedback types generate a large amount of noisy data that is unfeasible to analyze manually. Therefore, we need to support stakeholders by automatically and continuously collecting and filtering requirements-relevant feedback. Requirements-relevant feedback is, for example, feedback that describes problems, has inquiries to the stakeholders, or includes requests for features.

Analyzing and matching features. Stakeholders need to analyze the filtered user feedback to understand if users address already existing features or request new features. If stakeholders can extract the particular features users mention in their feedback, they can understand the problem and feature requests better. They can also gain insights like what the popular features are and how often users address them to understand their impact on user satisfaction. For example, a feature that occurs in many problem reports may have a higher impact on the app's success than a feature that only one user reported. For an automated understanding, if users discuss existing features, we have to match them with the features in the requirements documentation. Stakeholders document and advertise the features of apps on app pages. App pages contain a description that typically lists and describes the features of the app. As a consequence, we can identify and match the features mentioned in user feedback with the documented features on app pages.

Stakeholder needs. So far, stakeholders do not use automated tools as they are not available in the described sense, although, they seek for automated support. In the next chapter, we present two qualitative studies detailing on stakeholder's needs for analyzing both types of feedback. Then, we introduce requirements intelligence, a framework to automatically collect, filter, and match features from user feedback with the documented features from app pages. The framework provides an integrated interactive visualization to support stakeholders in their decision-making process.

Chapter 3

Stakeholder Needs for Automated User Feedback Analysis

Some people don't like change, but you need to embrace change if the alternative is disaster.

Elon Musk

Publication. This chapter is partly based on our 2016 publication "On the automatic classification of app reviews"[151] and extends its interview study. My contributions to this publication were to co-design the interviews, interviewing stakeholders, creating tool mockups, and to help analyzing and discussing the paper's results.

Contribution. This chapter contributes with two qualitative studies to investigate how stakeholders perceive the usefulness of user feedback and their needs for automated analysis of that feedback. We present a review of related discussing automated user feedback analysis approaches and an interview study, which further aims to evaluate an app review analytics mockup with stakeholders.

3.1 Motivation

In the previous chapter, we concluded that continuously involving users by analyzing explicit and implicit user feedback is crucial for the success of apps. In this chapter, we aim to investigate the usefulness of user feedback from the perspective of stakeholders and how they include it in their processes. **We distin-**

guish between the term stakeholder (any practitioner involved in the decision-making process for the software development) and the term user, who is the end-user of the system.

In Chapter 1, we briefly discussed the need for automated tool support when analyzing explicit and implicit user feedback. More specifically, in Chapter 2, we argued from a data-driven perspective as research highlights the large amount of feedback stakeholders receive. This chapter focuses on the industry perspective and investigates whether stakeholders require automated feedback analyses by performing qualitative research. We reviewed studies on user feedback usefulness and performed interviews to understand stakeholder needs. In total, our studies include the aggregated perspective of 90 stakeholders.

The chapter is structured as follows. First, we describe the research questions and methods followed in Section 3.2. Section 3.4 discusses the results of the interview study. Then, Section 3.3 extends the interviews by summarizing scientific evidence of stakeholder needs from related work. After that, we discuss and compare opportunities and challenges stated by stakeholders in Section 3.5. Eventually, we summarize the findings of this chapter in Section 3.7.

3.2 Study Design

This section describes the overall research methodology. First, we discuss the research questions. Then, we detail the overall study design, including the steps we followed. Eventually, we describe the research methods used.

3.2.1 Research Questions

We aim to understand the usefulness of user feedback from the perspective of stakeholders. We further want to explore if and how stakeholders integrate feedback in their work and summarize the opportunities and challenges they see. Therefore, we formulate the following research questions:

RQ3.1 User feedback usefulness. How do stakeholders perceive user feedback, their usefulness, and how do they include it in their work?

RQ3.2 Use cases for automated user feedback analyses. What are the main use cases for automated user feedback analyses?

RQ3.3 Integration into workflows. What kind of automated user feedback analysis integrations do stakeholders need?

3.2.2 Study Process

Figure 3.1 summarizes the overall study process. It shows that this study contains three main parts (the columns of the figure). The first part reviews related work about feedback usefulness studies. For that part, we first describe the search strategy and the selection criteria for the papers in Section 3.2.3. Then, we present a summary of the results for each selected paper in Section 3.3.

In the second part, we performed semi-structured interviews with 12 stakeholders. We prepared an app review analytics mockup for the interviews that showcases some feedback analysis results. In Section 3.2.4, we introduce the mockup, followed by our interview setting, including a description of the stakeholders in our interview study. In Section 3.4, we discuss the results of the interview study. In the third part, we compare the results of both qualitative studies. There, our focus is on the critical discussion of the opportunities and challenges stakeholders see in Section 3.5.

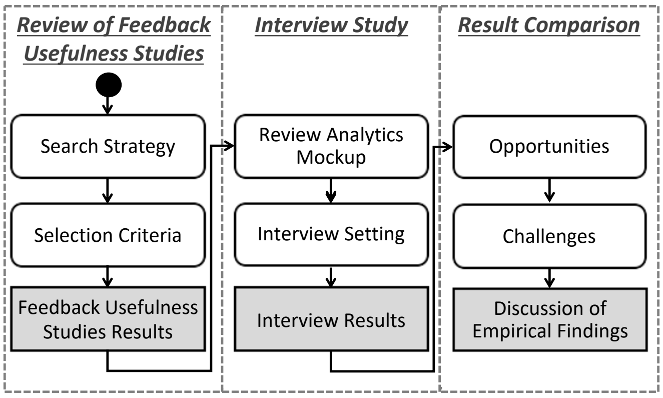


Figure 3.1: Study process.

3.2.3 Review of Feedback Usefulness Studies

We discuss the paper sampling for our related work analysis. In this qualitative analysis, we did not aim at completeness, i.e., finding all papers related to our research questions, but we aimed to cover studies regarding both feedback types.

Search strategy. Our research questions contain the major search terms *user feedback*, *stakeholder*, and *usefulness*. After reviewing the search results and papers previously known to us, we decided to include alternative terms for each major term. Further, as this work focuses on apps and app pages, we also included the term *app* in the search. As a result, we got the following search string:

> *((app OR software) AND (interview OR case study OR workshop OR survey) AND ("user feedback" OR "user review" OR "user involvement" OR feedback OR "app review" OR "explicit feedback" OR "implicit feedback" OR requirement OR feature) AND (automat* OR monitor* OR collect* OR elicit*) AND (stakeholder OR practitioner OR developer)).*

The search string contains the following elements: ((target) AND (stakeholder evaluation type) AND (feedback) AND (approaches) AND (stakeholder). We used the search string on Google Scholar.

Selection criteria. After we received the search results, we filtered out irrelevant studies with the following exclusion criteria.
Exclusion criteria. 1) Non-English studies. 2) Studies that are not in the domain of IT/CS/SE/IS/RE. 3) Non-peer reviewed studies. 4) Studies not following empirical methods.

We included papers that fulfilled the following criteria.
Inclusion criteria. 1) Studies that either report on automated approaches for explicit and implicit user feedback (including tool evaluations). 2) Evaluation with at least two stakeholders. 3) Published between 2013 and 2019.

In total, we discuss seven papers in detail, which cover approaches for explicit and implicit user feedback, as well as tool evaluations. We selected these papers so that we have at least two papers per approach. We also tried stratifying the papers for the selected time frame to include early and recent research approaches. We also limited the study to seven papers to focus on qualitative insights to better understand the stakeholder needs.

We report our results following the same procedure for each paper. First, we give a brief and structured overview of the paper stating 1) its title, target, research questions, and the overall evaluation setting. After that, we summarize the papers by describing the overall objective of the work, explaining in more detail the evaluation setting, and conclude with the results of their evaluation. This template fosters the readability of the results, which we later discuss in Section 3.5.

3.2.4 Interview Study

We interviewed 12 stakeholders using semi-structured interviews. The reason for selecting semi-structured interviews as our method is that we aim to gain in-depth, qualitative insights. This research method allows us to cover a fixed set of questions but also gives us the freedom to follow up on the answers to better understand the stakeholders' reasoning.

Based on our insights from Chapter 2, we created a mockup of a tool for app review analytics. The app review analytics mockup presents scenarios for filtering user feedback automatically. It filters app reviews into problems report and feature requests. We decided to create this mockup to propose a possible solution to stakeholder challenges introduced in Chapter 2. In the interviews, we first asked the stakeholders if they find user feedback useful and about how they include it in their work. Later, we presented the mockup and asked the stakeholders if they find that mockup helpful.

In the following, we present and explain the capabilities of the mockup. Then, we describe the interview setting, including a description of the interviewed stakeholders.

App Review Analytics Mockup

Figures 3.2, 3.3, and 3.4 show the web-based mockup for app review analytics. The objective of the mockup is to show stakeholders a potential solution on automatically filtering explicit user feedback. The mockup is the basis of our discussion in the semi-structured interviews.

All figures of the mockup have the top bar in common. On the top bar, stakeholders can navigate through the mockup and can log in to an existing account. If logged in, stakeholders can see personalized notifications. Personalized notifications are about updates in the analysis but also about "watched users" that update their feedback. If, for example, a stakeholder replies to an app review, the user might update their review in return. That update triggers a notification that allows the stakeholders to react on the update timely.

The landing page of the mockup is shown in Figure 3.2. It contains features such as an import for existing explicit user feedback. Stakeholders can download user feedback from the dedicated platform, such as the app reviews from the Google Play Store, and import them. Alternatively, stakeholders can develop crawlers that retrieve data periodically, in case the platform provides APIs. App stores, for example, usually do not provide these APIs, which makes crawling them cumbersome.

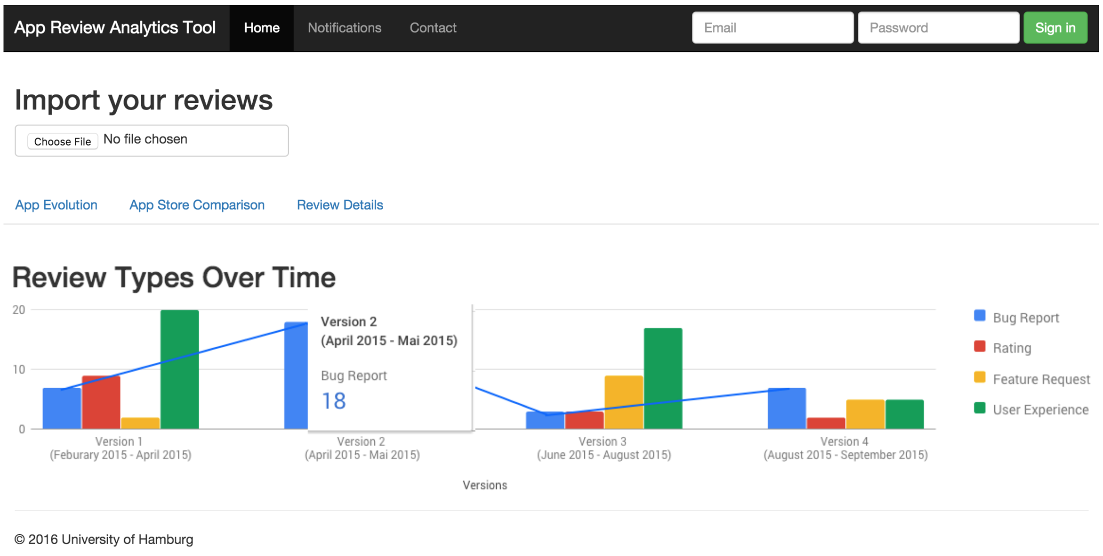


Figure 3.2: Mockup of a review analytics tool: review types over time.

As soon as the stakeholders imported app reviews, they see a trend analysis. Figure 3.2 shows an example for a trend. The mockup prototype creates trends around the four user feedback categories *bug reports*, *rating*, *feature request*, and *user experience*. The trend analysis visualizes the occurrences of the four categories in user feedback for each app release. Therefore, the analysis helps understanding if, for instance, a new release either introduced more bugs or if users report fewer bugs after the release. From that view, project managers can get inspiration for what to focus on next. For instance, if there is a peak of bug reports, the development team may want to switch the focus to fixing bugs before introducing new features. In case there are unusually many feature requests, the stakeholders of the project may want to discuss them and eventually turn them into official requirements.

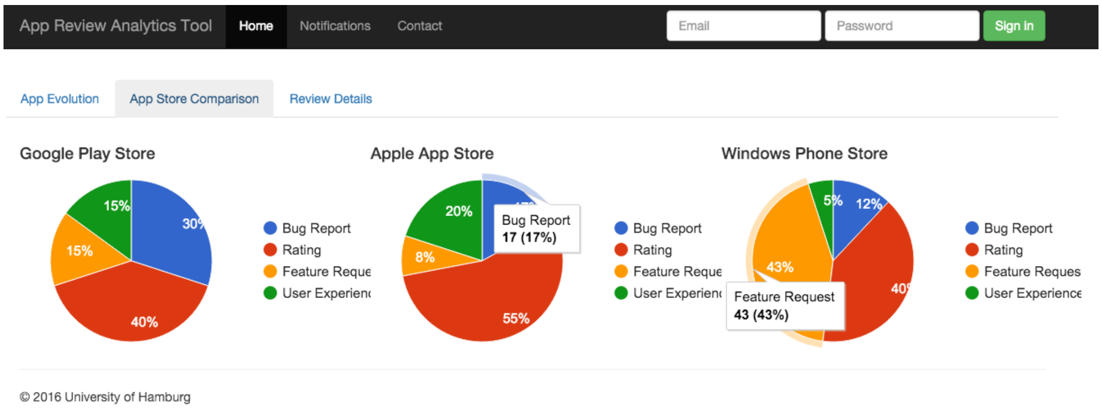
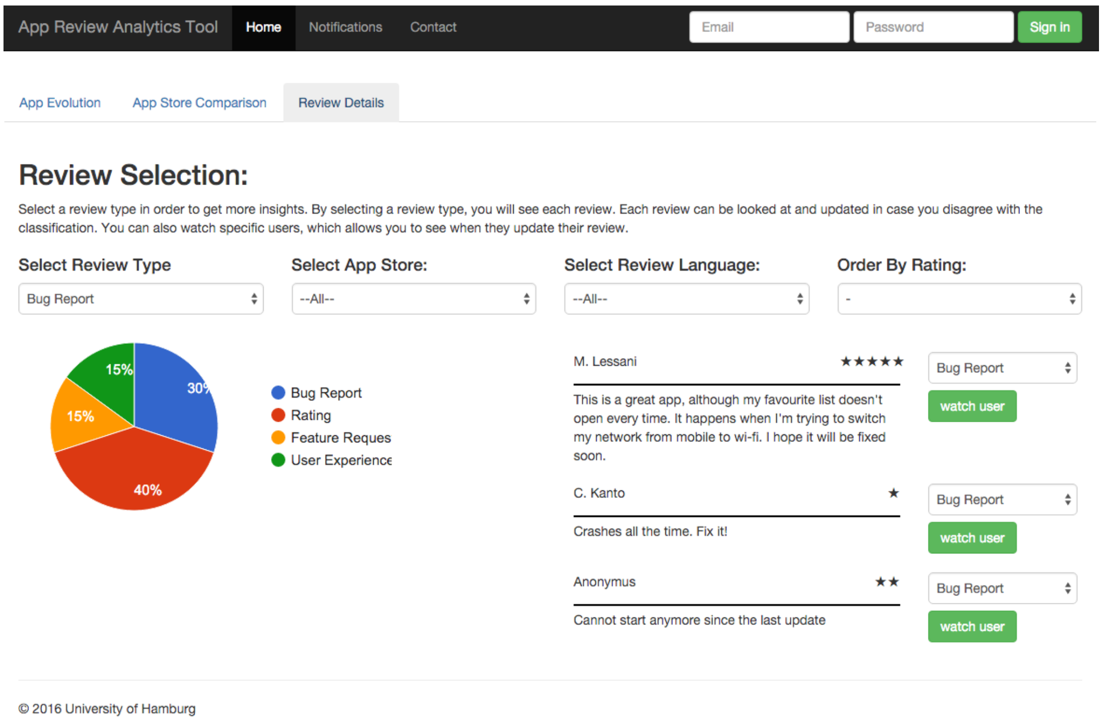


Figure 3.3: Mockup of a review analytics tool: app store comparison.

Software companies of popular apps usually support both major mobile operating systems—Android and iOS. In the next widget, we want to monitor how apps perform per platform because each operating system requires different programming languages, different development teams, and other factors such as operating system fragmentation. We illustrate the widget for this comparison in Figure 3.3. The comparison view shows for each app distribution platform the total number of occurrences per user feedback category, as well as their relative size among each other. In each pie chart (app distribution platform), the stakeholders can see if, e.g., the Android app has more bug reports than other apps. Additionally, the pie chart depicts if any user feedback category currently dominates. Therefore, Figure 3.3 shows a general overview to compare the overall performance of each app to enable individual decision making.

The previously discussed views give a general impression of the app's performance. However, stakeholders need to look at the actual user feedback to allow discussions about, e.g., feature requests or to learn from it for fixing bugs. For that purpose, we provide a third view in the mockup. Figure 3.4 shows the review details tab of the mockup. It contains the aggregated analysis results of the app review filtering. For instance, an approach classifies user feedback into the four categories bug reports, rating, feature request, and user experience. Stakeholders interested in discussing, e.g., feature requests, can use this view to filter all user feedback for this category. We further added additional filters to narrow and focus the results. One additional filter is the app distribution platform, as the



App Review Analytics Tool | Home | Notifications | Contact | Email | Password | Sign in

App Evolution | App Store Comparison | Review Details

Review Selection:

Select a review type in order to get more insights. By selecting a review type, you will see each review. Each review can be looked at and updated in case you disagree with the classification. You can also watch specific users, which allows you to see when they update their review.

Select Review Type	Select App Store:	Select Review Language:	Order By Rating:
Bug Report	--All--	--All--	-

M. Lessani ★★★★★ — Bug Report — watch user

This is a great app, although my favourite list doesn't open every time. It happens when I'm trying to switch my network from mobile to wi-fi. I hope it will be fixed soon.

C. Kanto ★ — Bug Report — watch user

Crashes all the time. Fix it!

Anonymus ★★ — Bug Report — watch user

Cannot start anymore since the last update

© 2016 University of Hamburg

Figure 3.4: Mockup of a review analytics tool: review details.

Android development team most likely wants to discuss bug reports on their app while the iOS development team wants to focus on their platform. Further, the view allows filtering for the user feedback language, which helps organizations that distribute apps in several countries.

We display the filtered user feedback below the filter bar (see Figure 3.4). The left side of the figure shows a pie chart illustrating the overall distribution of the user feedback categories. The chart further allows filtering for the user feedback categories. The right side of the figure shows a list of the user feedback that matches the filter options. This list of user feedback shows the available information like the user name, the star rating the user gave the app, as well as the feedback text body. Next to the list of feedback, the stakeholders can perform two actions. First, there is a dropdown menu that shows the classification result for the user feedback categorization. Therefore, if stakeholders filter for bug reports, all listed feedback will have “bug report” as the default selection in the dropdown menu. In case stakeholders disagree with the classification result, they correct it manually. These manual updates are fed back to the machine learning algorithm to help improve it. The second action is a button that allows stakeholders to watch certain users. If, for example, a user gave feedback, but the stakeholders may need more information, they can contact that user and get notified as soon as the user got back to them. Alternatively, the user may have given a bad star rating because of a frustrating bug. Stakeholders can then reply to, e.g., showing appreciation for the feedback or informing the user that the team is working on it. Finally, if the user updates the feedback, the stakeholders get notified.

Interview Setting

We describe the interview setting in more detail by stating how we performed the interviews and by describing the interviewed stakeholders.

Table 3.1: Overview of the interview participants.

#	Role	Company
P1	Senior app developer	German SME, app development
P2	Senior tester, quality manager	Large European social media company
P3	Lead engineer	European telecommunication company
P4	Project manager for apps	Global market research company
P5	Lead architect, project manager	Mac solutions software development
P6	RE Researcher with practice experience	University
P7	Usability/Requirements engineer	Large software development company
P8	Project manager, requirements analyst	Danish SME, app development
P9	Project manager, requirements analyst	Danish SME, app development
P10	Technology enterprise architect	Italian telecommunication company
P11	Technology innovation manager	Italian telecommunication company
P12	Research and Innovation Senior Manager	Italian telecommunication company

Participants. We interviewed twelve stakeholders. Table 3.1 gives an overview of the interviewed stakeholders. In the stakeholder selection process, we tried increasing diversity concerning roles, company size, and domains. The reason for varying the stakeholders' roles is that the review analytics mockup provides different levels of abstraction regarding the information it displays. As the table reveals, some stakeholders had several roles. Of the twelve stakeholders, six stated that they have management responsibilities. The review detail view of the review analytics mockup contains actual app reviews, which, e.g., developers can use as a direct input for understanding bugs, but also to gather feedback that suggests new ideas for features. Therefore, we not only wanted to rely on the management perspective but also interview stakeholders, such as developers who have experience in app development. Regarding the company size, we aimed at diversity because we want to understand if also smaller companies, which potentially receive less feedback, can benefit from the approach. We cover companies from small and medium-size to enterprises operating in the global market. Among the stakeholders, there is one researcher from a university who previously worked in a small iOS app development company in the role of an app developer. The domains of the companies are also diverse. We cover more general app development companies that work on commissions but also others that run their own product.

Interview details. We performed semi-structured interviews as we needed some questions to be answered directly but allowed open questions as we were also interested in more details such as potential use cases of our approach. Further, we aim at getting in-depth qualitative insights, which we can receive by asking follow-up questions. Therefore, we selected semi-structured interviews as our research method. The interviews with P1 to P9 took place in January 2016, while the interviews with P10 to P12 took place in March 2019. The reason for these two time frames is that in 2016, to the best of our knowledge, there had been no qualitative study, including interviews with stakeholders asking about the usefulness of software requirements intelligence with a focus on app stores. In 2019, we conducted a second interview study to not only extend our existing interviews but also to enrich them by including social media as a data source for the feedback analyses. The interview setting was similar, and each of them lasted about 30-45min. At least two researchers conducted each interview to avoid bias introduced when a single interviewer documents and leads the interview. Therefore, we had dedicated roles in the interview for leading and note-taking. One difference between the two interview iterations is that for the 2019 interviews, we could also make voice recordings, which we later transcribed. Additionally, we performed the 2019 interviews face to face while we conducted the 2016 interviews over the phone.

3.3 Review of Feedback Usefulness Studies Results

We summarize the publications of the feedback usefulness studies from the related work. We discuss seven papers evaluating approaches for analyzing explicit or implicit user feedback or automated tool support for the feedback types. All papers address explicit user feedback, two of them implicit user feedback, and six papers evaluated a tool approach. From a methodology perspective, three papers performed semi-structured interviews, two performed case studies, and two conducted surveys—one of these studies combined a case study and a survey. The paper summaries may state research questions that we extracted or simplified from the paper if no research question was presented or did not fit the format of the summary. Further, we may not list all of the authors' research questions, but those meaningful for this study. We cited all papers in their summary for fostering their quick identification to get more details from them.

Paper	User involvement in software evolution practice: a case study By Pagano and Brügge [186]. Published 2013.
Target	☐ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. How, where, and when do users provide feedback? **RQ2**. How and why do stakeholders work with user feedback? **RQ3**. What are stakeholders' needs for an automated tool support?
Setting	• Case study with semi-structured and open interviews. • Five interviewees (developer). • Interviewees from five different companies (software development). • Recorded and transcribed.

Approach. In 2013, Pagano and Brügge [186] published a case study about stakeholders involving users in the post-deployment scenario. The authors have three high-level goals. First, understanding how, where, and when users provide feedback. Second, exploring stakeholders' motivation for analyzing user feedback, how they do it, and what challenges they face. Third, identifying stakeholder's expectations and needs toward tool support for analyzing user feedback. In their study, Pagano and Brügge focus on developers, software architects, and product managers as stakeholders. For achieving their goals, the authors performed a case study, following the three-phased guide of Runeson and Höst [213] within which they conducted semi-structured and open interviews. They prepared 20 interview questions and seven additional meta-questions about the stakeholders' background. In total, they interviewed five stakeholders from five small to medium-sized companies. The study covers stakeholders working on mobile and desktop apps and have a work experience of three to ten years.

Findings. In total, the authors summarize 17 hypotheses, of which we selected findings related to our research questions. The top three platforms for users to provide feedback are emails, app stores, and integrated feedback mechanisms of the app. Users provide feedback frequently and intentionally chose public channels for critical feedback, but they do not always reach the stakeholders. The stakeholders state that feedback supports the continuous assessment of the app and that it helps to improve the quality of the app. They also say that feedback helps to identify feature requests but that it is hard to understand how many users may benefit from that feature. Positive ratings of the app create trust in other users, and therefore, help to advertise the app in the market. Stakeholders use frequent similar feedback to prioritize their requirements. However, the stakeholders also state challenges for analyzing user feedback, such as low quality and contradictory feedback. Stakeholders read feedback manually and sometimes need to read them several times, making it difficult to understand how many users the feedback affects. They need tool support for collecting and analyzing high amounts of user feedback automatically. Such a tool should categorize feedback (e.g., problem reports vs. feature requests), group similar feedback, count the feedback, and identify the affected features.

Paper	Release planning of mobile apps based on user reviews. By Villarroel et al. [249]. Published 2016.
Target	☐ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. Do stakeholders analyze user reviews for their release planning activities? **RQ2**. Is the categorization of reviews into bug report and suggestion for new feature sufficient for release planning? **RQ3**. Would stakeholders use the developed approach in their release planning?
Setting	• Three semi-structured interviews. • Three interviewees (project managers). • Three software companies (app development). • Neither recorded nor transcribed.

Approach. Villarroel et al. [249] developed CLAP, a prototype to classify, cluster, and prioritize app reviews. In their classification part, they utilize the categories *bug report*, *request for a new feature*, and *others*. After the classification, they applied clustering to find sets of similar app reviews. Eventually, they prioritize app review clusters with their algorithm, which utilizes factors such as the size of the clusters and their average rating. They developed a prototype of that approach, showed it to three stakeholders, and interviewed them to evaluate the approach. For the evaluation, the authors performed three semi-structured interviews with project managers. The interviewed stakeholders come from three different app development companies. Two of the companies develop their own apps, while the third one develops apps on commission. The authors do not state whether they recorded or transcribed the interviews.

Findings. Two of the stakeholders found CLAP very helpful because, so far, they only analyze app reviews manually but know that they contain valuable information. Both agree that their manual approach is time-consuming and that any tool support would be helpful in their processes. One of the stakeholders detailed that for analyzing 1,000 app reviews, a developer of their team spent two full days to analyze them. The second stakeholder argued that, in total, they analyzed 11,000 app reviews manually by dedicating 6-7 hours each week to that task. The stakeholders deem classified and grouped app reviews as helpful for planning their releases. One stakeholder also suggested classifying app reviews into the category "reviews to the app sales plan" in addition to the categories problem report and feature request. In the case of this stakeholder, they provide a free and paid version of their app. The stakeholder motivates the additional feedback category with users who explicitly stated that they would pay for certain features. The third interviewee stated that their company does not consider user feedback at all as they are commission-based app developers.

Paper	SUMMARY: SURF: Summarizer of User Reviews Feedback. By Di Sorbo et al. [59]. Published 2017.
Target	☐ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. How useful are user feedback summaries for stakeholders generated by the suggested approach?
Setting	• Manual evaluation of the developed app review summarization approach based on 2,622 app reviews from 12 apps and three app stores. • Survey about the perceived usefulness of the approach. • Evaluation and Survey with 12 stakeholders (including developers, software engineers and testers, and four academics).

Approach. Based on an approach developed in [58], Di Sorbo et al. present SURF, a summarizer of app reviews [59]. In their work, the authors stress that stakeholders need too much effort to get meaningful information from user feedback. To reduce the effort, they introduce SURF, an approach that generates summarizes of categorized app reviews. These categories, also called user intentions, are either *information giving*, *information seeking*, *feature requests*, *problem reports*, or *others*. Additionally, their approach extracts the topics of the feedback. The authors list twelve topics, such as *GUI*, *improvement*, and *security*, that they can assign to the reviews. SURF outputs an XML file that tools can import but also provides the functionality to visualize them on its own. To evaluate their approach and tool, Di Sorbo et al. surveyed twelve stakeholders. In their evaluation, they report on the study results based on 2,622 app reviews extracted from twelve apps from three different app distribution platforms. For each of the twelve apps, they generated the app review summarizes with SURF and assigned them to the study participants. The stakeholders have diverse roles; three were app developers, three software engineering postdocs, two software testers, three software engineers, and one software engineering master student. The task of the stakeholders was to check if the review summaries were classified correctly. Then, the authors invited them to a survey containing nine questions about the general usefulness of the approach and tool.

Findings. Nine of the stakeholders state that it is challenging to analyze app reviews without having them summarized. The reason is that there can be dozens and more reviews discussing similar issues. Further, the summaries focus on only the important parts of the feedback and foster the understanding of user needs. Three of the stakeholders state that using SURF's user feedback summarization helps them to save between 33% to 50% of their time, whereas eight stakeholders state that they saved more than 50% of their time. Seven stakeholders say that they do not feel to miss out on any information when only seeing the generated summaries. Only one participant states that crucial information is lost when not having the original feedback attached.

Paper	Feedback gathering from an industrial point of view. By Stade et al. [234]. Published 2017.
Target	☐ Implicit Feedback ☑ Explicit Feedback ☐ Tool
Relevant RQs	**RQ1**. How do software companies gather feedback from their end-users? **RQ2**. What is the quantity and quality of feedback that software companies receive?
Setting	• One case study with four stakeholders (CTO, user, helpdesk agent, and manager) from one German company. • The case study was a one and a half days onsite workshop. • Online survey with 18 stakeholders knowledgable in requirements engineering from German-speaking countries.

Approach. Stade et al. [234] conducted a case study and an online survey to better understand the user feedback gathering processes in the industry. The objective of the case study is to get in-depth insights into the experience of companies when gathering user feedback. The authors realized the case study as a one and a half days onsite workshop. The goal of the workshop was to look ad available feedback channels and to highlight aspects they deem important. Four stakeholders participated in the workshop. The stakeholders were one user, the CTO of the company, a helpdesk agent, and a manager. The authors audio-recorded the workshop was audio-recorded and photographed all taken notes. Two researchers later transcribed the audio and wrote short descriptions of demonstrated artifacts. Later, they sent the analysis results to the participating company to let them check if they agree or disagree with the results. The survey included German-speaking countries and targeted quantifiable results to validate the insights of the case study. The authors got complete answers from stakeholders of 18 companies that cover different sizes ranging from less than ten employees to companies with up to 100,000 employees.

Findings. The helpdesk agent of the workshop analyzes user feedback manually to prepare a report for their monthly meeting. However, that process is time-consuming, and the helpdesk agent desires a tool that creates alerts for urgent user feedback. All 18 companies use explicit feedback coming from hotlines and emails. Seventeen companies gather feedback directly at the customer site, while 15 companies gather feedback via contact forms. Eleven of the 18 companies gather feedback from ticket systems and forums. Ten surveyed companies gather user feedback from social media, whereas five use app stores as a feedback channel. Generally speaking, the surveyed companies provide at least three and up to 13 feedback channels to their users. While the company of the case study stated that user feedback they receive is of high quality, the survey participants were more diverse in their answers. Five companies are satisfied with the quality of user feedback. However, nine companies neither agree or disagree if they are satisfied with the quality of user feedback; no company strongly disagrees. The majority of the surveyed companies agreed that the received user feedback is relevant for software evolution but also stated that often, user feedback lacks information to understand it fully. Therefore, Stade et al. also shows the importance of analyzing user feedback and shows that tool support is appreciated even in companies that do receive less than 200 user feedback per month.

Paper	FAME: supporting continuous requirements elicitation by combining user feedback and monitoring. By Oriol et al. [184]. Published 2018.
Target	☑ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. Can the combination of explicit and implicit user feedback support stakeholders to elicit new requirements?
Setting	• Case study with a German software development company performing a two-phase workshop. • The workshop involved one software developer and one researcher.

Approach. Oriol et al. [184] argue that for successful continuous requirements elicitation, it is important to perform and combine both explicit and implicit user feedback. They base their arguments on existing literature [29, 153, 253]. They developed an approach called FAME in cooperation with a German SME company within the European Horizon 2020 project SUPERSEDE (same project as in the study of [234]). The focus of the approach is to combine explicit and implicit user feedback to foster the elicitation of new requirements. More concrete, the authors state the following research objective: "To provide a unified framework capable of gathering and storing both feedback and monitoring data, as well as combining them using an ontology, to support the continuous requirements elicitation process" [184]. Instead of relying on the feedback coming from social media or app stores, they developed a custom feedback mechanism, which they included on the website of their industry collaborator SEnerCon. Similar to app store reviews, users can use their feedback mechanism to give a star rating and to write feedback. Moreover, users can add screenshots, audio recordings, and select a feedback category such as bug reports. The authors collected feedback over four months. They then analyzed the gathered feedback in a two-phase workshop with two stakeholders—one researcher and one software developer. In the first phase, the developer had to elicit requirements from only explicit user feedback. In the second phase, the developer could elicit new requirements or refine the previously created, also using implicit feedback. About 5,000 users logged in during the feedback collection time frame.

Findings. From the logged in users, 24 created 31 explicit user feedbacks entries. The implicit feedback component recorded about one million clicks and 160,000 navigation actions. From the 31 user feedbacks, the stakeholder identified 16 as relevant, which eventually led to nine new requirements. As the study focuses on the identification of new requirements, they considered problem reports as irrelevant. The remaining 15 out of 31 explicit user feedback entries were either problem reports or issues related to customer service. In three cases, a single user feedback entry led to two requirements. By using the combination of explicit and implicit feedback, the stakeholder found one additional requirement and refined four previously elicited requirements. However, as analyzing about one million implicit feedback entries is too much to perform, the authors limited the number of analyzed clicks to 2,164. Therefore, similar to explicit user feedback, implicit user feedback comes in huge amounts unfeasible to analyze manually. However, if filtered purposefully, the combination of both feedback types can either lead to new requirements of help improving requirements extracted from only one feedback type.

Paper	Generating Requirements Out of Thin Air: Towards Automated Feature Identification for New Apps. By Iqbal, Seyff, and Mendez [115]. Published 2019.
Target	☐ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. What are contemporary practices and challenges requirements elicitation for developing new apps? **RQ2**. Do practitioners already analyze crowd-generated data or information provided by the crowd e.g. app store data and if so, how?
Setting	• Eleven semi-structured interviews. • Eleven interviewees (e.g., software engineer and project manager). • Interviewees from eleven companies (app development). • Recorded and transcribed.

Approach. In their 2019 published work, Iqbal et al. [115] focus on requirements elicitation for new mobile apps rather than the evolution of existing apps. For that, they performed eleven semi-structured interviews, including steps in the interviewee participation selection, to foster diversity. Their inclusion criterium was that the stakeholders have an overview of the requirements engineering process of their company and experience in mobile app development. Further, the stakeholders must be either requirements engineers, business architects, project managers, consultants, or software engineers. On average, the stakeholders have about six years of work experience.

Findings. Traditional requirements elicitation approaches, such as interviews and workshops, are the most common among the stakeholders. However, eight of the eleven stakeholders state that they use user feedback from the app stores, while three say that they also consider user feedback from social media. The remaining five stakeholders state that they do not analyze feedback from social media as it takes too much time and effort to get information from that feedback channel. While eight out of eleven stakeholders state that user feedback is the primary source to understand users better, three explicitly state that user feedback is not reliable as there is a lot of fake and auto-generated feedback. Eight stakeholders say that user feedback is the main source for understanding users of particular app features. The stakeholders have an interest in negative feedback because they help them in identifying gaps in the market. Negative feedback also helps in the feature selection process and to improve their own apps. The stakeholders are not aware of an automated tool for extracting information from app reviews such as feature requests and agree that they need an automated tool. That tool should analyze app stores to suggest a set of features for new apps.

Paper	How do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering? By Johanssen et al. [119]. Published 2019.
Target	☑ Implicit Feedback ☑ Explicit Feedback ☑ Tool
Relevant RQs	**RQ1**. Which user feedback do practitioners consider? **RQ2**. How do practitioners capture user feedback? **RQ3**. How do practitioners utilize user?
Setting	• 20 semi-structured interviews. • 24 interviewees (e.g., developer and project manager). • Interviewees from 17 companies (development and consultancy). • Recorded and transcribed.

Approach. In their paper, “How do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering?” [119], Johannsen et al. conducted a total of 20 semi-structured interviews with 24 stakeholders from 17 different companies in 2017. Their goal was to understand how the industry captures and utilizes user feedback. For that, they formulate three main research questions that cover 1) what kind of user feedback stakeholders consider (explicit/implicit), 2) how (and how often) they capture user feedback, and 3) how they use user feedback. The authors focussed on diversity when selecting the interviewees by varying the size of the companies (small to enterprise), stakeholder roles, as well as the project domains. The authors recorded and transcribed the interviews in their analysis process.

Findings. Their results show that all stakeholders consider explicit user feedback. Twelve of their stakeholders consider only explicit user feedback, and eight consider both implicit and explicit user feedback. No stakeholder solely considers implicit user feedback. Thirteen stakeholders use tools support for capturing user feedback. In such cases, the stakeholders rely on standard software provided by major companies such as Google, Microsoft, and Adobe like Redmine and JIRA. Five stakeholders exclusively perform manual analyses, such as reading emails, performing workshops, or interviewing users. However, the tools analyzing explicit user feedback do not cover the automated aggregation and analysis of user feedback. Five stakeholders developed custom tools for the capturing process of user feedback. The reason for capturing user feedback is different for the interviewed stakeholders. Again, five stakeholders use user feedback for multiple purposes, while four exclusively use it in the planning phase, two exclusively use user feedback for support, and one stated that they use it exclusively for improvements. It is important to note that in that paper, the definition of explicit feedback also covers verbal interactions such as workshops with users, as well as company internal communication.

3.4 Interview Results

We summarize the results of our interviews, as described in Section 3.2.4.

Stakeholders value user feedback but rarely use it in their decision-making process. Nine stakeholders agree that explicit user feedback is generally useful, but only three of them look at it regularly. One challenge is that user feedback comes from multiple channels like the app stores, email, the customer care department, or as well as from internal employees. The stakeholders do not have a single place to collect and analyze user feedback, although four of them state that it impacts the development process—but only if a certain amount of users reported the same issue or feature request. For instance, if only one user reports an issue that happens on a particular and rarely used device within the customer base, stakeholders are more likely to ignore or delay that issue in their planning. P2 looks into the received user feedback daily but estimates the majority of the received feedback as rather uninformative and, therefore, not helpful.

Most of the time, the stakeholders try reproducing the bugs reported but have a difficult time as essential information such as the steps to reproduce are missing. Often feedback only contains praise (e.g., "Great app!") or insults (e.g., "I hate the app so much."), which is tedious to filter, especially considering the cases when the participants receive much feedback. The stakeholders in our study do not have tool support for extracting information from user feedback but do this manually by looking at individual reviews, by inviting test groups, or by directly asking users. P10 to P12 state that user feedback is very useful but not yet part of the company's decision-making process because decisions about the product roadmap, so far, were decided top-down. Nonetheless, the company acknowledged the usefulness of user feedback recently and tried to integrate it into their processes as an enabler for innovation.

Analyzing the received feedback is not enough; stakeholders also consider feedback users send to their competitors. The general use case all stakeholders state is to better understand their users. More specifically, eight stated that they would like to analyze user feedback for identifying problems with the software and services provided. Further, eight participants stress that user feedback can be helpful to identify new features to foster innovation, which may help to either stay competitive or to get an edge over the competitors in the market. To find innovation, the stakeholders suggested to not only analyze the feedback of their own product but also look into the feedback of competitors. P3, for instance, stated that for their company, it is uttermost important to react to user feedback because otherwise, the image of the company may be damaged. Two stakeholders explicitly state that one use case should be the aggregation of user feedback from several channels to minimize places to look for and analyze user feedback. Therefore, most of the stakeholders wish automated tool support for extracting information from user feedback collected from several channels and their competitors.

Stakeholders need to combine and filter implicit and explicit user feedback and to quantify analyses results. P1 argues that only analyzing explicit user feedback is not enough and emphasizes that for fixing bugs, it is important to also analyze and attach implicit user feedback to the explicit user feedback. That way, context information such as steps to reproduce, as well as the used hardware and software version affected, can accelerate the bug fixing process. The stakeholders P2, P3, P6, and P7 state that a tool should filter non-informative feedback. Furthermore, P2, P4, P5, and P6 suggest to quantify and group feature requests and bugs to better prioritize them in the development phase. Five stakeholders find it helpful to see trends in user feedback to better understand the overall user satisfaction with the app to answer questions like "were we able to receive fewer problem reports?".

Stakeholders found our mockups review analytics tool helpful but miss key features like context data from implicit feedback and the aggregation of similar explicit feedback. All stakeholders agree that the app review analytics mockup is helpful and improves the current state of the art. Generally, they describe the mockup as clearly designed and well packed with features. Five stakeholders agree that the information in the trend widgets is meaningful for them. Four stakeholders highlight the presence of the single reviews, as this helped them to understand what users talk about in detail. Also, four stakeholders found the store comparison helpful because their company owns apps in multiple stores, and so far, they do not systematically compare their performance. However, the stakeholders made the following suggestions for improvement.

- Show numbers that indicate how much of the feedback is relevant and how mus is irrelevant.
- Display irrelevant feedback to allow checking whether feedback was miscategorized.
- Show the date of the feedback.
- Add a summarization highlighting the app features users address.
- Group/cluster similar user feedback.
- Add context information such as the hardware and software version.
- Attach implicit user feedback to explicit user feedback.
- Add a sentiment analysis to understand the overall mood/opinion of the users.

- Include competitors in the analysis to allow a comparison regarding the performance.

P1 and P2 highlight that attaching implicit user feedback would be the most important feature because that data would help in the bug fixing process. With the implicit user feedback, developers could understand the steps that led to the issue, and also, if it had been a non-crashing bug, see how the user and the app behaved afterwards. A dashboard should summarize and show the most important information to highlight major issues.

The role of the stakeholders influences the type of information they need. We found some relations between the role in the company and the most liked views of the prototype. Project managers found the trend analysis and the store comparison most helpful (see Figure 3.2 and Figure 3.3). The reason is that they usually do not have the time to check every single user feedback but instead need a quick overview of the current state of the product to make decisions. One stakeholder detailed that they would use such a tool with a dashboard on a monitor in the office so that the development team can get an impression of the current status every morning. Developers and testers found the actual user feedback most helpful (Figure 3.4 because rather than knowing that there are problems in the app, they want to know more details. In particular, P1 and P2 are interested in getting a description of the problem.

Analytics integration scenarios depend on the company's infrastructure. The answers on how to integrate feedback analytics vary and depend on the infrastructure used in the companies. However, we could identify three groups of suggestions for integration. The first group (six stakeholders) would use the tool as a standalone product. The standalone tool should come as a web-based solution that they can access anywhere, including mobile devices. A web-based solution enables the stakeholders to stay informed at any given time and place. Four stakeholders suggested adding the results of this tool into elaborated issue tracker systems such as JIRA. The most comfortable integration would be to auto-generate new issues for the issue trackers so that stakeholders could immediately discuss or even work on them. However, if the tool automatically creates the issues, some stakeholders argue that there should also be a human who edits them before they are visible to the team. Two stakeholders wish to integrate the tool into crash tracker systems such as *crashlytics* [89], which reports the stack trace of occurred crashes. Consequently, being able to observe crashing and non-crashing bugs would provide a complete view of current problems in one place. Five stakeholders additionally stated that export functionality is essential to back up the analysis results and to use the analysis results in other tools.

3.5 Discussion of Empirical Findings

Here we discuss the outcome of the literature review in Section 3.3 and our interview study from Section 3.4. Our discussion relies on a total involvement of 90 stakeholders in either interviews, surveys, case studies, or workshops. We discuss the findings by presenting the opportunities and challenges stakeholders see for analyzing explicit and implicit user feedback, as well as for automated tool support. We highlight the key findings with a bold font.

Explicit user feedback

The seven studies from the literature review, as well as our conducted interviews, address explicit user feedback. Research in that area is very active, as Martin et al. [162] show in their survey of app store analyses. In our interview study as well as in the seven research papers, all authors created approaches to analyze explicit user feedback. However, the actual use cases of the approaches vary. Some papers focus on categorizing app reviews into requirements related information like bug reports and feature requests, others focus on helping to prioritize issues, and some had the goal to aggregate information from categorized user feedback.

Stakeholders highlight that user feedback contains a lot of noise, resulting in a high effort for a manual analysis, which does not help to improve the software product. In all studies, the involved stakeholders agree that the analysis of explicit user feedback is needed. In the study of Villarroel [249], one stakeholder state that he spent two full workdays to analyze 1,000 app reviews, while a second argued that he analyzed 11,000 app reviews manually by dedicating a weekly 6-7 hour slot for that purpose only. However, stakeholders know that there is feedback, which is valuable for prioritization tasks.

Stakeholders see the opportunity to utilize automated approaches to reduce manual effort, remove irrelevant feedback, and include multiple feedback channels in a single place. A second challenge in analyzing user feedback is that it comes in different forms from different platforms. Depending on the company, they may have more than three feedback channels, including traditional channels like phone and email, or modern feedback channels such as Twitter and the app distribution platforms. This situation makes it cumbersome to aggregate the information as there is no single place in which stakeholders can collect and visualize feedback. Although an automated analysis of explicit user feedback brings opportunities, stakeholders also see challenges with the suggested approaches.

In particular, we have to convince stakeholders that the results of the automated analysis are truthful and, to some extent, complete. As automated approaches are not perfect, they may miss valuable feedback that a manual analysis would discover. When considering the SURF approach by Di Sorbo et al. [59], which summarizes similar user feedback, most stakeholders state that they do not feel to miss out on information. Nevertheless, one stakeholder states that these summaries miss crucial information, which suggests that the possibility of seeing the original user feedback might be a good step towards increasing trust in the developed approaches. Our interview study supports that finding as stakeholders state that they want to see all classification results and want a mechanism to correct them.

Stakeholders want to know the features user discuss and the features similar apps provide. Our interview study shows that stakeholders want to identify new features to foster innovation. In a tool, they want a summarization highlighting the features users address. Two publications from our related work review agree with this finding. They further say that stakeholders want to understand the users of particular features and that app store analyses shall suggest a set of features from similar apps for either improving an existing app or for developing a new one.

Implicit user feedback

There are only a few approaches that analyze and evaluate implicit user feedback for requirements engineering with stakeholders. In this chapter, we introduced two papers that evaluated approaches for implicit feedback. Additionally, in our interview study, two stakeholders state that they wish to have access to that kind of data.

Stakeholders suggest adding implicit user feedback to explicit user feedback to help developers understanding, e.g., what hardware and software versions are affected and what interactions in what context led to the issue reported. Our interview study, which initially targeted explicit user feedback, reveals the need for implicit user feedback. Though most stakeholders state that explicit user feedback is useful, some stakeholders declare the need to have context and usage data attached to the written feedback. In particular, problem reports rarely include technical information such as context data, which in previous studies also show [157, 187]. However, without context data, bugs are challenging to reproduce. The study of Johanssen et al. [119] further highlights the need for combining both feedback types as eight stakeholders stressed this fact. Therefore, they agree with the stakeholders from our interview study but further, add more use cases such as monitoring implicit user feedback to evaluate A/B testing and to collect general usage statistics. Another opportunity stakeholders state is that this kind of user feedback can help to improve specific parts of their software. For example, monitoring interaction data can help to learn how users use the app features, in particular, to identify the most frequently used features and the feature users address most often in problem reports. Further, stakeholders can monitor interaction data to identify unintended navigation flows and optimize them accordingly. Finally, the workshop of Oriol et al. [184] shows that stakeholders can use implicit feedback to elicit either new requirements or to refine existing ones.

As already a few users generate a million of implicit user feedback, stakeholders need an automated aggregation and filtering of the feedback before they can analyze it. In the analyzed studies, no stakeholder is stating that they solely rely on implicit user feedback. Usually, stakeholders combine it with explicit user feedback as they require an explanation from the user perspective of, e.g., what happened during a non-crashing bug. Oriol et al. [184] highlight that one challenge of collecting and analyzing implicit user feedback is the amount of data generated. The amount of generated data highly depends on the software, what stakeholders monitor, and how many users use the tool over what time frame. Nonetheless, their study shows that even 5,000 thousand logged in users generated about a million click events in four months. Though these numbers do not say much without knowing how long the users stayed on the website, they already suggest that getting meaningful information from implicit user feedback is only feasible if stakeholders can aggregate it.

Automated tool support

Five studies from the literature and our interview study included the evaluation of a tool with stakeholders. As the developed approaches differ, the tools differ, too. In the following, we compare the feedback from the stakeholders.

Most of the 90 studied stakeholders do not have dedicated tool support for feedback analytics yet, though most desire such. The majority of them state that "any" tool support would already be helpful. There are tools that stakeholders use, but they focus on collecting user feedback or documenting requirements in, e.g., issue trackers. Automatically analyzing user feedback is already a good step toward improving the current state in the industry, but often scripts and raw data are not sufficient to effectively analyze user feedback. Stakeholders, therefore, need automated approaches that visualize the results. Another opportunity is that with a web-based tool, stakeholders can stay informed about their product at any given time and place, which gives them the feeling of being informed and in control.

Stakeholders see the opportunity in tools to have a quick overview of the current situation in the user feedback, such as seeing how many bug users reported since the last release. This opportunity is particularly important for project managers, who are more interested in seeing the overall performance of the app instead of detailed feedback. These stakeholders further highlight that they need to feel in control and informed all the time. Therefore, they suggest having a dashboard available on the web, which they can access from different devices.

To gain trust in the results of the developed approaches, stakeholders also want to have the possibility to see the original, unprocessed user feedback. Another advantage of exploring the original user feedback, especially for developers, is to get a complete overview of the particular issues or requests of their users. However, as the explicit and implicit user feedback challenges suggest, user feedback is often noisy and uninformative. Therefore, stakeholders see the opportunity to have a reduced effort in analyzing user feedback if the tool is capable of hiding irrelevant user feedback. In Di Sorbos et al. [59], eight stakeholders state that using their tool to summarize user feedback helps them to save more than 50% of their time.

Tools have to give stakeholders control over the analysis results. Generally speaking, stakeholders see any tool support as an improvement over the current situation. Despite usability aspects specific to the tools developed in the summarized studies, most challenges lie in the underlying analysis of user feedback, which the tools need to mitigate. For instance, the developed approaches are not perfect, and as such, do make mistakes. The tool should have the capability to handle these mistakes by, e.g., giving the stakeholders the chance to correct machine learning-based results. Integrating human feedback in the tool increases trust and allows the underlying algorithms to improve over time.

Another challenge is the diversity of the infrastructure landscape as each company has different tools for, e.g., tracking issues, and documenting requirements. Therefore, stakeholders see diverse ways of integrating requirements intelligence into their current workflows. While some wish for a standalone tool, others would like to see it integrated into their existing tools. The conclusion is there is not a single solution to the stakeholder's needs, and therefore, such a tool looks differently, use different approaches, and is integrated differently in every company that may use it.

3.6 Limitations and Threats to Validity

We discuss the internal and external threats to validity.

Internal threats to validity. Regarding the internal validity, we might have missed some insights from the interviews or wrongly interpreted the answers as we could record only three of the interviews. We mitigated that threat by conducting the interviews with two researchers. One of the research was responsible for leading the interview, while the other was responsible for note-taking. Further, we performed semi-structured interviews that allow follow-up questions for more details in case the answer was unclear.

Regarding the review of related work, the authors of the papers, and we might have misinterpreted the answers of the stakeholders. We only included peer-reviewed empirical studies that performed measures to ensure the evaluation quality, such as recorded interviews, the triangulation of data sources, and the number of researchers that analyzed the answers. Additionally, to mitigate the threat of only covering insights that are due to popular research topics of specific years, we included papers ranging from 2013 to 2019.

External threats to validity. We cannot claim that our results generalize because our interview study included only 12 stakeholders and because we did not perform a systematic literature review for our second study. However, we tried mitigating the threat in our interview study by increasing diversity in the stakeholders' roles, company sizes, and the location of the companies. We cover eight unique stakeholder roles, nine different small to enterprise companies in diverse European countries. Still, as we have for some roles, only one stakeholder, the generalizability is limited. However, after 12 interviews, we did not gain additional insights. Further, our interviews and the review study share many of the findings. Regarding the review of related work, we tried mitigating the threat to external validity by covering the evaluation with 78 stakeholders, which have diverse roles in the requirements engineering and software development processes. We are confident that the evaluation of 90 stakeholders led to meaningful insights.

3.7 Summary

We explored stakeholders' perceived usefulness of user feedback, use cases for automated feedback analysis, and possible integrations into their workflow. We applied qualitative research by reviewing seven papers from related work and by conducting an interview study with twelve stakeholders. In total, our qualitative analysis encompasses the evaluation of 90 stakeholders from the industry. We summarize the findings of this chapter.

User feedback usefulness. Stakeholders value user feedback but rarely use it in their decision-making process. They argue that both explicit and implicit user feedback comes in amounts that are unfeasible to analyze manually. They are not aware of tools that analyze user feedback automatically and, therefore, rely on the manual analysis. One stakeholder said that for analyzing 1,000 app reviews, their team spent two full days, but as research shows, popular apps receive about 4,000 app reviews and about 32,000 tweets daily. Still, many of the stakeholders in our qualitative research state that they try analyzing the feedback manually. They realized that user feedback could lead to quicker bug fixes, increase the requirements quality, and can contain information for finding innovative features. Stakeholders do not consider their feedback useful but also the feedback their competitors receive to identify gaps in the market.

So far, some stakeholders have tools to collect user feedback, but most do not have any tool support that helps them to get meaningful information such as ideas for new requirements. Depending on the stakeholder's role, they either seek for more general data representation about the overall app performance or more detailed summaries hinting toward the actual pain points and requests.

Use cases for automated user feedback analyses. Stakeholders stated that feedback is most useful if automated approaches filter it so that only relevant feedback is visible. Additionally, if irrelevant feedback is filtered and only feedback describing, e.g., problem reports and feature requests, is available, the amount of feedback is sometimes still too high. Therefore, they wish for further aggregations like grouping similar feedback.

Typically, the majority of user feedback comes from non-technical users. Therefore, if stakeholders solely analyze explicit user feedback, they do not have access to interaction and context data, which are essential to understand described problems. Alternatively, if stakeholders only have access to implicit feedback, they can understand “what” users did but now “why”. They do not know the reasoning of the user. Our study shows that combining explicit and implicit user feedback allows for generating new and improving existing requirements. In the ideal case, stakeholders have filtered explicit feedback available that describes, e.g., a problem and have access to the interaction and context data that led to the problem. Therefore, the combination of both feedback types can lead to quicker bug fixes.

Nonetheless, some stakeholders stated that they would not necessarily trust automated approaches. For example, a stakeholder stated that they would like to see the filtered feedback, too, to understand if they miss important information. They want to correct wrongly categorized feedback and include it in the analysis. Therefore, it is essential to give stakeholders a control mechanism in the tool to correct and improve the underlying algorithms.

Integration into workflows. The available IT infrastructure is different or even unique for most companies. Therefore, there is no single integration scenario that fits the workflow of every stakeholder. Most stakeholders in our interview study wish for a web-based standalone tool that they can access from anywhere at any given time. Such a tool helps stakeholders to feel in control and to feel informed about their app’s performance in the market.

Chapter 4

Requirements Intelligence

It always seems impossible until it's done.

Nelson Mandela

Contribution. This chapter contributes with the introduction of requirements intelligence, a framework based on stakeholder needs. The framework leverages the analysis of explicit and implicit user feedback. We detail the analysis activities and the integrated interactive visualization, which visualizes the analysis results.

4.1 Motivation

In the previous chapter, we detailed on stakeholders' needs and expectations regarding automated analyses of explicit and implicit user feedback. We found that they need automated tool support to collect and to filter the vast amounts of user feedback they receive. The stakeholders further highlighted the need to identify the specific features (functional requirements) that their and their competitors' users address. The stakeholders want to get an overview of their app's performance and detailed insights into the user feedback and the addressed features in a web-based tool.

Following the insights of Chapter 2 and the results of Chapter 3, we propose *requirements intelligence*, a framework to continuously collect, preprocess, filter, transform feedback to requirements, and to interactively visualize explicit and implicit user feedback. First, we define the term requirements intelligence in Section 4.2. Then, we present the requirements intelligence framework and its activities in Section 4.3. After that, we introduce our machine learning pipeline that we employ within the activities of the framework in Section 4.4. Section 4.5 summarizes the chapter.

4.2 Definition

The overall goal of *Requirements Intelligence* is to leverage user feedback to inform project managers, developers, and any stakeholder by providing analytical insights into software features (functional requirements). We see user feedback as a continuous source for information that stakeholders can use to either get inspiration for new requirements, improve existing requirements, or to get analytics insights into the app's features and how users use them. Requirements intelligence mainly targets software products, such as mobile apps, desktop, and web applications. Although software apps are the primary target, the ideas and approaches of requirements intelligence, may also apply to other domains utilizing instrumental products and services.

In the following, we establish our definition of *requirements intelligence* by detailing both terms requirements and intelligence. For requirements, we look at requirements engineering and its activities we affect. For intelligence, we look into the closely related fields of business intelligence & analytics and software analytics.

4.2.1 Requirements

The term "requirements" determines that the intended use of the framework is within the area of requirements engineering. We understand requirements engineering as a continuous process in software engineering that takes part *before* and *during* the development of a software product, as well as *after* its deployment, and that is capable of adapting to change (see Section 2.1.1).

We discussed that change management in requirements engineering is crucial because user needs may change over time. Projects that do not adapt their requirements may exceed their budget or eventually fail. User satisfaction decides about the rise and fall of software in the market. Stakeholders can cope with the risk of unsatisfied users by continuously involving them in their requirements engineering activities. Requirements engineering is not a single-phased process but cyclic and iterative. Traditionally, requirements engineering involved users primarily in the early phases of development to elicit requirements by performing, e.g., physical workshops and interviews. However, these approaches are challenging to perform continuously and cannot easily cover a representative sample of the users. With the increasing popularity of social media and app distribution platforms, users started to state their opinion online. Online platforms are a rich source for requirements-related and continuous explicit user feedback. Also, software allows monitoring user interactions and the context of use, and therefore, provide valuable implicit user feedback. Stakeholders know that explicit and implicit user feedback is valuable but rarely use it in their work due to high amounts of noisy and irrelevant feedback.

Requirements. Requirements intelligence aims to make valuable user feedback available automatically. As that kind of user feedback is a continuous source for post-deployment information, it affects several requirements-related activities. In Section 2.1.1, we introduced typical requirements engineering activities and showed how agile requirements engineering interprets them. We focus on agile requirements engineering as we found it as the closest interpretation of requirements engineering capable of continuously involving users.

In the *requirements elicitation* activity, requirements evolve and are discovered in the whole development process. Requirements intelligence considers post-deployment feedback and can, therefore, support the continuous discovery of new requirements. Despite the suggestion of Ramesh et al. [202], requirements intelligence does not consider face-to-face communication but enables input from explicit and implicit user feedback.

Our framework supports the *requirements analysis and negotiation* activity by continuously providing input for refining and changing requirements by analyzing the features documented on app pages and matching them with the features addressed in user feedback. By quantifying the analyses results like knowing how many users describe a problem with a feature, stakeholders can use that insight for their prioritization process.

Requirements intelligence does not support the *requirements documentation* activity in the traditional sense but provides an integrated interactive visualization about the users' opinions and the features they address.

Requirements validation in agile requirements engineering asserts whether the requirements meet the current user needs. Requirements intelligence allows filtering user feedback for problem reports and inquiries (which contains feature requests and questions towards stakeholders). Analyzing these two categories helps stakeholders to understand the current needs of their users.

4.2.2 Intelligence

Intelligence describes the output of the framework. The definition of intelligence is under constant change and debate, and yet, no standard definition exists [138]. Since we target a technical landscape, we do not follow the definition of human intelligence but business intelligence & analytics and software analytics, which we consider closely related fields.

Business intelligence & analytics is the most recent evolution of decision support systems [38]. In the early 1960s, the industry started to transfer some of its processes to computer-based systems to have support for tasks such as billing, inventory records, or processing orders [136]. Decision Support Systems (DSS) were officially introduced by Gorry and Scott Morton [94] in the 1970s. These systems were improved over the years to support the organizational management of a company and single managers to get answers to predefined procedures [11, 225]. However, top management often needs to make nonroutine decisions that do not rely on predefined procedures. Tools supported managers with data coming from different sources [136, p. 78]. A major advancement is the introduction of multidimensional modeling, allowing its users to view data from different perspectives [11, 110].

Dedić and Stanier [52] define Business Intelligence as follows:

> [...] includes the strategies, processes, applications, data, products, technologies and technical architectures used to support the collection, analysis, presentation and dissemination of business information.

In the year 2000, Analytics became a key component of Business Intelligence & Analytics [50]. Chen, Chian, and Storey [38] state that in the recent understanding of this term, big data and big data analytics play an essential role as business data may come in terabytes or larger, and because modern systems collect complex data from sensors and social media.

Types of analytics. Analytics is the analysis, the use of data, and systematic reasoning to make decisions [49]. Analytics can deliver data-driven answers to events that happened in the past, are happening in the present, and that may happen in the future [34, 49]. We can categorize analytics into four types.

Descriptive analytics explains what happened and what is currently happening through existing data. It can summarize big amounts of data using statistical measures like averages and percentages. Dashboards like Google Analytics are an example that shows past and real-time descriptive statistics for stakeholders. In requirements intelligence, descriptive statistics are summaries like heatmaps of when users provide feedback and how much feedback they submitted over which channels.

Diagnostic analytics aims at identifying why something happened. For example, descriptive analytics identified that an app received unusual amounts of problem reports. With diagnostic analytics, we try to understand what the exact problem is and why it is a problem. In requirements intelligence, diagnostic analytics empowers explicit user feedback to, for example, understand what the problems are by identifying similar problems and relating the problem to specific features.

Predictive analytics tries to answer what may happen in the future. For answering forecasts, we need advanced techniques like machine learning algorithms that can make predictions based on a model trained on historical data. In requirements intelligence, we can, for example, use implicit user feedback to predict the context of the user or to predict the features users use.

Prescriptive analytics provides directions for stakeholders to make decisions. It aims at helping stakeholders to identify what the best next action is. Navigation apps that consider real-time traffic to find the best routes are an example of predictive analytics. In requirements intelligence, we do not focus on this type of analytics. However, stakeholders can use the analyses results of explicit and implicit user feedback to derive decisions, for example, for their release plan by looking at the most reported problems and inquiries.

Software analytics is about generating insightful and actionable information for stakeholders [264]. The use of analysis, data, and systematic reasoning lead to insightful information [165] for stakeholders like developers, testers, and program managers [167, 264]. Menzies and Zimmermann define the term as follows: “Software analytics is analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions.” [165]. Besides the analytics target (software), the definition is very close to the idea of business intelligence & analytics. Zhang et al. state that software analytics employs the three: technologies large-scale computing to handle large amounts of data, machine-learning-based and data-mining enabled analysis algorithms, and information visualization [264]. Further, the authors state that software analytics affects the areas of system quality, user experience, and development productivity, which is also shown by Martínez-Fernández et al. [163]. Software analytics creates insights such as problem reports from users, dependencies in source code, analysis of documentation completeness, and the understandability of the source code [34, 127].

4.2.3 Conclusion

Moving from business intelligence to business intelligence & analytics introduced confusion in the literature, wherefore Chen et al. [38] treat business intelligence and business analytics as synonyms. Following Chen et al., we use both terms synonymously. Therefore, we did not add the term analytics to requirements intelligence. In contrast to business intelligence & analytics, requirements intelligence has a different target and addresses other processes. Software analytics, however, is close to the definition of requirements intelligence. All three fields, business intelligence, software analytics, and requirements intelligence, have the same goal of providing meaningful and actionable information. To achieve that goal, we apply machine learning to large datasets and visualize the results for stakeholders. The term intelligence in our understanding represents, similar to business intelligence & analytics, the collection, analysis, presentation, and dissemination of (requirements-related) information.

We define requirements intelligence as follows:

Definition 4.1: Requirements Intelligence

Requirements Intelligence is a framework that continuously collects, preprocesses, filters, as well as extracts and matches explicit and implicit user feedback to requirements to generate analytical insights for stakeholders in an integrated interactive visualization.

A consequence of the stakeholders' needs and the definition of requirements intelligence is that we have to perform advanced analytics techniques to gain meaningful insights from user feedback. We can achieve the objective of requirements intelligence by applying machine learning approaches to both types of feedback. Therefore, the next section will introduce the framework and its activities. After that, we introduce the machine learning pipeline and practices that we rigorously followed in the later chapters presenting the approaches for the framework activities.

4.3 Framework

This section describes the requirements intelligence framework depicted in Figure 4.1.

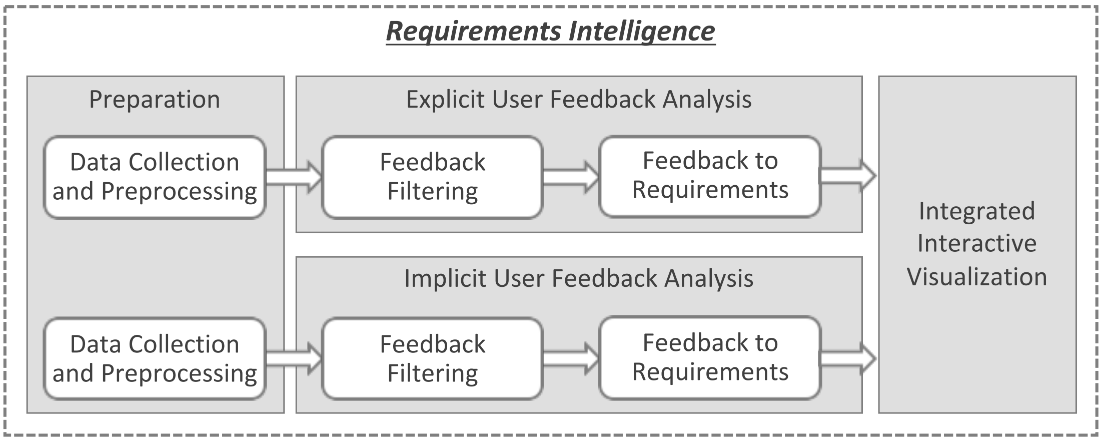

Requirements Intelligence
Preparation
Data Collection and Preprocessing
Data Collection and Preprocessing
Explicit User Feedback Analysis
Feedback Filtering
Feedback to Requirements
Implicit User Feedback Analysis
Feedback Filtering
Feedback to Requirements
Integrated Interactive Visualization

Figure 4.1: The requirements intelligence framework.

The requirements intelligence framework uses explicit and implicit user feedback to provide insights through an integrated interactive visualization. The figure shows four grey boxes, each of them being a thematic container. The first grey box on the left is dedicated to the data preparation. It includes the activities *Data Collection and Preprocessing*, which the framework provides for both types of user feedback. Then, there are two grey boxes, one for each type of user feedback. Again, both include the same activities. The first activity is *Feedback Filtering*, which resolves the primary burden of stakeholders, which is the amount of irrelevant and noisy user feedback. The second activity *Feedback to Requirements* uses the filtered feedback from the previous activity to extract the features from the relevant feedback and matches them with the features stakeholders documented. The output of the explicit and implicit user feedback activities serves as the input for the integrated interactive visualization, which provides analytics insights for stakeholders for each feedback type and their combination. In the following, we detail the activities and provide references to their dedicated chapters that present approaches for implementation. We describe how we approach the framework's activities for explicit and implicit user feedback.

4.3.1 Data Collection and Preprocessing

We discuss the data collection and preprocessing of explicit and implicit user feedback. The main difference between the activity for both feedback types lies within the nature of the data. Explicit user feedback is written text gathered from online sources. Implicit user feedback, on the other hand, is data collected from within the app. In the following, we detail the differences.

Explicit user feedback collection and preprocessing. For explicit user feedback, we focus on app reviews from the Google Play Store and the Apple AppStore, as well as on tweets from Twitter. In Section 2.2, we discussed the platforms and the information they provide in detail. To collect feedback from these platforms, we either have to develop web scrapers or request information via APIs.

Our conceptual model of explicit user feedback in Section 2.2.2 shows that it contains a written (body) text, a timestamp, as well as a language as the minimum. Depending on the platform, explicit feedback may contain additional data, such as a star rating. Our preprocessing steps focus on the body text of the feedback and, therefore, on natural language. Additionally, as we apply machine learning for the activities feedback filtering and feedback to requirements, the preprocessing transforms the feedback to machine-readable data.

There is no dedicated chapter for this activity for explicit user feedback, but we describe the activity in Chapter 5 and Chapter 6.

Implicit user feedback collection and preprocessing. For implicit user feedback, we work with context and usage data collected from apps. In Section 2.3.3, we discussed Android as a platform to collect implicit user feedback. There, we presented examples of context data (e.g., app version) and interaction events (e.g., clicks). We also perform machine learning for implicit feedback and, therefore, have to transform it into machine-readable data.

There is no dedicated chapter for this activity, but both Chapter 7 and Chapter 8 explain the preprocessing steps necessary for their approaches in detail.

4.3.2 Feedback Filtering

This activity addresses the challenge of most stakeholders, which is facing large amounts of irrelevant and noisy user feedback coming from several platforms. Therefore, the main objective of this activity is to filter feedback irrelevant to stakeholders. This step ensures that stakeholders need less effort to analyze user feedback by only having requirements-relevant feedback available.

Explicit user feedback filtering. Chapter 5 introduces an approach for filtering explicit user feedback from app stores and Twitter in the English and Italian languages. The approach utilizes natural language processing and machine learning to filter irrelevant user feedback. We further filter the remaining feedback into the categories problem report and inquiry. Problem reports describe any problem that users encounter in the app like software bugs, or a service provided by an organization. Inquiries are requests for new features and questions directed to the stakeholders.

Implicit user feedback filtering. Chapter 7 introduces an approach that filters implicit user feedback to the context of use. The approach can filter implicit user feedback by automatically identifying private and professional device usage. Private and professional are states describing the usage context (see Section 2.3.2). Our approach only presents one example of how we can utilize context to filter implicit user feedback. Other, more simple context-based filtering approaches are, for example, the device a user is using (e.g., smartphone vs. TV), a particular connection (e.g., Wi-Fi vs. mobile), or the location (e.g., nearby places).

4.3.3 Feedback to Requirements

This activity is about gaining insights into the features users address in explicit user feedback, and the features users use with implicit feedback. With this activity, we can address stakeholders' need to understand the particular features users discuss. With that knowledge, they can better understand which features cause problems and which features users wish. Stakeholders can compare the addressed features with their requirements documentation to learn, e.g., which of their features are popular or maybe not frequently used.

Explicit user feedback to requirements. In Chapter 6, we introduce an approach that stakeholders can apply to extract the app features users address in their feedback and to extract the documented app features from app pages. The approach helps stakeholders identify frequently discussed features and automatically matches the features users mention with those documented on the app page.

Implicit user feedback to requirements. Chapter 8 introduces an approach that automatically detects the app features users currently use. The approach performs the identification solely on interaction events. If stakeholders utilize the approach based on the previously filtered usage context, they can get in-depth insights like what features users use in a working context. Stakeholders can learn more about how users use their apps if they compare the result of this approach for diverse usage contexts.

4.3.4 Integrated Interactive Visualization

The explicit and implicit user feedback analysis activities are the core enablers for the integrated interactive visualization. In Chapter 3, stakeholders stated that they need a web-based automated tool visualizing the analysis results of explicit and implicit user feedback. In particular, they need a tool with which they can stay informed and understand how their app generally performs. Depending on the role of the stakeholders, they either need a general overview like a dashboard or more details like the actual user feedback and the features users address.

The *integration* in the integrated interactive visualization stands for integrating and combining both types of user feedback in one single place. It presents a dashboard with descriptive analytics showcasing, for example, when and how often users provide feedback. Besides dedicated views for the feedback types, the visualization combines both by, e.g., enriching explicit user feedback with context data like the app and operating system version. The underlying machine learning models of the other activities and the combination of both feedback types allow visualizations for all four types of analytics. We address the stakeholders' need for providing a single point of access for different feedback types, coming from different platforms, which is available on the web.

The term *interactive* in the integrated interactive visualization stands for allowing stakeholders to interact with the presented information. The visualization allows stakeholders to, e.g., select filter mechanisms, to define time frames of interests, and to correct the underlying algorithms of the feedback analyses activities. Here, we address the stakeholders' need for being in control of the machine learning models and the need for selecting and filtering information appropriate to the role of the stakeholder.

4.4 Machine Learning Pipeline

The definition of requirements intelligence in Section 4.2 highlighted that to perform analytics techniques, we have to apply machine learning, specifically in the case of predictive analytics. Before applying any type of analytics, we have to follow the activities of the requirements intelligence framework. All activities in the framework either prepare feedback for machine learning approaches, i.e., the activity data collection and preprocessing, or apply machine learning. **This work presents machine learning-heavy (and natural language processing) approaches with rigorous benchmarking to find the optimal machine learning model configuration for the task of the approach**. The machine learning steps are similar for all framework activities and feedback types. Therefore, we present an overview of the machine learning procedure we followed in this section. Each chapter applying a machine learning-based approach explains each step of the machine learning procedure in detail, including a brief introduction to the techniques applied. This way, the reader learns about the techniques when they apply.

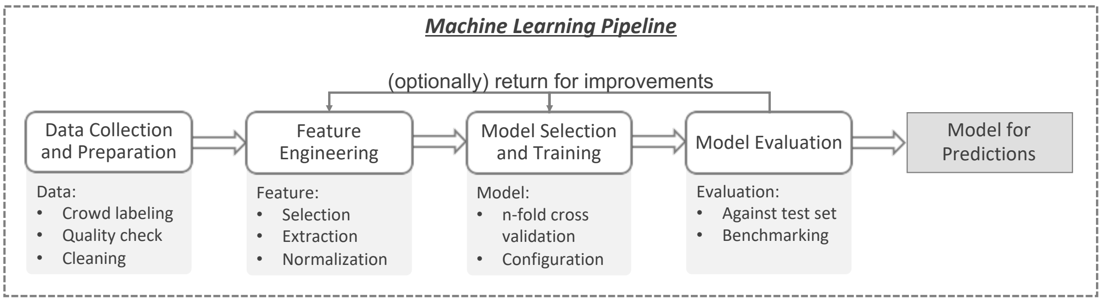


Figure 4.2: The requirements intelligence machine learning pipeline.

Figure 4.2 shows the overview of the machine learning pipeline we utilized for the requirements intelligence approaches. The general idea of the pipeline is to have a standard procedure to produce robust machine learning models for our approaches. In every approach presenting machine learning-based solutions, we performed a benchmark for finding an optimal model. For the benchmark, we performed several machine learning feature extraction techniques and tested different feature combinations. We further selected several machine learning models, tuned their hyperparameters, and tested them with an unseen test set.

The machine learning pipeline contains typical steps suggested by the literature [23, 32, 83, 203]. The developer and advocate at Google Cloud, Yufeng Guo, summarizes the typical steps for machine learning similar to our approach [96]. In the following, we briefly introduce the pipeline steps of Figure 4.2.

Data collection and preparation. For creating supervised machine learning models, we have to collect and prepare the data as a first step. Supervised machine learning models learn to make predictions based on a labeled input. For example, a supervised machine learning approach for classifying user feedback into either problem report or not a problem report (binary classification problem), we have to feed an algorithm with examples. Typically, humans take a look at a sample of user feedback and label it to one of the two categories.

The quality of the data is one of the most crucial parts of machine learning because models fed with, e.g., wrongly labeled data, cannot make accurate predictions [250]. Either domain experts or the crowd (e.g., users, but not necessarily domain experts) can label the collected data (e.g., explicit user feedback). No matter if experts or the crowd perform the labeling task, both need training for understanding the problem, the machine learning algorithm has to solve. We can achieve a common understanding of the problem by performing training or by writing a coding guide. In either approach, we clarify what the exact problem is, what their boundaries are, and give positive and negative examples. By running a prior pilot labeling, we can check if the labeling task is well understood. Our studies rely on crowdsourcing studies for labeling the datasets.

Then, we perform cleaning techniques on the labeled data, such as handling missing values and further split the data into a training and a test set. The training set is part of the model selection and training step, which includes the tuning of the model and performance checking. The test set is only part of the last step of the pipeline, the model evaluation, to ensure that our optimized model is not, e.g., over-fitted and performs well on unseen data.

Feature engineering is the step that uses domain knowledge to create meaningful data representations (features) for machine learning models [222]. Typically, a feature can either be numerical (continuous or discrete) and categorical (ordinal or nominal). Depending on the machine learning algorithm, we can create features of both types or only one of the two types. However, we can transform features from one type to the other. For example, the context data "internet connection" can be any of the categories "Wi-Fi", "mobile", "not connected" (categorical feature). However, most machine learning algorithms can work with numerical features. For that reason, we can either one-hot encode features or assign a number to each categorical value [208]. We can prepare the usage context data and interaction events into a set of features (feature vector). In natural language processing, we also have to transform the text into feature vectors using techniques like bag-of-words [42], tf-idf [232], or fastText [123]. The chapters employing those techniques describe them in more detail.

Besides transforming the data into feature vectors (feature extraction), feature engineering also concerns the selection of features. Sometimes, we do not want to include all possible features (e.g., all context data) but select some of them. The reasons for limiting the number of features are, among others, privacy concerns regarding the collected data [243], the decreasing performance of the model, and longer times for training the algorithms. Therefore, we can perform feature selection approaches like the attribute evaluator *CfsSubsetEval* and the search method *BestFirst* [102]. Feature selection techniques return the features that have the highest impact on the machine learning model's accuracy.

One final step is the normalization of features, sometimes called feature scaling [2]. Data often comes in different scales or different units like counts and percentages. We include feature normalization in our benchmark experiments because normalizing features can improve the machine learning model [30].

Model selection and training. The model selection and training step of the machine learning pipeline is about selecting and optimizing the machine learning models for the train set, which we created in the first step (data collection and preparation).

Our goal is to find the best machine learning algorithm and configuration for each approach of the requirements intelligence framework activities. Therefore, we selected several algorithms available in either the Python Scikit-learn library [194] or the Java Weka data mining software [103]. Most machine learning algorithms allow hyperparameter tuning, which can improve performance [19, 20].

For each selected machine learning algorithm, we run n-fold cross-validation [33, 130, 188] to validate the performance of our models on the train set. Cross-validation splits the train set into a train and validation set for each of the n-folds. This technique then trains a model using the train set and checks its performance using the validation set. After the nth fold, all data points were once part of the validation set. We selected the n-fold parameter based on the number of available training samples. We detail more on our decision in each chapter using n-fold cross-validation. We chose *grid search* for hyperparameter tuning to find the optimal values for various parameters. Grid search exhaustively samples hyperparameter combinations for a defined grid of configurations [20]. Performing this method gives us full control over the hyperparameters.

In case, we thought that the performance of the classification model could be improved, we either changed the hyperparameter grid or went back to the feature engineering step.

Model evaluation. In the last step of our machine learning pipeline, we perform a final test of our trained machine learning model from the previous step. Therefore, the input for this step is the best performing model of the n-fold cross-validation, including hyperparameter tuning. The goal of this step is to check how our model performs on unseen data. For that reason, we created a train and a test set in the data collection and preparation step. The test set usually compares the performance of competing models. It is a curated sample, which includes a real-world representation of the data (i.e., realistic distribution of the classes) [220]. As it is unseen to the trained model, we can make assumptions about the model's performance and compare the results of several models. If the evaluation of the model is unsuccessful, we might have overfitted the model and have to go back to either the second or third step of the pipeline.

We use the evaluation on the test set to create our benchmarks, which are tables describing the performance of each selected machine learning algorithm and its hyperparameters. We only report on the best configuration for each algorithm as sometimes we performed hundreds of machine learning experiments to find an optimized model.

The final result is a machine learning model that we can use to make predictions on new data points (i.e., newly received user feedback).

4.5 Summary

This section introduced the requirements intelligence framework.

Requirements Intelligence is a framework that continuously collects, preprocesses, filters, and transforms explicit and implicit user feedback to requirements to generate analytical insights for stakeholders in an integrated interactive visualization. We defined the term requirements intelligence by looking into the fields of requirements engineering, business intelligence & analytics, and software analytics. We further based the definition on the stakeholder needs we identified in Chapter 3. The definition of requirements intelligence serves as a guide throughout this work.

The explicit and implicit feedback analyses are the core enablers for requirements intelligence. As discussed in Chapter 3, stakeholders find user feedback valuable but rarely utilize it for several reasons like the high amount of received feedback and the amount of irrelevant feedback. We, therefore, propose a framework that continuously collects explicit and implicit user feedback. Then, it filters the collected feedback to only show feedback that is valuable for stakeholders, i.e., requirements-relevant feedback like problem reports. Then, the framework analyzes the features users address in explicit feedback and analyzes how they use it with implicit feedback. Finally, requirements intelligence extracts and matches the documented features with those from the user feedback to enable advanced analytical insights in the integrated interactive visualization.

Requirements intelligence applies rigorous machine learning for its core enablers. The core enablers for requirements intelligence are the analyses of explicit and implicit user feedback. For each feedback type, we perform the three activities *data collection and preprocessing*, *feedback filtering*, and *feedback to requirements*. The data collection and preprocessing activity prepare the data for the feedback filtering and feedback to requirements activities that apply machine learning.

Part II

Core Enablers for Requirements Intelligence

Chapter 5

Explicit User Feedback Analysis: Feedback Filtering

Everything we hear is an opinion, not a fact. Everything we see is a perspective, not the truth.

Marcus Aurelius, Meditations

Publications. This chapter is based on and extends our publications "On the automatic classification of app reviews" [151]. My contributions to this publication were co-designing the interviews, interviewing stakeholders, creating tool mockups, and helping to analyze and discuss the paper's results. Further, this chapter is also based on "Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning" [235]. My contributions to this publication were conducting the research, developing the traditional machine learning part, and leading the analysis and writing.

Contribution. The approach, we introduce in this study concerns the first activity of the *explicit user feedback analysis* of the requirements intelligence framework (see Figure 4.1), feedback filtering. The overall idea of the feedback filtering activity is to identify the user feedback containing requirements-relevant information automatically. For example, if stakeholders are interested in using this activity to find problems for resolving them quicker, the approach implemented for this activity must extract all user feedback that reports problems or hints toward them. Our approach can filter irrelevant feedback and categorize the remaining feedback into problem reports and inquiries. We look into user feedback in different languages coming from two different platforms.

Addressed stakeholder needs. In Chapter 3, we found that stakeholders value user feedback but rarely use it in their decision-making process. The reasons are that user feedback comes in large amounts, from diverse platforms/channels and often contains irrelevant and noisy information. As stakeholders do the analysis manually, they need an automated approach to filter irrelevant user feedback, allowing them to focus on requirements-relevant feedback.

5.1 Motivation

Research has shown the importance of extracting requirements related information from explicit user feedback to improve software products and user satisfaction [140, 190]. Apps with better ratings and frequent feedback get a higher rank in the app distribution platforms, leading to more visibility, and therefore, higher download numbers [73]. But as user feedback on social media or app stores can come a thousandfold daily, a manual analysis of that feedback is cumbersome [187]. However, analyzing this feedback brings opportunities to understand user opinions better because it contains valuable information like problems users encounter or features they miss [78, 99, 112, 187, 191]. Although user feedback contains valuable information for requirements engineers, the vast amount of user feedback is rather uninformative and of low quality [180]. Such uninformative feedback is often spam, praise for the app, insulting comments, or a repetition of the star rating in words [109, 187]. When analyzing explicit user feedback, we typically face the following challenges:

Uninformative Feedback: Pagano and Maalej [187], as well as Guzman et al. [99] show that more than 70% of the user feedback is not related to requirements engineering and therefore considered as noise.

Feedback Channels: User feedback is provided on many different channels, such as social media and app distribution platforms. Development teams have to identify the channels by which users provide feedback, then aggregate that feedback, and analyze it to extract the desired information. As these channels have different purposes, the language of the users may differ. Tweets, for example, have a fixed and limited length and are usually used to state a general opinion or to request support. The purpose of app reviews, on the other hand, is to provide a review and rating of the app itself.

Language Diversity: Apps are usually not restricted to a single country nor a single language. In particular, popular apps have users all over the world and provide their interface in many languages. When users give feedback, they often do so in their native language. Organizations such as *Spotify* are aware of that situation and have dedicated support profiles on Twitter to help their users in their native language. When automatically classifying user feedback, machine learning algorithms usually use language-dependent models for text representations. Therefore, when designing automated approaches and trying to be inclusive to other than English speaking users, it is important to consider the implications of those languages on the machine learning models.

Researchers have applied supervised machine learning to filter noisy, irrelevant feedback, and to extract requirements related information [69, 100, 151]. Most related works rely on traditional machine learning approaches that require domain experts to represent the data with hand-crafted features. In contrast, end-to-end deep learning approaches automatically learn high-level feature representations from raw data without domain knowledge, achieving remarkable results in different classification tasks [87, 230, 262].

This study's overall objective is to develop approaches that automatically extract requirements-related information from explicit user feedback and to filter uninformative feedback. We develop these approaches by carefully addressing the described challenges. In this work, we aim at understanding if and to what extent deep learning can improve state-of-the-art results for classifying user feedback into problem reports, inquiries, and irrelevant. We focus on these three categories because practitioners seek for automated solutions to filter noisy feedback (irrelevant), to identify and fix bugs (problem reports), and to find feature requests as inspiration for future releases (inquiries) [151]. We consider all user feedback as problem reports that state a concrete problem related to a software product or service (e.g., "Since the last update the app crashes upon start"). We define inquires as user feedback that asks for either new functionality, an improvement, or requests information for support (e.g., “It would be great if I could invite multiple friends at once”). We consider user feedback as irrelevant if it does not belong to problem reports or inquires (e.g., “I love this app”).

To fulfill our objective, we employ supervised machine learning fed with crowdsourced annotations of 10,000 English and 15,000 Italian tweets from telecommunication Twitter support accounts, and 6,000 annotations of English app reviews. We apply best practices for both machine learning approaches (traditional and deep learning) and report on a benchmark.

The remainder of this chapter is structured as follows. In Section 5.2 reports on our study design by stating our research questions, our study process, and the dataset the study relies on. In Section 5.3 we detail on how we applied traditional machine learning. Section 5.4 shows our approach to using deep learning. In Section 5.5 we report our classification results. Section 5.6 discusses the implications of the study, while Section 5.7 summarizes the overall work.

5.2 Study Design

We discuss the research questions, our study design, and the data our analysis relies on.

5.2.1 Research Question

The goal of this work is to identify the top-performing model to classify user feedback (tweets and app reviews) into problem reports, inquires, and irrelevant by comparing the traditional machine learning approach with deep learning. We, therefore, state the following research questions:

RQ5.1. To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using traditional machine learning?

RQ5.2. To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using deep learning?

RQ5.3. How do the results of the traditional machine learning approach and the deep learning approach compare, and what can we learn from it?

5.2.2 Study Process

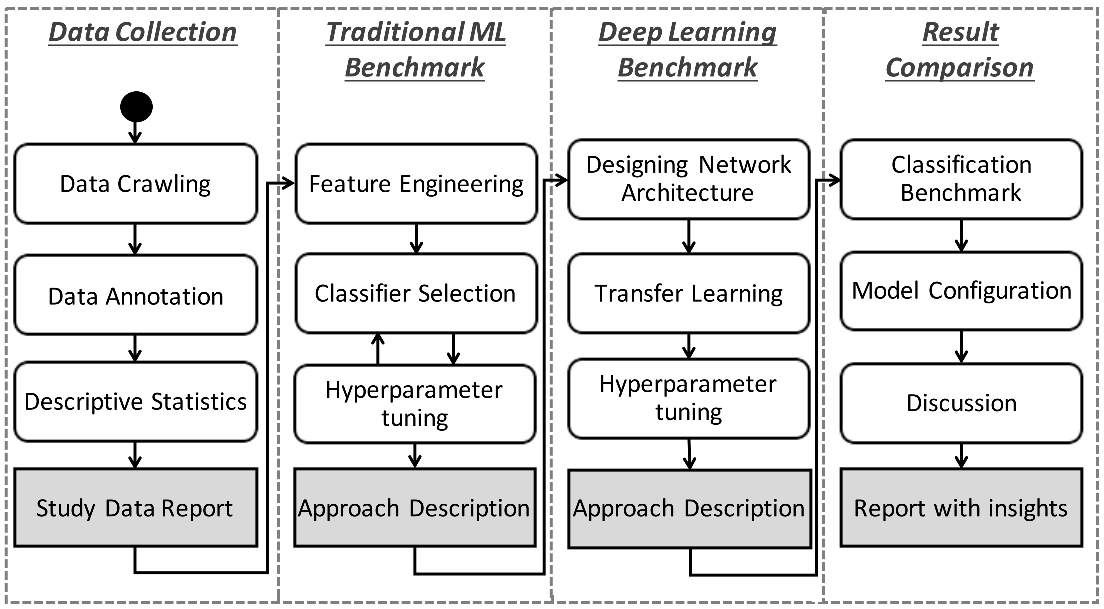


Figure 5.1: Overview of the study design.

Figure 5.1 shows the overall study design. Each white box within the four columns describes a certain step that we performed while the grey boxes show the result that each column produced. The first part of the paper is about the *Study Data*, which we describe later in this section. In the second part, *Traditional Approach*, we perform traditional machine learning engineering, including feature engineering and hyper-parameter tuning. In part *Deep Learning Approach*, we design a convolutional neural network architecture, apply transfer learning for the embedding layer, and finally evaluate the fine-tuned models. In the fourth part *Result Comparison*, we report on the results of our classification experiments (benchmark) comparing the traditional and the deep learning approaches.

5.2.3 Study Data

We collected about 5,000,000 English and 1,300,000 Italian Tweets addressing Twitter support accounts of telecommunication companies. From that corpus, we randomly sampled ∼10,000 English tweets and ∼15,000 Italian tweets that were composed by users. As the annotation of so many tweets is very time-consuming, we created coding tasks on the crowd-annotation platform, *figure*

Table 5.1: Overview of the study data.

	App Reviews	Tweets	
	English	English	Italian
n_problem_report	1.437	2.933	3.414
n_inquiry	1.100	1.405	2.594
n_irrelevant	3.869	6.026	9.794
TOTAL	6.406	10.364	15.802

eight [72]. Before starting the crowd-annotation, we first wrote a coding guide to describe our understanding of problem reports, inquiries, and irrelevant tweets with the help of the innovation center of a big Italian telecommunication company. Second, we run a pilot study to test the quality of the coding guide and the annotations received. Both coding guides were either written or proof-read by at least two native speakers, and we required that the annotators are natives in the language. Each tweet can belong to exactly one of the before mentioned classes and is annotated by at least two persons, three in case of a disagreement. As for the annotated app reviews, we rely on the data and annotations of Maalej et al. [151]. Table 5.1 summarizes the annotated data for both languages.

Replication package. To encourage replicability, we uploaded all scripts, benchmark results, and provide the annotated dataset upon request [204].

5.3 The Traditional Machine Learning Approach

5.3.1 Preprocessing

We preprocessed the data in three steps to reduce ambiguity. Step 1 turns the text into the lower case; this reduces ambiguity by normalizing, e.g., “Feature”, “FEATURE”, and “feature” by transforming it into the same representation “feature”. Step 2 introduces masks to certain keywords. For example, whenever an account is addressed using the “@” symbol, the account name will be masked as “account”. We masked account names, links, and hashtags. Step 3 applies lemmatization, which normalizes the words to their root form. For example, words such as “see”, “saw”, “seen’, and “seeing” becomes the word “see”.

5.3.2 Feature Engineering

Table 5.2: Extracted features before scaling. If not further specified, the number of features applies to all data sets.

Feature Group	Value Boundaries	Number of Features
n_words	$\mathbb{N}$	1
n_stopwords	$\mathbb{N}$	1
sentiment$_{neg}$	$\{x \in \mathbb{Z} \mid -5 \leq x \leq -1\}$	1
sentiment$_{pos}$	$\{x \in \mathbb{N} \mid 1 \leq x \leq 5\}$	1
keywords	$\{0, 1\}$	37 (IT), 60 (EN)
POS tags	$\mathbb{N}$	18 (IT), 16 (EN)
tense	$\mathbb{N}$	4 (IT), 2 (EN)
tf-idf	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	665 (app reviews, EN) 899 (tweets, EN) 938 (tweets IT)
fastText	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	300
TOTAL		1.047 (app reviews, EN) 1.281 (tweets, EN) 1.301 (tweets IT)

Feature engineering describes the process of utilizing domain knowledge to find a meaningful data representation for machine learning models. In NLP it encompasses steps such as extracting features from the text, as well as selection and optimization. Table 5.2 summarizes the groups of features, their representation, as well as the number of features we extracted for that feature group. For instance, the table shows that the feature group "keywords" consists of 37 keywords for the Italian language, each of them being 1 if that keyword exists or 0 if not.

We extracted the *length (n words)* of the written user feedback as Pagano and Maalej [187] found that most irrelevant reviews are rather short. One example of such a category is *rating*, which does not contain valuable information for developers as most of the time, such reviews are only praise (e.g., "I love this app."). Excluding or including *stop words*, in particular in the preprocessing phase is highly discussed in the research. We found papers that reported excluding stop words as an essential step (e.g., [101]), papers that leveraged the inclusion of certain stop words (e.g., [118]), and others that tested both (e.g., [151]). However, the decision for exclusion and inclusion depends on the use case. We decided to use them as a feature by counting their occurrence in each document.

Further, we extracted the *sentiment* of the user feedback using the sentistrength library [244]. We provide the full user feedback (e.g., a tweet) as the input for the library. The library then returns two integer values, one ranging from -5 to -1, indicating on how negative the feedback is, the other ranging from +1 to +5 indicating how positive the feedback is. The sentiment can be an important feature as users might write problem reports in a neutral or negative tone, while inquiries tend to be rather neutral to positive [101, 151, 187]. *Keywords* have proven to be useful features for text classification [104, 151, 249] as their extraction allows input of domain experts' knowledge. However, keywords are prone to overfit for a single domain and, therefore, might not be generalizable. In this work, we use the same set of keywords for the English app reviews and tweets. We extracted our set of keywords by 1) looking into related work [112, 151, 249], and 2) by manually analyzing 1,000 documents from the training set of all three datasets following the approach from Iacob and Harrison [112]. Kurtanović and Maalej [133, 134] successfully used the counts of *Part-of-speech (POS) tags* for classification approaches in requirements engineering. Therefore we also included them in our experiments.

Maalej et al. [151] successfully utilized the *tenses* of sentences. This feature might be useful for the classification as users often write problem reports in the past or present tense, e.g., “I updated the app yesterday. Since then it crashes.” and inquiries (i.e., feature requests) in the present and future tense, e.g., “I hope that you will add more background colors”. When extracting the tense using spaCy [231] the Italian language model supported four tenses while for the English language, we had to deduce the tense by extracting the part-of-speech tags. *Tf-idf (term frequency-inverse document frequency)* [232] is a frequently used technique to represent text in a vector space. It increases proportionally to the occurrence of a term in a document but is offset by the frequency of the term in the whole corpus. Tf-idf combines term frequencies with the inverse document frequency to calculate the term weight in the document.

FastText [123] is an unsupervised approach to learn high-dimensional vector representations for words from a large training corpus. The vectors of words that occur in a similar context are close in this space. Although the fastText library provides pre-trained models for several languages, we train our own domain-specific models based on 5,000,000 English app reviews, 1,300,000 Italian tweets, and 5,000,000 Italian tweets. We represent each document as the average vector of all word vectors of the document, which is also a 300-dimensional vector. We chose fastText for our word embedding models as it composes a word embedding from subword embeddings. In contrast, word2vec [166] learns embeddings for whole words. Thereby, our model is able to 1) recognize words that were not in the training corpus and 2) capture spelling mistakes, which is a typical phenomenon in user feedback.

5.3.3 Configuration

For the experiment setup, we tried to find the most accurate machine learning model by varying five dimensions (no particular order). In the *first dimension*, we target to find the best-performing features of Table 5.2 by testing different combinations. In total, we tested 30 different feature combinations such as "sentiment + fastText" and "n_words + keywords + POS tags + tf-idf".

The *second dimension* is testing the performance of (not) applying feature scaling. Tf-idf vectors, for example, are represented by float numbers between 0 and 1, while the number of words can be any number greater than 0. This could lead to two issues: 1) the machine learning algorithm might give a higher weight to features with a high number meaning that the features are not treated equally. 2) the machine learning model could perform worse if features are not scaled.

In the *third dimension*, we perform Grid Search [20] for hyper-parameter tuning. In contrast to Random Search, which samples hyper-parameter combinations for a fixed number of settings [19], Grid Search exhaustively combines hyperparameters of a defined grid. For each hyper-parameter combination, we perform 5-fold cross-validation of the training set. We optimize the hyperparameters for the F1 metric to treat precision and recall as equally important.

The *fourth dimension* checks whether sampling (balancing) the training data improves the overall performance of the classifiers. For unbalanced data, the machine learning algorithm might tend to categorize a document as part of the majority class as this is the most likely option. In this work, we test both, keeping the original distribution of documents per class and applying random under-sampling on the majority class to create a balanced training set.

Finally, the *fifth dimension* is about testing different machine learning algorithms. Similar to our reasoning for the feature selection, we tested the following algorithms frequently used in related work: Decision Tree, Random Forest, Naive Bayes, and Support Vector Machine [100, 151, 257]. As for the classification, we follow the insights from Maalej et al. [151] and employ binary classification (one for each: problem report, inquiry, and irrelevant) instead of multiclass classification.

5.4 The Deep Learning Approach

Traditional classification approaches require a data representation based on handcrafted features, which domain experts deem useful criteria for the classification problem at hand. In contrast, neural networks, which are used in deep learning approaches, use the raw text as an input, and learn high-level feature representations automatically [87]. In previous work, researchers applied them in diverse applications with remarkable results to different classification tasks, including object detection in images, machine translation, sentiment analysis, and text classification tasks [46]. However, neural networks are not a silver bullet, and they have also achieved only moderate results in the domain of software engineering [67, 76, 97].

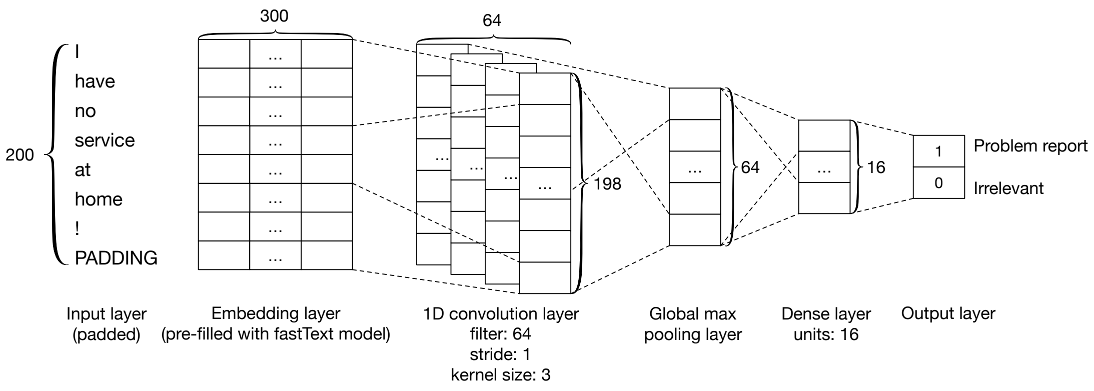


Figure 5.2: Neural network architecture for the classification.

5.4.1 Convolutional Neural Networks

Although convolutional neural networks (CNNs) have mainly been used for image classification tasks, researchers also applied them successfully to natural language processing problems [128, 147]. In most cases, deep learning approaches require a large amount of training data to outperform traditional approaches. Figure 5.2 shows the architecture of the neural network that we used for the experiments in this study. Haering et al. [104] used this model to identify user comments on online news sites that address either the media house, the journalist, or the forum moderator. They achieved promising results that partly outperformed a traditional machine learning approach. The input layer requires a fixed size for the text inputs. We choose the size 200, which we found appropriate for both the app review and the Twitter dataset as tweets are generally shorter, and we identified less than 20 app reviews that exceed 200 words. We cut the part, which is longer than 200 words and pad shorter input texts, so they reach the required length. After the input layer, our network consists of an embedding layer, a 1D convolution layer, a 1D global max-pooling layer, a dense layer, and a concluding output layer with a softmax activation. For the previous layers, we used the tanh activation function. During training, we froze the weights of the embedding layer, whereby 15,000 trainable parameters remain.

5.4.2 Transfer Learning

Transfer learning is a method often applied to deep learning using models pre-trained on another task [87]. In natural language processing, a common application of transfer learning is to reuse word embedding models, e.g., word2vec [166] or fastText [123], which were previously trained on a large corpus to pre-initialize the weights of an embedding layer. We applied transfer learning to pre-initialize our embedding layer with three different pre-trained fastText models [123]. During training, we froze the weights of the embedding layer.

5.4.3 Hyperparameter Tuning

The network architecture and the hyperparameter configuration can be a crucial factor for the performance of the neural network. Therefore we compared variations of both our CNN architecture as well as training parameters and evaluated the best-performing model on the test set. We performed a grid search and varied the number of filters and the kernel size of the 1D convolutional layer, the number of units for the dense layer, the number of epochs, and the batch size for the training, and the number of units for the final dense layer. Due to the small size of our training set, we conducted a stratified 3-fold cross-validation on the training set for each hyperparameter configuration to acquire reliable results.

Subsequently, we evaluated the model with the best-performing hyperparameter configuration on the test set. We trained the models with seven epochs and a batch size of 32. We used the Python library Keras [41] for composing, training, and evaluating the models.

5.5 Results

Table 5.3: Classification benchmark for the traditional machine learning approach (Trad.) and the deep learning approach (DL). The best F1 score per classification problem and dataset are in bold font.

		app review EN				tweet EN				tweet IT			
		p	r	F1	auc	p	r	F1	auc	p	r	F1	auc
Trad.	problem report	.83	.75	**.79**	.85	.46	.82	**.59**	.72	.51	.88	**.65**	.83
	inquiry	.68	.76	.72	.85	.32	.70	**.43**	.73	.47	.82	**.60**	.82
	irrelevant	.88	.89	**.89**	.86	.73	.75	**.74**	.69	.78	.89	**.83**	.73
DL	problem report	.46	.60	.52	.82	.51	.42	.46	.74	.62	.57	.59	.84
	inquiry	.69	.79	**.74**	.94	.40	.40	.40	.75	.51	.57	.54	.83
	irrelevant	.78	.93	.85	.90	.74	.70	.72	.75	.85	.77	.81	.86

In this section, we describe and discuss the results of the classification experiments. We first explain the evaluation metrics. Then, we report on the benchmark in Table 5.3, showing the top accuracy. Finally, we explain the configuration of the models leading to the best results from Table 5.4.

For this work, we report on the classification metrics *precision*, *recall*, and *F1* as presented in related work [101, 151, 249]. For the calculation of these metrics, we used sklearn's strictest parameter setting *average=binary*, which is only reporting the result for classifying the true class. Additionally, we report on the Area Under the Curve *AUC* value, which is considered a better metric when dealing with unbalanced data as it is independent of a certain threshold for binary classification problems. In machine learning, Area Under the Receiver Operating Characteristics *ROC AUC* is a metric frequently used to address class imbalance. Davis and Goadrich [51] argue that Precision-Recall AUC (PR AUC) is a more natural evaluation metric for that problem. We optimized and selected the classification models based on F1, the harmonic mean of precision and recall. Thereby either precision or recall can have a rather low value compared to the other.

Table 5.3 shows the classification results of the best model for each of the three data sets and each classification problem. For the English app reviews, traditional machine learning generally performs better than deep learning when considering the F1 score. One reason for this difference might be that for the app reviews, we have only about 6,000 annotated data points, while for tweets, we have about 10,000 for English tweets and about 15,000 for Italian tweets. For the English tweets, both approaches perform quite similar. While the F1 score seems to be lower for the deep learning approach, the AUC values are similar for both approaches. The results for the Italian tweets show when optimizing towards F1, that deep learning reaches a higher precision, while the traditional approaches achieve a higher recall. The F1 score reveals again that both approaches perform similarly. Based on our results, which are generated by a large series of experiments, we cannot say that for our setup, either of the approaches performs better.

Table 5.4: Configuration of the best performing classification experiments for the traditional machine learning and the deep learning approaches. RF = Random Forest, DT = Decision Tree. CNN = Convolutional Neural Network.

Traditional Machine Learning	app review (EN)	problem report	RF(max_features:None, n_estimators:500). **features**:sentiment, tfidf, **sampling**:true, **scaling**:false
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:4, splitter:random). **features**:tfidf, keywords, **sampling**:false, **scaling**:false
		irrelevant	DT(criterion:gini, max_depth:8, min_samples_leaf:2, min_samples_split:4, splitter:random). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, **sampling**:false, **scaling**:false
	tweet (EN)	problem report	RF(max_features:auto, n_estimators:1000). **features:**sentiment, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:2, splitter:best). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, fastText, **sampling**:true, **scaling**:true
		irrelevant	RF(max_features:none, n_estimators:1000). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, fastText, **sampling**:true, **scaling**:false
	tweet (IT)	problem report	RF(max_features:log2, n_estimators:1000) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:entropy, max_depth:8, min_samples_leaf:10, min_samples_split:6, splitter:random) **features:**n_words,n_stopwords, n_tense, n_pos, keywords, **sampling**:true, **scaling**:false
		irrelevant	DT(criterion:entropy, max_depth:8, min_samples_leaf:8, min_samples_split:2, splitter:random) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, keywords, **sampling**:false, **scaling**:true
Deep Learning	app review (EN)	problem report	CNN(dense_number_units:32, kernel_size:3, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet (EN)	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:16, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet (IT)	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true

5.6 Discussion

5.6.1 Implications of the Results

In this work, we classified user feedback for two languages from two different feedback channels. We found that when considering the F1 score as a measure, traditional machine learning performs slightly better in most of the examined cases. We expect that our approaches can also be applied to further feedback channels and languages, although some features are language-dependent and need to be updated. For example, our deep learning model requires a pre-trained word embedding model on top of a training set for each language, such as the English and Italian fastText models. Word embeddings capture the similarity between words depending on the domain and language. They are highly adaptable to language development by retraining the model regularly on current app reviews and tweets. It can capture the meaning of transitory terms like Twitter hashtags or emoticons. In traditional approaches, the language-dependent features are keywords, sentiment, POS tags, and the tf-idf vocabulary. This requires more effort to create models for multiple languages. The rest remains language and domain-independent.

Traditional approaches often perform better on small training sets as domain experts implicitly incorporate significant information through hand-crafted features [41]. We assume that for these experiments, the hand-crafted features derived from the domain experts lead to considerably better classification results. Deep neural networks derive high-level features automatically by utilizing large training samples. We presume that with more training data, a deeper neural network would outperform the traditional approach.

5.6.2 Field of Application

Classifying user feedback is an ongoing field in research because of the high amount of feedback organizations receive daily. In particular, Pagano and Maalej [187] show that popular apps such as Facebook receive about 4,000 reviews each day. When considering Twitter as a data source for user feedback for apps, Guzman et al. [99] show that popular app development companies receive, on average, about 31,000 daily user feedback. Such numbers make it difficult for stakeholders to employ a manual analysis of user feedback [98]. Recent advances in technology and scientific work enable new ways to tackle these challenges. Our results show that we achieve the best classification results for identifying irrelevant user feedback. For that feedback category, we achieved F1 scores of .89 for English app reviews, .74 for English tweets, and .83 for Italian tweets. Therefore, we can reduce the effort for stakeholders applying our approach to filter the user feedback they receive. However, the results are yet not good enough to ensure that stakeholders do not miss important feedback. As a consequence, we suggest including a human control mechanism. That control mechanism should allow stakeholders to correct the output of the approach to continuously improve it over time.

Stakeholders can use the results of our approach to investigate the feedback categories further. For example, if stakeholders want to understand if there are problems several users report, they can apply approaches like topic modeling. If they apply topic modeling on the feedback our approach classified as problem reports, stakeholders already know that all topics are related to problems. If they apply topic modeling on feedback classified as inquiries, they can find common feature requests.

5.6.3 Alternative Implementations from Related Work

We presented a study giving one example of how to implement the requirements intelligence framework's feedback filtering activity. However, there are several alternatives possible. Here we describe some alternatives found in the literature.

In our study, we focussed on the three categories problem report, inquiry, and others. Research shows that we can find many more categories in explicit user feedback. Pagano and Maalej [187] found 17 topics in app reviews, which they further summarized into the four categories rating, user experience, requirements, and community. Depending on our goals in the requirements engineering process, we could consider including other categories than those identified in our study. For example, if we consider the topic helpfulness, which is about scenarios where the app has proven helpful, we could train a machine to identify this topic. This topic may be particularly interesting to understand how our users perceive our use cases if we cover all intended use cases and if they found unintended use cases. When we understand which of our use cases are perceived as the most helpful, we can leverage this information in our decision-making, e.g., by giving more attention to it in the bug fixing or improvement process.

In our study, we considered explicit user feedback as static entities. However, depending on the platform, user feedback may be updated or evolve in lengthy discussions [107, 124, 157]. App stores, for example, allow users to write feedback and developers to answer on it. Both the feedback and the answer can be updated. In forums or on Twitter, you may have a full conversation. In our study, we did not consider these dynamics, although they can include useful information. For example, Martens and Maalej [157] show that users who give feedback on Twitter, often do not include much information in case of a problem report. The support team on Twitter will reply to those messages asking, for example, for context information, such as the version of the software and steps to reproduce. This information is particularly crucial if we want to use the feedback to create an issue tracker entry for developers [22, 28, 170]. Therefore, a possible alternative for the feedback filtering activity is to bundle messages from a conversation to one that contains all stated information.

5.6.4 Limitations and Threats to Validity

We discuss the internal and external validity of our results as well as the limitations of the approach.

Internal threats to validity. Concerning the internal validity, one threat for conducting crowdsourcing studies to label data is that human coders can make mistakes, resulting in unreliable classifications. We took several measures to mitigate this threat. First, we wrote a coding guide that described our understanding of the feedback categories problem reports, inquiries, and irrelevant. Second, we run a pilot study to test the quality of the coding guide and the annotations received. Third, both coding guides were either written or proof-read by at least two native speakers. Fourth, we further required that the annotators are natives in the language. Fifth, we employed peer-coding, which involved at least two persons, and three in case of a disagreement.

Additionally, there are many more machine learning approaches an, in particular, classifiers we could have included. Specifically, the deep learning part discusses only one approach. To mitigate the threats, we selected several classifiers from the related work that had similar objectives to us. Future research might want to extend the benchmarks by including other deep learning like BERT [57].

External threats to validity. Concerning the external validity, we are confident that our results have a high level of generalizability. We mitigated the threat regarding generalizability by performing rigorous machine learning experiments that report their results on an unseen test set. Therefore, our approach is not prone to overfitting.

Further, we addressed the issue of classifying feedback in different languages. Our approach only considers two languages and, therefore, might not generalize to other languages. We mitigated this threat by describing in detail the effort stakeholders have to invest in creating machine learning approaches for other languages.

5.7 Summary

This chapter introduced an approach to filter explicit user feedback by categorizing English and Italian tweets and English app reviews into irrelevant, problem reports, and inquiries. Therefore, the approach addresses the stakeholder need for automated identification of requirements-relevant user feedback to reduce the amount of feedback they have to analyze. We summarize our main findings.

Traditional machine learning and deep learning achieve similar results. We performed a series of traditional machine learning and deep learning experiments to compare their performance. Due to the number of time-consuming experiments, existing algorithms, and techniques, we had to scope our study. However, this study is the most extensive for user feedback classification to the best of our knowledge. Based on the assumption that deep learning approaches perform better if they have access to large amounts of data, we conducted a crowd-based labeling study that, in total, led to 31,000 labeled feedback. In most cases, we found that the traditional machine learning approach achieved slightly better results than deep learning (the F1 score average median difference is 0.04). We assume that this is the case because, in the feature engineering step of traditional machine learning, we can leverage the domain experts' knowledge for extracting and selecting machine learning features. We further assume that selecting other deep learning approaches could make a difference, too.

The manual effort for applying traditional machine learning is higher than for deep learning. Global organizations have users across the world, meaning that they receive feedback in diverse languages. We selected the English and Italian language to analyze how much effort stakeholders need in configuring and creating a multi-language approach. We further wanted to understand how well we can perform the classification in different languages. The feature engineering step in traditional machine learning requires domain experts because knowledge about the domain guides the extraction and selection of machine learning features. Our study shows that for each language, stakeholders put more effort into feature engineering, as many machine learning features are language-dependent. For example, the terms users use to describe a problem may differ between the languages, and if stakeholders want to include keywords as a machine learning feature, they have to understand the language specifics. Deep learning, on the other hand, does not require a domain expert as it extracts machine learning features automatically. Therefore, the effort for domain experts to apply deep learning is lower.

Irrelevant user feedback identification achieves the best results. One of the stakeholders' primary concern with analyzing user feedback is the large amount of irrelevant feedback (see Chapter 3). To reduce the amount of user feedback stakeholders analyze manually, stakeholders require automated approaches that extract requirements-relevant feedback. Our approach works best for identifying irrelevant user feedback by achieving F1 scores of .89 for English app reviews, .74 for English tweets, and .83 for Italian tweets. Therefore, our approach can reduce the effort for stakeholders. Further categorizing requirements-relevant user feedback to problem reports and inquiries is possible with our approach. We achieve an average F1 score of .68 for identifying problem reports and an average F1 score of .59 for inquiries (both are averages across platforms and languages).

Chapter 6

Explicit User Feedback Analysis: Feedback to Requirements

Look deep into nature, and then you will understand everything better.

Albert Einstein

Publication. This chapter is based on our publication "SAFE: A **S**imple **A**pproach for **F**eature **E**xtraction from App Descriptions and App Reviews"[118]. My contributions to this publication were creating and evaluating the app feature extraction approach, including all steps, performing the analyses, and supporting the writing.

Contribution. This chapter concerns the second activity of the *explicit user feedback analysis* of the requirements intelligence framework (see Figure 4.1), feedback to requirements. We introduce an approach in this study that extracts app features users address in their feedback and that stakeholders document on app pages. Further, the approach can match the app features addressed in the feedback with the documented features on the app pages.

Addressed stakeholder needs. In Chapter 3, we found that stakeholders need to know the features user discuss and the features similar apps provide. In particular, they want to foster innovation and improve the app's quality by understanding the features users address and the features similar apps documented on their app page.

6.1 Motivation

App stores are particularly interesting to the software and requirements engineering community for two main reasons. First, they include millions of apps with the possibility to access hundreds of millions of users [187]. Second, aggregate information from the customer, business, and technical perspectives [73]. Each app in the store is presented by its own app page. This typically includes the app name, icons, screenshots, previews, and the app description as written and maintained by the stakeholders. Users rely on this information to find and download the app they are looking for. Later, some users review the app and comment on its features and limitations. Some apps might receive even more than a thousand reviews per day [187], some of which include hints and insights for the stakeholders.

Over the past years, we observed a "boom" of research papers, tools, and projects on app store analytics [162]. Most of these works focus on mining a large amount of app store data to derive advice for analysts, developers, and users. One popular analytic scenario is to mine app reviews for identifying popular user complaints or requests [78, 112, 152] and assisting release planning [39, 101, 249]. We can analyze the impact features have on the success of an app if we can identify the specific feature users want, or the feature being affected by a problem [48, 218]. Another scenario is to mine the app pages [93, 106] and their evolution over time [217] to derive recommendations for users or to identify combinations that increase download numbers [105, 178].

A common and elementary step that these approaches share is the following: how can the app features included in the natural language text be automatically and accurately identified, in particular since the text is unstructured, potentially noisy, and uses different vocabulary. Previous approaches focus on mining the features from a particular artifact in the store, typically either from the pages or from the review. However, app vendors are likely interested in a holistic approach that works for multiple types of artifacts to be able to combine the customer, business, and technical information.

In this paper, we introduce SAFE, a simple, uniform approach to extract and match the app features from the app descriptions (written by stakeholders) and the app reviews (commented by users). Our approach neither requires an a-priori training with a large dataset nor configuration of machine learning features and parameters and works on single text elements such as a single app review. Based on deep manual analysis, we identify a set of general textual patterns that are frequently used in app stores to denote app features This includes a) 18 part-of-speech patterns (such as Noun_Noun_Noun to represent a feature like "Email chat history"), b) five sentence patterns (indicating enumerations and conjunctions of features), and c) several noise filtering patterns (such as filtering contact information). After preprocessing the text in a single app description or a review, we apply those patterns and extract a list of features: each consists of two to four keywords. Finally, we match the extracted features from the reviews to those extracted from the descriptions.

We implemented our approach and applied it to the descriptions and the reviews of 10 popular apps from the Apple App Store. We also reimplemented two frequently cited state-of-the-art approaches: one focuses on app pages [106], the other on reviews [101]. We evaluated and compared the feature extraction accuracy as well as the matching between the features in the reviews and the descriptions.

6.2 The SAFE Approach

Features have manifold definitions in Requirements Engineering. According to Wieger and Beatty a feature denotes "one or more logically related system capabilities that provide value to a user and are described by a set of functional requirements" [255]. Kang et al. define a feature as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems" [125]. Harman et al. state "a feature to be a property, captured by a set of words in the app description and shared by a set of apps" [106].

Features of software represent an important way to communicate its capabilities to internal and external stakeholders. Features can help users to understand what the software is about and what they can and can't do with it. Features are also important for stakeholders and vendors to structure their software projects and artifacts. They are frequently used in release planning and in issue trackers. Features can also be used in requirements and design artifacts and are also often traceable to source code and APIs.

We focus on high-level features that describe the essential functional capabilities of the software and which are both understandable to stakeholders and users. For instance, the app *Dropbox* has the following features described in its app page: "back up files", "send large files", "access files online", "access files offline", "create microsoft office files", "edit microsoft office files", "share links to files", and "includes storage". In contrast, low-level features contain only a set of words, e.g., Internet, Wi-Fi, Connectivity or photo, picture, upload

There is no uniform way of how stakeholders describe the features of their app, but we recognized some patterns. Descriptions often contain continuous texts, which describe the features and purposes of an app. In addition, features are often summarized in a bullet-point list. In app pages, features from the continuous texts are often repeated within the bullet points.

We exclude games from our analysis (and the evaluation) since we assume that they represent a special type of software [101], where requirements and features are described and handled differently. Murphy-Hill et al. [171] studied game development at Microsoft and found that game developers are "designing [...] an emotional experience [...] which is very subjective [...] and is an artistic achievement." The authors gave the example: "in an e-commerce application, a user has a task to complete that typically takes only a few minutes. In contrast, in some games, people play for hours straight on a daily basis over the course of months. As a result, the requirement for games is that the user should be able to stay engaged on multiple timescales, and the mechanism to achieve that will vary from game to game".

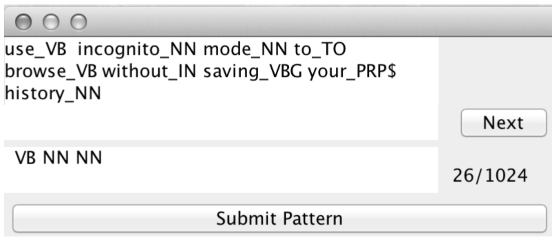
Our approach includes three types of patterns to extract features from app pages and app reviews. First, it defines a set of text patterns that frequently describe features. We identified these patterns based on a manual analysis of real app pages and reviews. Second, we apply these patterns on app pages and app reviews with some pre- and post-processing of the text. Finally, we use text similarity algorithms to match the extracted features from multiple app artifacts.

6.2.1 Identifying the SAFE Patterns

The SAFE patterns consist of two types of text patterns that are frequently used to describe features in app stores: SAFE Part-of-Speech (POS) patterns and SAFE sentence patterns.

SAFE POS Patterns

In a first step, we analyzed app descriptions to identify commonly used POS patterns to denote app features. For this, we crawled the descriptions of 100 apps from the Google Play store. We then POS-tagged the text using the *Natural language Tool Kit (NLTK)* and extracted all sentences. Afterwards, we rendered each sentence in a tool. One researcher manually looked at the sentences to find those patterns. Figure 6.1 shows an example of this task: the upper part shows the sentences in the tool including the POS tags; in the lower part, the researcher can enter the POS pattern (that is the list of POS tags) that describe a feature and insert that pattern to the pattern catalog. In Figure 6.1, the POS pattern is *VB NN NN* (verb, noun, noun) [195]. It describes the feature *use incognito mode*. Table 6.1 summarizes the most frequent POS patterns in our sample that we use in SAFE. We cut off all POS patterns that occurred less than 10 times. The most frequently used SAFE POS patterns include two words. The remaining patterns include up to four words such as "choose from popular versions".


use_VB incognito_NN mode_NN to_TO browse_VB without_IN saving_VBG your_PRP$ history_NN

VB NN NN

26/1024

Figure 6.1: Tool to identify POS tag patterns used to denote app features.

SAFE Sentence Patterns

Existing approaches often miss app features, if a single sentence contains a lot of independent features. For example, the sentence: "send and receive images, videos and sticker" contains the following six app features: "send images", "send videos", "send stickers", "receive images", "receive videos", and "receive stickers". As language has a high ambiguity, we do not cover all possible cases but those we frequently found in the app descriptions. To extract all app features, we analyze the sentence structure and handle five different sentence patterns illustrated in

Table 6.1: Overview of the SAFE POS patterns.

#	POS Pattern	Freq.	Example
1	Noun Noun	183	Group conversation
2	Verb Noun	122	Send message
3	Adjective Noun	119	Precise location
4	Noun Conjunction Noun	98	Phone or tablet
5	Adjective Noun Noun	70	Live traffic conditions
6	Noun Noun Noun	35	Email chat history
7	Verb Pronoun Noun	29	Share your thoughts
8	Verb Noun Noun	28	Enjoy group conversations
9	Verb Adjective Noun	26	Perform intuitive gestures
10	Adjective Adjective Noun	20	Super bright flashlight
11	Noun Preposition Noun	18	Highlight with colors
12	Verb Determiner Noun	14	Share an image
13	Verb Noun Preposition Noun	14	Use depth of field
14	Adjective Noun Noun Noun	12	Fast system virus scanner
15	Adjective Conjunction Adjective	12	Pre-installed and user-installed
16	Verb Preposition Adjective Noun	11	Choose from popular versions
17	Verb Pronoun Adjective Noun	11	Create your greatest album
18	Noun Conjunction Noun Noun	10	Song and artist album

the following, with real examples from app descriptions:

- **Case 1**: "send and receive attachments".
- **Case 2**: "View documents, PDFs, photos, and videos".
- **Case 3**: "Discuss and annotate notes and drafts".
- **Case 4**: "Use camera capture to easily scan and comment on pieces of paper, including printed documents, business cards, handwriting and sketches".
- **Case 5**: "Write, collect and capture ideas as searchable notes, notebooks, checklists and to-do lists".

SAFE identifies *enumerations* (comma-separated text), *conjunctions*, and *feature identifiers* (words might be used to construct a feature) within each sentence (1). Then SAFE searches for occurrences and the placement of conjunctions within the sentence (2). Afterwards, the text parts of the left and the right side of each conjunction are analyzed to identify the placement of possible enumerations and feature identifiers (3). Finally, the feature identifiers are combined with the remaining list of app features (4).

Case 1 is rather simple. It is identified by the single conjunction "and" as well as the feature identifiers to the left and the right side of the conjunction. SAFE combines the feature identifiers to the app features "send attachments" and "receive attachments".

In **Case 2** SAFE identifies a conjunction on the right side of an enumeration so that the approach can extract "view documents", "view PDFs", "view photos", and "view videos" as app features.

Case 3 contains two conjunctions and four feature identifiers. Based on the sentence structure, SAFE creates the cross product of the feature identifiers leading to the app features "discuss notes", "discuss drafts", "annotate notes", and "annotate drafts".

For **Case 4** SAFE first identifies two conjunctions. Then, it finds the two feature identifiers "scan" and "comment" besides the first conjunction in the sentence. At the second conjunction, there is an enumeration on the left side and a feature identifier on the right side, which will be combined to a list of the five feature identifiers "paper", "printed documents", "business cards", "handwriting" and "sketches". In a final step, SAFE combines the first set of feature identifiers "scan" and "comment" with each element of the list resulting in the features "scan paper", "comment paper", and so forth.

Case 5 also contains two conjunctions. SAFE looks at the first conjunction and identifies an enumeration on its left and a feature identifier on its right. The second conjunction also has an enumeration on its left and a feature identifier on the right side. The final step combines each feature identifier of both lists to create app features such as "write notebooks" and "collect to-do lists".

6.2.2 Automated Feature Extraction

Figure 6.2 illustrates the overall steps of SAFE.

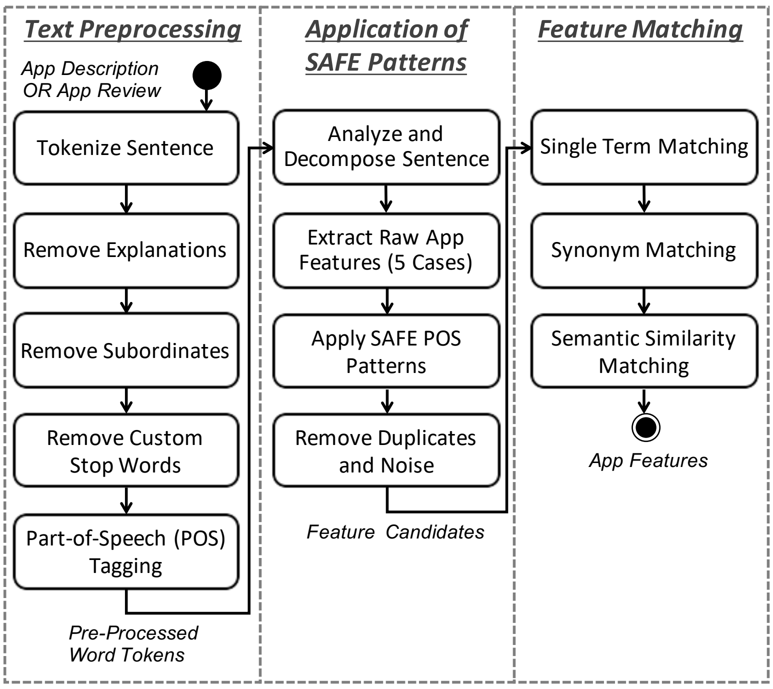

Text Preprocessing
App Description OR App Review
Tokenize Sentence
Remove Explanations
Remove Subordinates
Remove Custom Stop Words
Part-of-Speech (POS) Tagging
Pre-Processed Word Tokens
Application of SAFE Patterns
Analyze and Decompose Sentence
Extract Raw App Features (5 Cases)
Apply SAFE POS Patterns
Remove Duplicates and Noise
Feature Candidates
Feature Matching
Single Term Matching
Synonym Matching
Semantic Similarity Matching
App Features

Figure 6.2: The main steps of the SAFE approach.

Text Preprocessing

The input of the preprocessing is either a single app description or a single review. First, SAFE performs sentence tokenization. Then, all brackets and the text in between are removed as we observed that app features are prominently stated and that text in brackets often simply contains additional explanations. Next, SAFE filters three types of sentences: 1) sentences that contain URLs and 2) sentences that contain email addresses, as those are used to provide contact information, and 3) sentences that contain quotations, as they are most likely to be quoted reviews. Those types of sentences are removed from the data set. Further, we removed all bullet points and multiple symbols (such as '*' or '#') in the remaining sentences. Stakeholders often use these symbols to highlight different parts of the descriptions visually.

Within the clean sentences, SAFE searches for keywords that introduce a subordinate clause. We cut off subordinate clauses because they do not describe features (**what** the app is doing) but give reasons of why they are useful (**how** the app is behaving) as the following sentence shows: "The app let you upload images, so that you don't have to worry about losing them". Next, SAFE wordtokenizes each sentence without changing the order of the words. Thereafter, it removes stop words and the name of the app from each tokenized sentence to reduce the number of words that might introduce noise in the feature extraction [101]. The final step of the preprocessing is to attach Part-of-Speech tags to each word token.

Application of SAFE Patterns

This part starts with analyzing and decomposing the sentences based on the *conjunctions*, *enumerations*, and *feature identifiers*, as described in the SAFE patterns. Then, SAFE applies the sentence patterns on the decomposed sentences to extract *raw app features*. The raw app features and the remaining sentences that did not match any sentence pattern represent the input for the next step. That step uses the SAFE POS patterns from Table 6.1 to extract a list of feature candidates. Finally, SAFE iterates through all extracted feature candidates, removes duplicates and noise such as identical word pairs (e.g., “email email” which matches the SAFE POS pattern *Noun Noun*).

6.2.3 Automated Feature Matching

The input of this step is a list of feature candidates extracted from a single app description and the extracted feature candidates from the related user reviews. SAFE matches the feature candidates from both the app description and the related user reviews. The matching is based on a binary text similarity function at sentence level that is inspired by approaches presented by Guzman and Maalej [101] and Li et al. [141].

Overall, SAFE performs three steps (see Figure 6.2) , each representing a different approach to identify matching features, starting with the most restrictive approach. First, SAFE performs the matching on a single term level. This means that two feature candidates are matching if every single term of them is equal, ignoring the order of the terms. For instance, “send email” and “email send” are considered matching features. Second, SAFE uses the synonym sets of the words captured from WordNet to perform the second step of finding matching features. In this approach, we consider two app features as a match if their terms are synonyms. For instance, "take photo" and "capture image" represent a match. However, the accuracy of this step declines when the number of words of both candidate features is not equal. Hence, we also use a semantic similarity matching at sentence level introduced by Li et al. [141] as our third step. This approach incorporates two similarity metrics: on the one hand, the semantic similarity employing statistical characteristics of a lexical corpus that can be inferred as a domain knowledge base; on the other hand, the order similarity of word order vectors extracted from the sentences. Since we assume that the ordering of the words does not affect the meaning of the features, we skip the word order similarity metric.

Finally, to deal with the candidate features that have an unequal number of words, we utilized cosine similarity calculated between two feature candidates through the semantic similarity algorithm introduced by Li et al. [141]. We set the threshold of the similarity to .7 based on experiments we performed while creating the feature matching approach. Correlation factors between .7 and 1 are considered significant. Some examples of the calculated similarities of candidate features are shown on Table 6.2.

Table 6.2: Cosine similarity of exemplary candidate features.

Feature 1	Feature 2	Cosine Similarity
compose new email	create new email	0.85
taking image	capture image	0.73
send attachments	send attached files	0.63
add image	delete image	0.38
send new email	receive emails	0.35

6.3 Empirical Evaluation

We introduce our evaluation goals, evaluation questions, and data used. We then describe the methodology followed.

6.3.1 Evaluation Goals and Data

We implemented and tested SAFE as a python library and conducted an empirical evaluation to assess the approach's effectiveness and accuracy to extract and match features. We focused on the following evaluation questions:

RQ6.1 How precise is SAFE when extracting features from app descriptions and from app reviews?

RQ6.2 How does SAFE perform compared to other state-of-the-art approaches?

RQ6.3 How accurate can SAFE match the extracted features from the app descriptions and the reviews?

For the evaluation, we created a dataset by crawling descriptions and reviews of 10 apps from the Apple App Store. We selected the top five free and top five paid apps (January 2017) from the category *Productivity* of the Apple App Store. We chose the storefront of the United States to obtain reviews in English. For each app, we gathered its description page using the iTunes Search API [10]. From the app description page, we extracted metadata consisting of 44 values, such as the name, version, description, category, or price. The app reviews were programmatically scraped using a self-developed tool, which accesses an internal iTunes API. A review consists of 10 values, including the reviewer's name, title, description, rating, and date. For further analysis, the data was persisted inside a MongoDB database. To enable replication, we publish the dataset on our website [205].

We also reimplemented two recent, frequently cited approaches for extracting features from app store data: the approach of Harman at al. [106] (also used by Al-Subaihin et al. [3]) focuses on app descriptions and the approach by Guzman and Maalej [101] focuses on for app reviews.

The feature extraction approach by *Harman et al.* includes four main steps [106]. First, it extracts coarse features by analyzing app descriptions of multiple apps to find the part of the description that contains app features. Then, it removes stop words to filter out noise. Afterwards, word frequencies and trigram collocations are computed, which leads to featurelets. Finally, the authors use a greedy clustering algorithm to group featurelets that share at least two words.

The approach of *Guzman and Maalej* focuses on feature extraction on a large set of reviews. The authors gather the title and the comments of each review of an app. Then, they perform several preprocessing steps to extract nouns, verbs, and adjectives, to remove stop words, and to perform lemmatization and group inflected words. Afterwards, the authors perform bigram collocation finding by means of a likelihood ratio test [155]. Thereafter, they filter the collocations by keeping those that appear in at least three reviews while ignoring the order of the words within the collocations. As users might use different terms for the same app feature, the authors group the remaining collocations which have the same synonyms. Finally, they choose the collocation with the highest frequency to be the representation of its group.

6.3.2 Evaluation Method

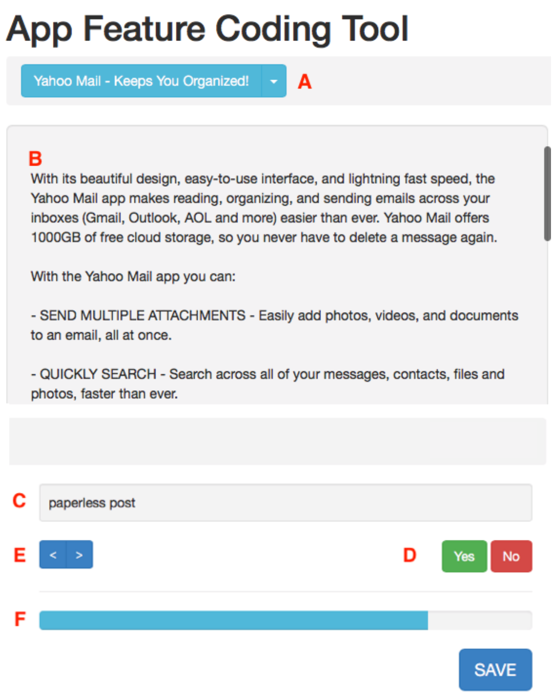
To evaluate the accuracy of SAFE and compare it with the two other approaches, we first created three evaluation sets: for the extraction from app descriptions, for the extraction from app reviews, and for the feature matching. Then we instructed human coders to verify the results of the three approaches. The creation of the evaluation sets are described in each of the following subsections.

The evaluation was performed in two parts. First, we evaluated the app feature extraction on app descriptions and then on app reviews. The benchmark approaches had been replicated to the best of our knowledge. As the approach of Harman et al. focuses on app descriptions, we used it as a benchmark to evaluate the feature extraction from app descriptions. Likewise, as the Guzman and Maalej approach was developed for app reviews we applied their approach as a benchmark for extracting features from reviews. In the following, we explain both procedures.

App Descriptions

For the creation of the evaluation set, two authors independently and manually extracted app features from the description of each app. We compared the extracted app features of both authors and agree on what features should be in the evaluation set. In case of disagreements, a third person was involved. However, this was only necessary in 2 cases.

We then extracted the app features using Harman's et al. approach and SAFE and checked the results with the manual extraction. For this, we implemented a coding tool shown on Figure 6.3. The tool shows the app description on the left side and asks two independent coders to carefully check the manual set and state, on a binary scale, if the extracted app features are part of the app description.



With its beautiful design, easy-to-use interface, and lightning fast speed, the Yahoo Mail app makes reading, organizing, and sending emails across your inboxes (Gmail, Outlook, AOL and more) easier than ever. Yahoo Mail offers 1000GB of free cloud storage, so you never have to delete a message again.

With the Yahoo Mail app you can:

- SEND MULTIPLE ATTACHMENTS - Easily add photos, videos, and documents to an email, all at once.

- QUICKLY SEARCH - Search across all of your messages, contacts, files and photos, faster than ever.

paperless post

Figure 6.3: Coding tool for evaluating the feature extraction from app descriptions. A: The app to be coded. B: The raw app description. C: A feature extracted by an automated approach (i.e., either SAFE or Harman et al.). D: Code if C is a feature and part of the description. E: Navigation. F: Progress bar.

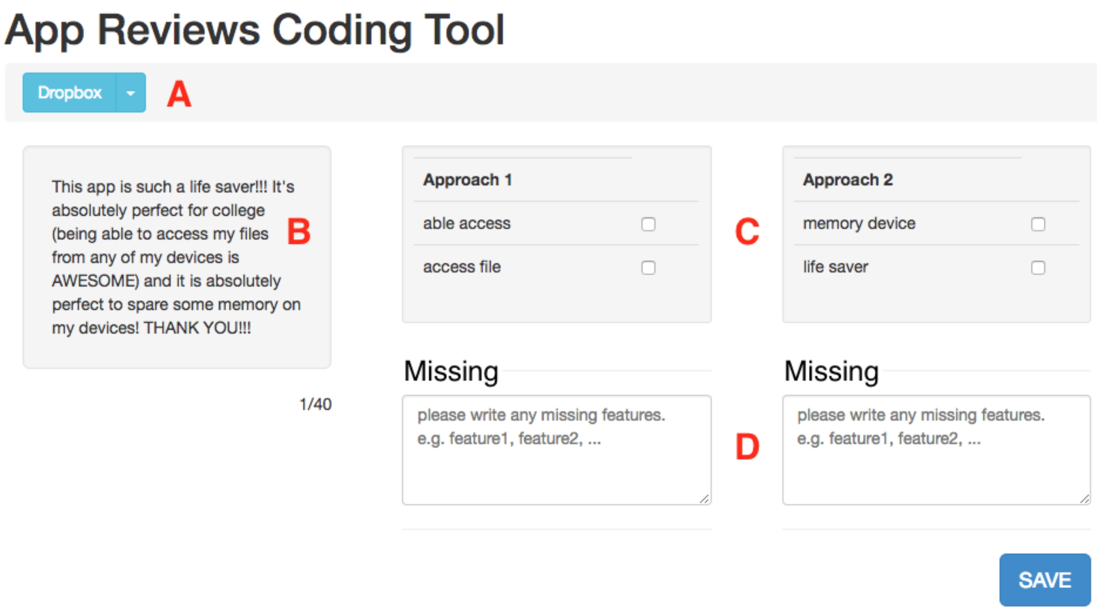
The coding tool displays one extracted app feature at the time. We randomized the extracted app features of both approaches to reduce possible bias that might be introduced if the results of each approach would be shown one after another. Finally, we use this evaluation set to calculate precision, recall, and the f-measure by comparing the coding results with the evaluation set.

For the manual feature extraction, we agreed on the following rules:

1. Focus on functional aspects and omit quality aspects: “Quickly search emails” becomes “search emails”.

2. Focus on the essential feature: “Upload your family photos” becomes “upload photos”.

3. Remove the benefit gained from a feature: "Undo Send, to prevent embarrassing mistakes" becomes "undo send”.


App Reviews Coding Tool

Dropbox

This app is such a life saver!!! It's absolutely perfect for college (being able to access my files from any of my devices is AWESOME) and it is absolutely perfect to spare some memory on my devices! THANK YOU!!!

1/40

Approach 1	
able access	☐
access file	☐

Approach 2	
memory device	☐
life saver	☐

Missing

please write any missing features. e.g. feature1, feature2, ...

Missing

please write any missing features. e.g. feature1, feature2, ...

Figure 6.4: Coding tool for evaluating the feature extraction from reviews. The app to be coded. B: The raw app review. C: The features extracted by both approaches (SAFE and Guzman and Maalej with random order). A checked box marks a correctly identified feature. D: Text field to add features that the approach might have missed.

4. "Open, edit, and save documents" is split to "open documents", "edit documents", and "save documents".

App Reviews

For the evaluation of the feature extraction from the reviews, we selected the 80 most recent app reviews of five apps from our data set. We compared the feature extraction by Guzman and Maalej and SAFE. As the approach of Guzman and Maalej expects a big set of app reviews, we fed it with a random sample of 5000 reviews for each app (except for one app, which only had a total of 3,742 app reviews). In contrast, SAFE expects single app reviews as input.

We also adjusted the coding tool to match the review evaluation, as shown in Figure 6.4. Other than for the app description, the creation of the ground truth was during the coding by manually adding missing features. On the left side of the tool, a single app review is displayed. On the right side, all extracted app features from both approaches are listed randomly. Two independent coders have a binary choice to check if a feature was correctly extracted. To reduce a possible coder bias, the order of the approaches is randomized for each app review. In addition, the coder added the app features each approach missed by writing them down in the appropriate text box below the correlating approach.

6.3.3 Feature Matching

We evaluated the feature matching by using the true positive features extracted from the app descriptions and the true positives from the app reviews. We took the extracted app features from the five apps we used for coding the reviews. We then ran the feature matching of SAFE and created a table containing the results. Table 6.3 shows a simplified example of the evaluation. It includes the app features extracted from the reviews, the app features extracted from the description, as well as the matching result of SAFE and a column in which the coder evaluates the result. Two human coders evaluated the matching results. During the evaluation, the coders also had access to the list of all true positive extracted app features from the descriptions. This enabled the coders to verify if app features for which SAFE could not find a match, really did not contain any.

There are three main use cases for the matching: First, we can identify app features users refer to. Second, we can use this information to identify app features that users refer to but are not described in the app page. Finally, we might be able to extract app feature requests from mismatches. After all extracted app features of a review have been evaluated, the coder will be asked if the approaches missed any feature. This additional step allows us to calculate the recall because we have not created an evaluation set for the app reviews.

Table 6.3: Simplified example of the feature matching evaluation.

Feature (review)	Feature (desc.)	Matching Result	Coder's Statement
send email	send emails	match found	True
month view	week view	match found	False
retrieve images	retrieve pictures	no match	False
email notification		no match	True

6.4 Evaluation Results

We separately evaluated the feature extraction of the SAFE approach for descriptions, reviews, and feature matching. We compared the results of the feature extraction with two recent approaches. We chose the approach of Harman et al. [106], which was designed for feature extraction from descriptions. For feature extraction from reviews, we chose the approach of Guzman and Maalej [101]. We calculated precision and recall for feature extraction from descriptions and reviews as well as for the feature matching as in 6.1 and 6.2.

$$Precision = \frac{tp}{tp + fp} \tag{6.1}$$

$$Recall = \frac{tp}{tp + fn} \tag{6.2}$$

For the feature extraction steps, the precision is the fraction of retrieved features that are part of our manual evaluations set. Recall is the fraction of features that are part of our evaluation set and are retrieved by the respective approach. The true positives (tp) are the number of features that are correctly extracted by the approaches (those that are part of the evaluation set). The false positives (fp) are the number of features extracted by the approaches but are not considered a feature (not part of the evaluation set). The false negatives (fn) are the features that are not extracted by the respective approach but are part of the evaluation set.

6.4.1 Feature Extraction from the App Descriptions

We evaluated the features extracted from the descriptions by our approach and by the approach of Harman et al. SAFE extracts features from the whole description, whereas Harman et al.'s approach only uses texts from the bullet lists of the app page. Table 6.4 shows the results. The evaluation is based on ten apps and contains 197 features (∼20 per app). The Harman et al.'s approach extracted 208 and SAFE 161 feature candidates.

SAFE achieved an overall average precision of .559 and a recall of .434. Respectively, our implementation of the Harman et al. approach achieved an average precision of .278 and a recall of .292. This results in an increase of about 100% in precision and 48% in recall between both approaches. The results are highly dependent on the analyzed app (standard derivation of precision: .21 and recall: .12 for SAFE; .15/.13 for Harman et al.). Both approaches have the highest precision for the *Google Drive* app and the lowest for the *Printer Pro* app. Highest recalls are achieved for *Fantastical 2* (SAFE) and *Cloud App Mobile* (Harman). The low performance for the *Printer Pro* app is because the verb “print”, which is needed to construct app features in this case, occurred outside of the bullet list in the description.

Table 6.4: Evaluation results for app descriptions. # (count), P (precision), R (recall). Highest values in a column are in bold.

App Name	**# Reviews**	**Evaluation Set**	**Approaches**							
			Harman et al. [106]				**SAFE**			
		Feature #	#	P	R	F_1	#	P	R	F_1
Forest: Stay focused, be present	913	13	15	.266	.286	.275	13	.462	.400	.429
Yahoo Mail	29,186	34	25	.400	.294	.339	19	.737	.389	.509
Printer Pro	6,377	11	2	.000	.000	-	14	.214	.250	.231
Gmail	57,212	24	19	.474	.391	**.429**	14	.714	.400	.513
Google Drive - free online storage	21956	17	10	**.500**	.357	.417	8	**.875**	.389	.538
CloudApp Mobile	111	26	41	.317	**.464**	.377	18	.722	.481	.578
Google Docs	18,766	12	13	.231	.231	.231	9	.667	.462	.545
Dropbox	12,563	9	6	.333	.200	.250	10	.300	.300	.300
Fantastical 2 for iPhone	3,724	30	65	.123	.258	.167	46	.500	**.697**	**.582**
iTranslate Voice	11,834	21	12	.333	.211	.258	10	.500	.278	.357
Overall		197	208	.278	.292	.27	161	.559	.434	.458

6.4.2 Feature Extraction from the App Reviews

Similarly, we evaluated the feature extraction from reviews. Users also describe the misbehavior of apps, which are syntactically very similar to a feature description. For example, “The app deleted all my notes". We used the SAFE approach and replication of the Guzman and Maalej’s [101] approach to extract features from reviews. Table 6.5 shows the results for the extractions. Both approaches achieved a comparably low precision whereas the SAFE approach has a significantly higher recall (.709) compared to the approach of Guzman and Maalej (.276), which is a difference of about 157%. The low precision of the approaches can be traced back to the fact that user reviews are very noisy and do not always refer to features as it is often the case in app descriptions.

In contrast to descriptions, reviews often contain slang, typos or incorrect grammar and thus represent a more challenging input for feature extraction. The average recall of .709 means that SAFE does not miss too many features. In practice, it might be more relevant to extract most of the features than the reduction of the noise.

Table 6.5: Evaluation results for app reviews. F (number of features), # (total features extracted), P (precision), R (recall).

		Approaches							
	Manual	Guzman & Maalej [101]				SAFE [118]			
App Name	#F	#	P	R	F_1	#	P	R	F_1
Yahoo	46	147	**.260**	**.283**	**.271**	74	.238	**.761**	.363
Gmail	53	158	.224	**.283**	.250	104	.247	.736	.370
Dropbox	52	154	.237	.280	.257	90	**.260**	.727	**.383**
Fantastical	65	196	.189	.274	.224	146	.230	.652	.340
iTranslate	28	89	.189	.250	.215	60	.213	.679	.325
Overall means	244	744	.218	.276	.244	474	.239	.709	.358

6.4.3 Matching Features in the Descriptions and the Reviews

Finally, we report the results of our evaluation of the matching of the extracted features. The evaluation of the app reviews showed that in total, 178 app features had been extracted correctly by SAFE from reviews. The data set contains these 178 true positives. The feature matching approach of SAFE uses the data set to look for matches in the true positive extracted app features from the app descriptions. SAFE could identify 27 matches of which the coders found that 19 are correct. SAFE was able to achieve a recall of .560 and a precision of .704 to identify matches correctly. On the other hand, SAFE's precision to identify non-matches correctly is .900. Additionally, we calculated the accuracy as it also contains the true negatives and the false negatives, as shown in Equation 6.3. SAFE had a promising accuracy of .870.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{6.3}$$

6.5 Discussion

We discuss how researchers and tool vendors can use SAFE and similar approaches to help software engineering teams analyze and exploit app store data. We then discuss how SAFE can be combined with other approaches to overcome its limitations. Finally, we discuss the threats to the validity of our research and the measures we used to mitigate these threats.

6.5.1 Implications of the Results

Feature extraction is an elementary step for implementing many app store analytic scenarios in practice [151, 153]. Perhaps the most intuitive scenario is *monitoring app features'* health, identifying and continuously measuring which app features are mentioned in user comments, how frequent, and which sentiments are associated with which feature [101]. Moreover, app store analytics approaches that aim at classifying the reviews as informative or not [39] and those that aim at classifying feedback like bug reports, feature requests, rating, and user experience [152] would profit from organizing the results based on app referred features. For instance, bug reports associated with certain features might help stakeholders to faster trace the bugs to the relevant software components or to find other bug reports related to the same feature, which include additional information for the bug reproduction.

The next analytic scenario enabled by feature extraction is the **identification of the features' delta and new insights** that is identifying the differences between features described by stakeholders and those described in the user reviews. The features' delta can even distinguish between different reviews (and thus users') subpopulations, such as features mentioned by certain users, in certain regions, stores or channels (e.g., forums or social media).

Such a difference can reveal two insights: either users apply a different vocabulary or the features discussed in the reviews are missing in the app. In the first case, stakeholders and users might learn how to communicate better and bridge their "vocabulary gap". In the second case, stakeholders can easily get the list of suggested features or at least those that are associated with their apps but not included in them: useful information for requirements identification, scoping, and prioritization. As opposed to analytic approaches that allow identifying reviews with feature requests (e.g., by Iakob and Harrison [112] or by Maalej et al. [152]), feature extraction also allows **filtering the relevant review parts** (e.g., sentences), which include the relevant information – something particularly useful as the quality and structure of the reviews are heterogeneous. Relevant insightful information about the feature request might be in the first or in the last sentence of a review. This will also enable the clustering and aggregation of artifacts (such as comments) based on features as well as identifying feature duplicates in these artifacts.

Finally, feature extraction is the preliminary underlying step in many, if not all, **feature recommendation and optimization** scenarios [105, 178, 217]. Feature recommendations might target both users and stakeholders. For users, feature extraction can help building recommender models that learn dependencies of reported and used features. From the app usage, such models can also derive which features users are interested in. When combining features mentioned in the pages of many apps (e.g., apps within the same category), a predictor model could derive the optimal combination of features, and answer questions such as "which dependencies of features get the best ratings". This might also help to identify which features are missing in a particular app release from the "feature universe".

6.5.2 Field of Application

SAFE has one major advantage since it is uniform: that is, it can be applied to multiple artifacts, in particular on app descriptions and app reviews. SAFE automatically identifies app features in the descriptions and maps those to the app features in the reviews. It identifies fives feature sets:

- App features mentioned in the description.
- App features mentioned in the reviews.
- App features mentioned in the description and in the reviews (intersection).
- App features mentioned only in the description (probably unpopular features).
- App features mentioned only in the reviews (probably feature requests).

Another major advantage of SAFE is that it does not require a training or a statistical model and works "online" on the single app pages and the single reviews. This enables processing the pages and the reviews immediately during their creations. This also reduces the risk for overfitting a particular statistical model to certain apps, domains, or users.

Nevertheless, we think that the achieved accuracy of SAFE—even if it outperforms other research approaches—is not good enough to be applied in practice. We think that a hybrid approach (a simple, pattern, and similarity-based as SAFE together with a machine learning approach) is probably the most appropriate. For instance, machine learning can be used to pre-filter and classify reviews before applying SAFE on them. Indeed, we think that this is the main reason why our accuracy values were rather moderate: because we were very restrictive in the evaluation and applied SAFE on "wild, random" reviews.

SAFE can also be extended by a model that is trained from multiple pages and multiples users. This might, e.g., be used to support users to improve their vocabulary by auto-completing feature terms.

Also, stakeholders should be able to correct the extracted features, e.g., following a critique-based recommender model, or simply building a supervised classifier based on SAFE and continuously improving the classifier model based on stakeholders' feedback (identifying false positives and true negatives). This can either be used to persist the list of actual app features or to train a classifier to learn or adjust patterns per app.

6.5.3 Alternative Implementations from Related Work

Our study focused on identifying the app features either organizations describe, or users address in their feedback. The feedback to requirements activity, however, covers the full spectrum of analyzing requirements-relevant user feedback. The overall idea of this activity is to take the feedback that is relevant for requirements engineering and to perform advanced analysis of this text to gain deeper insights tailored to the needs of the company using requirements intelligence. We there suggest the following examples for alternative implementations of the feedback to requirements activity.

Once we identified the relevant feedback for requirements engineering, we could consider performing aggregation and summarization tasks. For example, a company wants to analyze all feature requests they received since the release. That company received 1,000 app reviews that the feedback filtering activity identified as feature requests. A person needs to read through all of the feedback to get an understanding of the data, and afterwards, may have to apply qualitative methods, such as a thematic analysis to understand topics and clusters within the data. If done correctly, the analysis takes several iterations and quality metrics like peer coding. Automated approaches can support the company by automatically identifying frequently discussed topics, which is an improvement to the previous situation as we were able to split the 1,000 reviews into smaller chunks based on semantic similarity. If we continue the example, the just-described procedure may lead to three semantically different chunks of app reviews, each having a size of about 333. To further ease the effort for the company of what their users discuss, we could perform summarization techniques [95, 100, 257]. With summarization, we could, for example, take each of the three chunks and select one representative review. Other techniques may return common keywords or try to compose a synthetical text made from the whole chunks of reviews. These techniques allow companies to get a quick overview of what their users are saying, and if they find that interesting, they can dig deeper by reading the reviews related to the topic.

In a second example, we consider a company that introduced a new feature in its software and collects user feedback from Twitter. The company aims to understand the users' opinion about that feature and how it is generally perceived. For that, the feedback to requirements activity uses the results of this study as a first step and extracts the features addressed in the tweets. Then, we take the tweets addressing the newly integrated feature and perform sentiment analyses [1, 189, 214]. Guzman and Maalej [101] performed a study in 2014 that is closely related to this idea. From analyzing app reviews, they were able to extract fine and coarse-grained app features and assigned a sentiment score to each feature in the reviews. They suggest that using the extraction and the assignment of the sentiment could help get an overview of the features addressed and decide if there are necessary changes to the requirements.

6.5.4 Limitations and Threats to Validity

As for every study that includes manual coding, the coders in our evaluation might have done mistakes a) indicating wrong features in the descriptions and reviews or b) wrongly assessing the extraction of the SAFE or the two reference approaches. To mitigate this risk, we conducted a careful peer-coding based on a coding guide (e.g., stating what a feature is) and using a uniform coding tool. Therefore, we think that the reliability and validity of the app description and feature matching evaluations – both involving peer-coding – is rather high. For the reviews, we decided to have one single coder for two reasons. First, the large number of reviews would require more resources. Second, the coders were well-trained from the first phase of coding app descriptions. In the future, manually pre-extracting review features, as we did for the features from app descriptions, would improve the reliability. As we share the data, we encourage follow researchers to replicate and extend the evaluation.

Concerning the sampling bias [161], our evaluation relies only on Apple App Store data and apps from one category. While this helped to concentrate the effort and to learn more about the evaluation apps, it certainly limits our results' external validity. However, it is important to mention that the SAFE patterns, the core "idea" of the approach, were identified from Google Play Store data of 100 apps from multiple categories. Since SAFE also works on Apple Store data, we feel comfortable to claim that our approach is generalizable to other apps, except for games which, as described, we ignored on purpose.

As for the accuracy and benchmark values, we refrain from claiming that these are exact values, and we think that they are rather indicative. We think that the order of magnitude of precisions and recalls calculated and the differences between the approaches are significant. We carefully selected our measures to conduct a fair and non-biased comparison. For instance, during the evaluation of the extraction, the tool randomly showed the extracted results in the same layout, without mentioning which approach extracted which features.

Finally, when replicating the reference approaches, we might have made mistakes. We tried to conduct the comparison as fair as possible and implemented all steps described in the papers to the best of our knowledge. In some cases (e.g., list of stop words), we tried rather be spacious and fine-tuned the reference approach to have results comparable to those reported on the papers.

6.6 Summary

This chapter introduced an approach to extract features discussed in explicit user feedback and features documented on app pages. The approach can further match features discussed in the reviews with those documented. Therefore, we created an approach that can address the stakeholder's needs to know the features user discuss and the features similar apps provide. If stakeholders combine this approach with the feedback filtering approach of Chapter 5, they can understand the features users request in inquiries and the features user address in problem reports. We summarize our main findings.

Grammatical rules allow us to extract features from app reviews and app pages. By applying grammatical rules on app reviews and the app's description on the app page, we can identify features without leveraging machine trained models. Stakeholders use the app description of app pages to document their features. They do this in a more structured way than users, by, e.g., utilizing bullet points. Therefore, we achieved better results for extracting features from app pages. The F1 score for that extraction averages at .46, while the average F1 score for extracting features from feedback is at .36.

Feature matching is difficult with few words. Users do not necessarily use the same terms for describing the same features as stakeholders do. For example, *take photo*, *capture photo*, *take selfie*, and *photograph landscape*, which may describe the same feature on a certain abstraction. Therefore, we cannot use simple word matching. As a consequence, we introduced a three-step matching approach. First, we checked on a single term level. Second, we checked synonyms. Third, we perform a semantic similarity matching algorithm. By applying these steps, we achieved an accuracy of .87.

Chapter 7

Implicit User Feedback Analysis: Feedback Filtering

One can state, without exaggeration, that the observation of and the search for similarities and differences are the basis of all human knowledge.

Alfred Nobel

Contribution. This chapter concerns the first activity of the *implicit user feedback analysis* of the requirements intelligence framework (see Figure 4.1), feedback filtering. We introduce an approach in this study that can identify the usage context of mobile device users automatically. In particular, we distinguish between private and professional device usage. Based on a crowdsourcing labeling study, we apply machine learning for a within-users and between-users analysis. The within-users analysis gives insights about how well machine learning can identify the usage context if a user labels their own machine learning model. The between-users analysis investigates how well a single machine learning model can perform if we train it with a set of users and apply it to new users. The study further elaborates on how we can minimize the context data we need to feed the machine learning algorithm.

Addressed stakeholder needs. In Chapter 3, we found that already a few users can generate a million clicks. Stakeholders are overwhelmed with the amount of data and the data representations because, in contrast to the natural language in explicit user feedback, implicit user feedback consists mainly of numerical data.

Further, we found that the usage context is an important type of information to better understand how and when users use the app.

7.1 Motivation

Mobile apps are our everyday companion, supporting us in many situations such as socialization, navigation, health care, or entertainment [121]. Besides private usage scenarios, smartphones are useful productivity tools in many professional situations [129, 197], for instance, to manage tasks and meetings, to plan and track business trips, to handle deadlines and urgent requests, or to simply communicate with colleagues.

Professional organizations are increasingly relying on mobile apps and devices, too. They either offer their employees smartphones and tablets as part of their technical equipment or allow employees to use their private devices—a phenomenon called Bring Your Own Device (BYOD). The aim is often to reduce the overall equipment costs and to increase the flexibility, productivity, and availability of employees [65].

Regardless of these advantages, the blurring boundary between private and professional usage of mobile devices introduces risks [24]. Mobile devices are often less protected than computers [5, 211, 252, 260], which might threaten private and company sensitive data like emails and work documents. Further, users might behave differently, giving less attention to guidelines and organization security policies on mobile than when using office infrastructure. In contrast, if organizations control the mobile devices of their employees with, e.g., Device Management Systems (DMS), there is a significant risk that private data will be collected with corporate data, which threatens the employees' privacy [35, 227].

The main goal of this work is to study the feasibility of using supervised machine learning to classify private and professional device usage automatically. This approach enables stakeholders to filter implicit user feedback based on the usage context, while the usage context, in this case, is private and professional.

To achieve this goal, we first conducted a crowdsourcing study to understand how and when users are using their devices privately and for work. We then use the crowdsourced data to create machine learning models that automatically classify mobile device usage. We performed a series of experiments to create a benchmark that compares the results of machine learning algorithms frequently used in related work. This benchmark reports on a within-users analysis.

As privacy begins with the question of what data to collect, we investigated how we can minimize the variety of data used in the machine learning models. To understand what data is necessary to achieve an accurate classification for private and professional usage, we performed feature selection and created a second benchmark. We found that for the within-users analysis, we achieve similar performance for the full and the minimized feature set with precision and recall values up to .99 for all study participants.

Our approach relies on supervised machine learning models. Hence, users have to label data before we can train a machine learning model. The need for labeling data introduces a barrier for new users who want to benefit from the approach. Therefore, we look into the generalizability of the machine learning models by reporting on a between-users classification benchmark that creates a machine learning model from a subset of users and provides it to new users. For the between-apps analysis, we found that using the full feature set leads to an average F1 score of .95, while the minimized feature averages at .87.

The remainder of the study is structured as follows. In Section 7.2, we describe our study design, including important terms and the data collection in a crowdsourcing study. Section 7.3 and Section 7.4 presents the *privacy benchmark* along with a within-users and between-users analysis. Section 7.5 discusses the results and provides insights on how to use the approach. Section 7.6 concludes the study.

7.2 Study Design

We discuss the research questions, our study design, and the data our analysis relies on.

7.2.1 Research Questions

In this work, we answer the following research questions.

> **RQ7.1 To what extent can we automatically classify private and professional mobile device usage?** Several classification algorithms sup-

port supervised machine learning, but they may have different results in terms of accuracy (precision, recall, and F1-score). Therefore, we compare conventional classifiers to search for the best result by performing a benchmark. We answer this research question in Section 7.3.1.

RQ7.2 What is the minimal feature set for an accurate and reliable classification? We study the accuracy of the classification under the consideration of data minimization. We use as few features (i.e., data collected) as possible for two reasons: (1) to reduce the privacy concerns of the approach and (2) to increase the computing performance and lower battery consumption. To answer this research question, we create a second benchmark that compares the accuracy of four classifiers with a minimized feature set. We explain our results in Section 7.3.2.

RQ7.3 To what extent can we use a trained classification model across multiple users? One challenge is to create a classification model that we can use without new users having to label data. In this study, we analyze the accuracy and the features to generalize our approach by creating a single classification model made of the data from different persons and test it with data that is unseen to the model. We compare the performance of the full and the minimized feature sets of RQ7.1 and RQ7.2. We answer RQ7.3 in Section 7.4.

7.2.2 Study Process

Figure 7.1 depicts our overall research process. The figure includes two thematic columns that correspond to our main phases: *Crowdsourcing Study* and *Machine Learning Experiments.*

The crowdsourcing study had the goal of collecting labeled data for the machine learning approach (first column). We first developed an Android app for the labeled data collection. Then, we ran a pre-study to check that the app can collect the data we need and to confirm the feasibility of the approach. Finally, we conducted a two-week crowdsourcing study with 18 participants to get labeled data. The result of the crowdsourcing study is the labeled data (our raw study data).

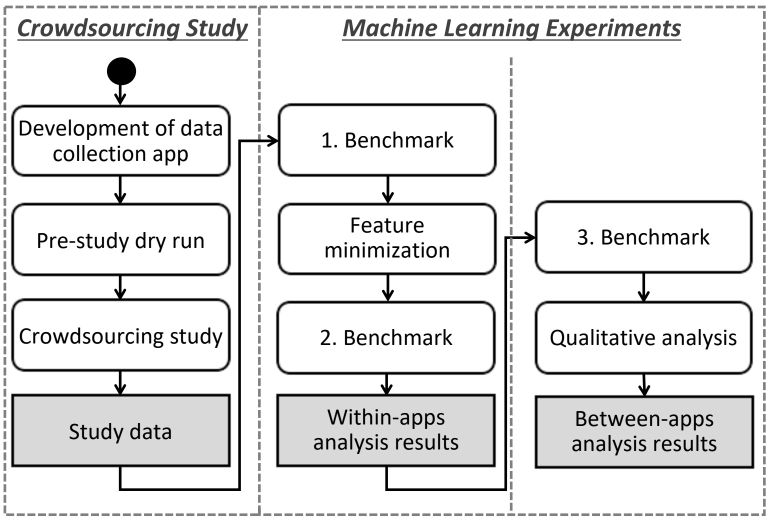


Figure 7.1: Steps of the study.

The machine learning experiments are the core of classifying the labeled data into private and professional device usage. The automatic classification considers two perspectives, the within-users analysis, which is about one user collecting context-data, and a personalized classification for that user. We created the first benchmark using all collected context types. Then we minimized the number of context types and created the second benchmark. The output of these steps is the within-apps analysis results in Section 7.3. The between-users analysis builds a classification model from a subset of people and applies that model to others, unseen people (test set). We create the third benchmarking reporting the between-users classification results and then qualitatively discuss why some cases performed better than others.

The remaining part of this section focuses on the crowdsourcing study. We describe the crowdsourcing study, the participants, and the collected implicit user feedback (the labeled dataset).

7.2.3 Study Data

Here we describe how we collected our study data for the machine learning experiments.

Data Collection App

For the machine learning experiments, we had to collect real-world (not synthetic) labeled implicit user feedback. For the data collection, we developed an Android app that is capable of collecting the context types that we introduced in Table 2.3 of Chapter 2. We decided to target Android because of its high market share [113] and its powerful APIs, which are less restrictive than those of iOS.

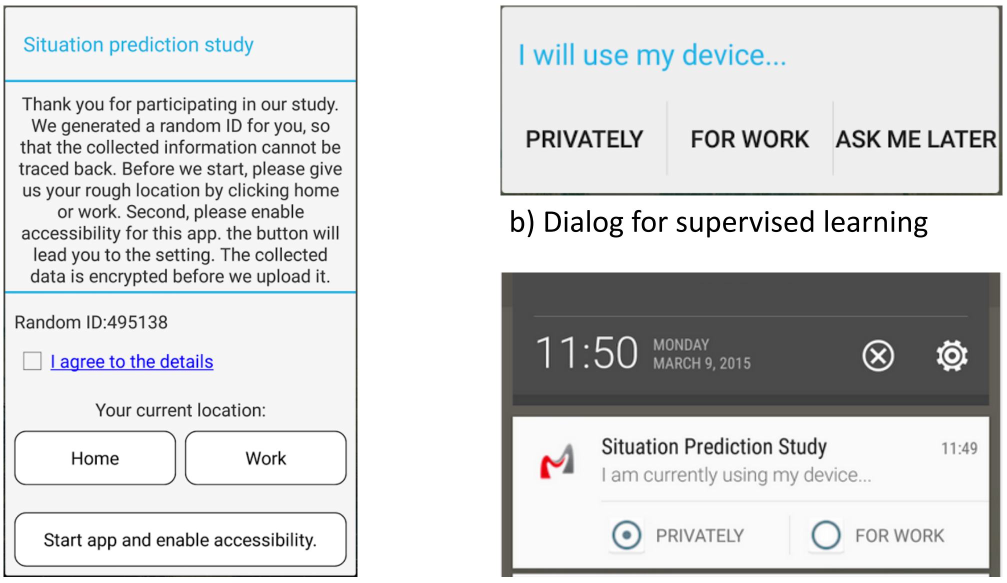
As a reminder, we describe the context and interaction data we collected. The **App Sensor**[software|pull] identifies the current foreground app that the user sees. The **Interaction Sensor**[software|push] monitors touch events like clicking, scrolling, and writing, but also reads the text of the interacted element. The **Location Sensor**[hardware|push] retrieves the longitude and latitude of the user's location. The **Connectivity Sensor**[software|push/pull] is a sensor that bundles several sub sensors collecting, e.g., the BSSID or the encryption of the current network. The **System Sensor**[software|pull] gathers general information such as the time, date, and running background services.

Crowdsourcing Study

First, we conducted a dry run with four participants to check the feasibility of our approach and to confirm that the app is collecting all the necessary data. The result was a refined version of the data collection app, which we used for the crowdsourcing study.

Afterwards, we run the crowdsourcing study. In the crowdsourcing study, we collected context data to analyze and understand if we can automatically detect private and professional device usage. For this purpose, we developed an Android app that uses the sensors previously described.

After the installation of the app, a dialogue opens that asks the users about their consent, as well as whether he/she is currently at *home* or *work*, as shown in Figure 7.2 a). Afterwards, the app runs in the background and collects context data in real-time. Whenever the screen of the device is unlocked, the app asks the user about the usage purpose, as shown in Figure 7.2 b). The dialogue has three options: *Privately*, *For work*, and *Ask me later*. If the purpose of the usage is unclear, users could choose *Ask me later*, which re-opens the dialog when the screen turns off. In some cases, a user might also want to switch between private and professional usage during a session. For instance, a user chats with


Situation prediction study

Thank you for participating in our study. We generated a random ID for you, so that the collected information cannot be traced back. Before we start, please give us your rough location by clicking home or work. Second, please enable accessibility for this app. the button will lead you to the setting. The collected data is encrypted before we upload it.

Random ID:495138

☐ I agree to the details

Your current location:

a) Initial installation dialog

I will use my device...

b) Dialog for supervised learning

11:50 MONDAY MARCH 9, 2015

Situation Prediction Study 11:49

I am currently using my device...

c) Notification to update the usage type

Figure 7.2: Screenshots of the study app.

her partner and receives an important email from the office and starts reading it. In such a case, the participants were able to adjust the current usage in the notification bar, as shown in Figure 7.2 c). A session is the time between unlocking the device and locking it again.

Before we started the crowdsourcing study, we distributed a manual to the study participants that explained the process of the crowdsourcing study. After these preparation steps, we started the two-week crowdsourcing study with 18 participants from six different countries, of whom six were female, and 12 were male (demographics in Table 7.1). Nine participants were from academia, employed by universities and research institutes, while the remaining nine participants were from private companies of different sizes. From the 18 initial participants, six either participated for less than a week or did not enable all sensors as described in the manual; e.g., the accessibility sensor needs manual approval from the user. As a consequence, we removed these participants from the data analysis. Since we anonymized the data and did not track the users' identities, we cannot determine which participants of Table 7.1 we had to remove. From the 12 participants who submitted complete data, we collected 88,596 context events that occurred in 6,486 sessions. In total, the participants labeled 1,268 sessions as professional and 5,205 sessions as private. Therefore, about 80% of the data

Table 7.1: Overview of the study participants.

#	Gender	Age	Country	Position	Affiliation	Experience
P1	M	25-34	Germany	Researcher	University	2 years
P2	F	25-34	Germany	Researcher	University	3 years
P3	M	25-34	Germany	Researcher	University	2 years
P4	M	25-34	Germany	App Developer	Health App Company	4 years
P5	F	35-44	Germany	Lead Engineer	Telecommunication Company	10 years
P6	F	25-34	Austria	Researcher	Research Institute	1 year
P7	M	25-34	Austria	Researcher	Research Institute	4 years
P8	M	25-34	Austria	Researcher	Research Institute	5 years
P9	F	25-34	Spain	Researcher	University	3 years
P10	M	35-44	Spain	Associate Professor	University	1 year
P11	M	35-44	Spain	Software Engineer	Information Security Company	12 years
P12	M	35-44	Spain	Chief Product Officer	Information Security Company	12 years
P13	M	65-74	Spain	Project Manager	Information Security Company	24 years
P14	M	35-44	Spain	Chief Operating Officer	Information Security Company	5 years
P15	F	55-64	Italy	Project Manager	Telecommunication Company	20 years
P16	M	55-64	Italy	Project Manager	Telecommunication Company	20 years
P17	F	25-34	Tunesia	Software Engineer Intern	ERP & Health App Company	2 years
P18	M	25-34	Switzerland	Student Assistant	University	1 year

belongs to private sessions and 20% to professional sessions. On average, each participant had 540 sessions with a standard deviation of 475.

7.3 Results: Within-Users Analysis

This section describes the results for RQ7.1 and RQ7.2 by performing a benchmark between different classifiers, namely: *Decision Tree (J48)*, *Decision Table*, *Naive Bayes*, and *Support Vector Machine (LibSVM)*. We also report on the feature selection experiments to reduce the number of collected features.

We chose the classifiers based on related work and our own experience. Shin et al. [224] used several *Naive Bayes* models for predicting mobile app usage on smartphones. They also compared their results with the *Decision Tree* classifier, which Guo et al. [18] employed as a *Regression Tree* in. The *Support Vector Machine* is a widely used classifier for two-class classifications that can handle unbalanced data [18].

We report on our results using *precision*, *recall*, and *F1-score*, which we calculated for each classifier. The precision represents how many of the classified instances are correct. The recall is the percentage that shows how many true positives we identified correctly. The F1-score is the harmonic mean of the precision and the recall. In our study, precision and recall are equally important because if our approach would be used for, e.g., real-time classifications, each misclassified could harm the application that relies on our approach. Therefore, we are also reporting on the F1-score, which takes both values into account.

Table 7.2: Mean accuracy of evaluated classifiers for all participants using the **full** feature set with 10-fold cross validation.

Classifier	Usage Type	Precision	Recall	F1
Decision Tree	Private	0.978	0.987	**0.983**
	Professional	0.951	0.873	**0.908**
Decision Table	Private	0.952	0.985	0.968
	Professional	0.902	0.734	0.797
Naive Bayes	Private	0.919	0.880	0.898
	Professional	0.581	0.735	0.626
LibSVM	Private	0.946	0.949	0.946
	Professional	0.752	0.591	0.639

7.3.1 Classification Benchmark

The first research question RQ7.1 aims to study the extent to which different classifiers can determine if the device usage is *private* or *professional*. Table 7.2 shows the results of the classification using 10-fold cross-validation and using all context data described in Table 2.3 as features. The results shown in the table are the mean values across all participants. We attached the detailed classification benchmarks per participant in the appendix. The table highlights the highest F1-score across the classifiers in bold font. We report on the precision, recall, and F1 score of a binary classifier deciding whether the current device usage is private or professional. We present our metrics on both classes to avoid reporting an accuracy paradox [248] due to our previously discussed data imbalance. In the full feature set, we used all the collected context data to see how accurate our potential results in classifying private and professional usage can be.

When comparing the classifiers in Table 7.2, we found that all classifiers perform better than a random classification. The *Decision Tree* classifier achieves the best results for both usage types and across all participants. This classifier has mean F1-score values of >0.980 for private and >0.900 professional usage. Table 7.2 reveals that the *Decision Table* classification has similar results to the *Decision Tree* for classifying private usage, but detecting professional usage is about 0.11 worse. In Appendix A.1, we show the complete benchmark results for each participant. It shows that, e.g., the *Decision Table* classifiers' minimum and maximum F1-scores are 0.197-0.954 for professional and 0.895-0.997 for private usage, while the *Decision Tree* classifiers' F1-scores range from 0.738-0.995 for professional and 0.944-0.999 for private usage. The *Naive Bayes* and *SVM* classifiers seem to have difficulties for classifying professional sessions with a comparably low and wide range of the F1-score. We were able to increase the accuracy of the *Naive Bayes* classifier during our experiments by using the supervised discretization of Weka [103]. As shown by Dougherty et al. [62], the discretization of continuous features can improve the overall accuracy. In our case, we could improve the accuracy of the *Naive Bayes* classifier by up to 15% for professional usage. Nevertheless, as shown in Table 7.2, the accuracy of *Naive Bayes* falls behind the other classifiers. The mean values show that all classifiers have high precision, recall, and F1-score for classifying private sessions with values greater than 0.890. The mean values of the F1-score of all classifiers range from 0.898-0.983 for private usage, while professional usage ranges from 0.626-0.908. We found that balancing the datasets achieved the best classification results when we looked at each participant.

7.3.2 Feature Set Minimization

The second research question RQ7.2 aims to investigate which and how many features we need for accurate classification. The reason for reducing the number of features is twofold. First, we want to minimize the data collection of sensitive and private data as this can increase the trust in such a system [145]. Sheth and Maalej [223] found that users' privacy concerns are data breaches and data sharing. Van der Sype and Maalej argue from the legal perspective that collected data should always be proportionate to the original purpose of a system [243]. Furthermore, we want to reduce the use of resources as this approach should solely run locally to avoid potential data leakage. In the following, we first explain the process of minimizing the number of features and then report on the results of the second classification benchmark.

First, we executed the default feature selection algorithm of Weka (attribute evaluator *CfsSubsetEval* [102] and the search method *BestFirst*). We ran the feature selection separately across all participants and looked at the most informative features that the algorithm calculated for at least three participants. We then made different combinations of these features by using as few features as

Table 7.3: Mean accuracy of evaluated classifiers for all participants using the **minimized** feature set (app, day of the week, time of the day, Wi-Fi encryption, and number of available Wi-Fi networks) with 10-fold cross validation.

Classifier	Usage Type	Precision	Recall	F1
Decision Tree	Private	0.977	0.987	**0.982**
	Professional	0.950	0.864	**0.901**
Decision Table	Private	0.952	0.982	0.967
	Professional	0.890	0.735	0.792
Naive Bayes	Private	0.920	0.920	0.920
	Professional	0.685	0.655	0.661
LibSVM	Private	0.957	0.959	0.957
	Professional	0.920	0.681	0.739

possible to check which combination achieved the best result. Table 7.3 illustrates the resulting benchmark. The final features of this process are the *app*, *day of the week*, *time of the day*, the *number of available Wi-Fi networks*, and *Wi-Fi encryption*. The most informative feature is the app (for 9 participants). The second most informative feature is the hour of the day, which the feature selection calculated for 8 participants. This process eliminated the privacy-intrusive features location and the interaction data. We assume that the location clusters have no high impact because we found that especially private sessions are done independently of the location. On the other hand, information of the Wi-Fi connection such as the BSSID, the number of available Wi-Fi networks and the Wi-Fi encryption could also indicate a rough location, because these features do not frequently change in common places like home and the office. Touch interactions also do not improve the classification accuracy, because this information might be too detailed as, e.g., touch events on specific elements like buttons dependent on the app used. The table in Appendix A.1 shows the complete benchmark results for every participant.

In the following, we compare the classifiers and their results for the minimized feature set described in Table 7.3. With the minimized feature set, we were able to get close to the results of the full feature set. Again the *Decision Tree classifier* performed best. For the *Decision Tree* and the *Decision Table* classifier, the mean F1-score differs by less than 1% compared to the full feature set. For professional usage, the *Decision Tree* outperforms the other classifiers as the *Decision Tree*'s mean F1-score is 0.901 while the second-highest mean is 0.792.

By looking at the mean values of the *Naive Bayes* and *LibSVM*, we can see that compared to the full feature set, the accuracy increased in the minimized set of features. In this case, we assume that the variety of features negatively influenced the classification. The *Naive Bayes* classifier performance improved for private usage by 0.022 for the F1-score, which was due to the 0.040 rise in the recall. For professional usage, the *Naive Bayes* classifier improved its mean F1-score by 0.035, the precision increased by 0.104 while the recall decreased by 0.080. The *LibSVM* classifier improved its mean F1-scored by 0.011 for private usage and by 0.100 for professional usage.

From these results, we conclude that it is possible to focus on the most informative features and to remove the privacy-intrusive ones to keep high overall accuracy.

Concluding the section Within-User Analysis, we found that the classification of private and professional usage has promising results. Further, our feature selection section highlighted that reducing the features for the classification can still achieve results with an accuracy of above 90%. We were able to remove the two privacy-intrusive features location and interaction data because other features like the package name or the time of the day have a high information gain.

7.4 Results: Between-Users Analysis

We present the results of RQ7.3 with a between-user analysis, in which we created a classification model from a subset of people and re-used that model for others. This way, we reduce/remove the effort of new users to label their own usage sessions, decreasing the hurdle for users actually to use this approach. In the following, we describe how we build and test the classification models. Afterwards, we do a qualitative analysis of the outliers of the classification to create a better understanding of the difficulties.

In our study, we are limited by the data from 12 participants. As data minimization is important to this work, we test the between-user analysis on the full and minimized feature set. For this, we combined the labeled data of 11 participants to train the model and then tested it for the remaining participant. We did not include the labeled data of the test set in the classification model to simulate

Table 7.4: Between-user classification: Accuracy of J48 *Decision Tree* with full and minimized feature sets. Difference values relate to the results of the within-user analysis for the tested/unseen participant.

Test Set		Full Feature Set		Min. Feature Set	
(Unseen P)	Usage Type	F1	Diff.	F1	Diff.
P1	Private	0.997	− 0.001	0.993	− 0.005
	Professional	0.871	− 0.034	0.707	− 0.198
P2	Private	0.999	+ 0.001	0.998	± 0.000
	Professional	0.974	+ 0.019	0.959	+ 0.010
P3	Private	0.986	+ 0.024	0.979	+ 0.022
	Professional	0.991	+ 0.015	0.987	+ 0.013
P4	Private	0.969	+ 0.025	0.959	+ 0.017
	Professional	0.898	+ 0.101	0.866	+ 0.072
P5	Private	0.984	− 0.007	0.986	− 0.001
	Professional	0.670	− 0.164	0.745	+ 0.005
P6	Private	0.990	− 0.003	0.989	− 0.005
	Professional	0.674	− 0.064	0.653	− 0.114
P7	Private	0.985	+ 0.003	0.967	− 0.012
	Professional	0.913	+ 0.038	0.802	− 0.072
P8	Private	0.996	+ 0.008	0.996	+ 0.008
	Professional	0.986	+ 0.026	0.985	+ 0.028
P9	Private	0.982	+ 0.031	0.979	+ 0.023
	Professional	0.990	+ 0.016	0.989	+ 0.012
P10	Private	0.999	+ 0.001	0.999	+ 0.001
	Professional	0.976	+ 0.009	0.976	+ 0.015
P11	Private	0.995	− 0.004	0.993	− 0.004
	Professional	0.979	− 0.019	0.974	− 0.013
P12	Private	0.995	+ 0.007	0.996	+ 0.004
	Professional	0.963	+ 0.060	0.969	+ 0.036
Mean	Private	0.990	+ 0.007	0.970	− 0.012
	Professional	0.907	+ 0.001	0.775	− 0.126

a new user. We repeated these steps in a way that each participant was the test set once (simulated as a new user). We trained the model with the J48 *Decision Tree*, as this classifier outperformed other classifiers within our study. Table 7.4 reports on the calculated F1-scores. Further, it reports on how this classification model performs in comparison to the results we reported in the Within-User Analysis. For instance, Table 7.4 shows that the difference for participant P1 in the full feature set for professional sessions is -0.034. The difference shows that the performance of this participant decreased by -0.034 with classification model because of the F1-score in Table 7.2 (full feature set) of P1 is 0.905 for professional sessions while the F1-score of this classification model is 0.871.

Table 7.4 shows that for the full feature set, classifying private usage achieves a mean F1-score of 0.990, which is an improvement of 0.007 compared to the mean in Table 7.2 of the Within-User Analysis. The classification of professional usage, on the other hand, has a mean F1-score of 0.907, which is a decline of 0.001. The performance of the minimized feature set declined by 0.012 with a mean F1-score of 0.970 for private and declined by 0.126 to 0.775 for professional usage. From these results, we conclude that having a classification model of multiple persons can free a new user of labeling a dataset on their own when using the full feature set. The results for classifying professional usage, based on the minimized feature set, might be too low for being practical.

To understand why some classification models work better than others, we also analyze the outliers. As classifying private usage achieves high results for the full feature set (>0.969) and >0.959 for the minimized feature set, we concentrate on the professional usage. Participant P6 has a low F1-score for classifying professional usage in both feature sets. Taking a closer look at the data of this participant, we found that for both feature sets, the *Decision Tree* uses the Wi-Fi encryption on the first level. About 90% of the misclassified instances in the minimized feature set for professional usage are using the Wi-Fi encryption [WPA-PSK-TKIP][WPS][ESS]. The lower F1-score results from the fact that both classification models used this Wi-Fi encryption solely for private usage. Participant P5 losses 0.164 of the F1-score for classifying professional usage in the full feature set. Again, after checking the classification model, we found that similar to the previous case, the model uses the Wi-Fi encryption on the first level. In 67% of the misclassified instances, the Wi-Fi encryption of P5 is unknown, which means that the device had no connection to any Wi-Fi network. Therefore, there is no BSSID. However, as the BSSID is part of the second level, the model goes to the third level of the *Decision Tree*, which is the used app. P5 used *Google Now*'s search field professionally, although, in this specific combination of features (Wi-Fi encryption, BSSID, and app), the training set contains 1,000 private usages associated to this app. Because this participant used Google Now's search field for professional usage when no Wi-Fi connection was available, the F1-score decreased.

Summarizing the results, training a single classification model based on participants from different countries and job positions can lead to a reasonable accuracy when used for new users. We are aware that these results are limited to our dataset and might not be generalizable. A further study might focus on the generalizability of this approach by looking into specific groups of people such as colleagues in the same office or grouping job positions such as consultants.

7.5 Discussion

We discuss the implications of the results, the field of applications, and present related wor for alternative implementations regarding feedback filtering.

7.5.1 Implications of the Results

Our benchmark results show that we can classify the usage of mobile devices into private and professional with high precision and recall values automatically. The average F1 was about 0.90 for professional usage and .098 for private usage. We also found that we can train the classification models based on a subset of people and apply it to new users. This finding is very encouraging, given the main hurdle of supervised machine learning approaches, which is to label data (for training the model) before it can be deployed and used.

Another encouraging finding is that we were able to minimize the number of needed context types for accurate classification. We found that the minimal set of features needed is: the app name, day of the week, time of the day, number of available Wi-Fi networks, and Wi-Fi encryption. We consider this list of context types less privacy-intrusive than the full feature set. Moreover, we can capture these context types at the session start, which allows for an immediate classification. In this case, we can also consider the classification as automated identification or prediction of the session's context usage.

7.5.2 Field of Application

The main use case for this approach is filtering implicit user feedback for stakeholders who want to analyze user feedback. Filtering implicit user feedback based on the usage context allows stakeholders to learn when and how users use the app in which context. Stakeholders can either focus on a particular context or compare different usage contexts.

Further, we can use the approach to user enhance privacy. For example, if we develop the approach as a background app on Android devices, we can control what data other apps can access, send to a corporate server, or share with third-party apps, even if its a company-owned device.

The third field of application addresses data security. Security policy enforcement frameworks such as the MUSES project [174], who aim at preventing and avoiding information security risks introduced by the behavior of users, can benefit from our approach. Such policy enforcing systems often process the usage and context data on servers in the cloud. Our approach allows on-device classification without the need to share sensitive data. Further, we can limit the data collection of policy enforcement systems to professional usage sessions.

If Android would include our approach in the operating system, we can improve and automate functionality like *Android for Work* [90]. In *Android for Work* and other *Device or Space Management Solutions*, users have to explicitly select whether they want to perform private or work-related tasks by, e.g., opening specific apps or by specifying it beforehand. Our approach can automate this step. Further, we could leverage the data from *Android for Work*, and similar systems to continuously train and improve our machine learning model.

The next field of application is the development of time management systems that automatically log private and professional activities helping to create productivity, billing, time tracking, and overtimes reports. Such a system could understand the distribution of work and create a more accurate overview of the total worked hours. Employers can use that field of application, too, to get a better overview of the workload of their employees.

Finally, our approach can help to personalize advertisements. Apps can use our approach to show advertisements that are related to the current usage purpose. During professional-related tasks, we can show advertisements related to the work, while we can show ads for personal interest can as soon as the device detects private usage.

7.5.3 Alternative Implementations from Related Work

Our study focused on privacy and security scenarios by analyzing context data to understand if a device is used privately or for work. However, the analysis may look completely different depending on the domain that needs to analyze context data. Here, we present two alternative ideas for analyzing user context data.

Our first example is an analytics tool that provides information about the performance of the software (computing environment context). Mobile app monitoring is a field in research that targets the performance of apps in terms of, e.g., CPU, memory, and battery usage [192, 200] or response times for apps relying on online services [242]. This kind of analysis can find weak spots in the software that either slow down its performance or unusually drains the energy of the device. Identifying these weak spots can be useful in requirements engineering to understand how well certain aspects of an app perform, and in case it has a bad performance, maybe re-worked in a future iteration. For example, we may find that sending chat messages takes five seconds on average, although, the non-functional requirements envision a performance of less than 1 second. Continuous monitoring performance indicators could help for improving an existing software early; in the best case, before users write negative reviews [216].

In the second exemplary scenario, we are a company providing a service for taxi drivers that helps them finding customers, navigates them to the customer, and to the customer's target location (physical context). In this example, we want to analyze the location of the user (the taxi driver) to support insights for the driver and the company the driver belongs to [56]. Both parties want to optimize their routes to have short travel times without a customer to maximize the earnings [37]. Besides, the driver and the customer may have personalized preferences [137, 168]. The driver might want to stop working soon, while the customer might either want to be at the target location as quick as possible or needs more than four seats. By analyzing the location of the driver and the customer either with GPS or with Wi-Fi-based data, we can receive their exact position and optimize their routes [142]. In requirements engineering, we can use this detailed feedback on the people's locations to test and simulate different algorithms and options for finding the best routes. Further, we can check their performance results against our non-functional requirements metrics.

7.5.4 Limitations and Threats to Validity

We discuss the internal and external validity of our results as well as the limitations of the approach.

Internal threats to validity. Concerning the internal validity, one potential threat is related to the comparison of the classifiers. As the number of the used classifiers is limited to four, we might have missed one that could have better results than the *Decision Tree.* We might improve the approach by extending the performed benchmark with further classifiers, by further tweaking the parameters of the chosen classifiers, and by considering other techniques like association rules.

The reported results rely on the crowdsourcing study, we could think of possible participants' bias, as the participants might have selected the wrong label or forgot to update the label during the usage. To avoid bias by unintentional wrong labels, we have shown the currently selected usage in the notification bar of Android, so that the participants were aware of what is currently selected. Furthermore, we created a manual for the app that created awareness of how to use the application, the possibility to change the label at any time, as well as the information, that the currently selected label is visible in the notification bar.

External threats to validity. The number of participants might have influenced the reported results. We tried to mitigate the side effects by choosing participants with different demographics to avoid the possible bias of focusing on a specific group of persons. However, since we had only a few participants, there is a threat to the generalizability of the approach.

Although the *Decision Tree* classifier performed best in our study, we cannot guarantee that this will be the case for different participants. Therefore, our results are rather indicative. For instance, the *Decision Table* often has similar results to the *Decision Tree* and might generate better results for other participants outside this study.

Our approach works on Android devices. The feasibility of our approach might differ on other devices, like computers and laptops, and platforms, such as iOS. Although we focused on Android, we think that we cover many mobile devices because of the high market share of Android, which is about ~87%.

Finally, after creating a classification model, the accuracy might change or decrease over time. Since users might install and uninstall applications, change their jobs or interactions with the device in general. The classification accuracy might decrease if the model does not adapt to such changes.

7.6 Summary

This chapter introduced an approach to identify the usage context automatically. In this study, we automatically distinguish between private and professional usage contexts based on 88,596 labeled context events from 12 participants.

Therefore, we created an approach that can address the stakeholder need to filter the amount of implicit user feedback, because already a few users can generate a million clicks and gigabytes of data. Further, stakeholders want to know the usage context as it helps to understand better how and when users use the app. We summarize our main findings.

Excluding privacy-intrusive context data has little impact on the performance of the approach. First, we performed a benchmark with all collected context data (see Section 2.3.3) and achieved a promising result of an average F1 score of .945. After we reduced the included number of context types by excluding privacy-intrusive context types, we still achieved an average F1 score of .941, which is a small difference of 0.004 compared to the full feature set.

The between-users analysis shows that we do not have to train a machine learning model for each user. If we train the machine learning model with only a subset of users and apply it to unseen users, we achieve a high average F1 score of 0.95 using all context types and an average F1 score of .87 using the limited set of context types. Therefore, we do not need every user to provide labeled data for creating this machine learning model.

Chapter 8

Implicit User Feedback Analysis: Feedback to Requirements

All truths are easy to understand once they are discovered; the point is to discover them.

Galileo Galilei

Contribution. This chapter concerns the second activity of the *implicit user feedback analysis* of the requirements intelligence framework (see Figure 4.1), feedback to requirements. We introduce an approach in this study that, based on crowd-labeled interaction events, automatically learns the features users use. In a within-apps analysis, we identify the features users use within one particular app by taking all the labeled interaction events of that app and train and evaluate a machine learning model based on that data. In a between-apps analysis, we combined the labeled data of all apps to understand if the user interactions per feature are similar between different apps. We further analyze how the machine learning models automatically identified the feature usage and found that with a few human-readable machine learning features, we can reach explainable and accurate conclusions.

Addressed stakeholder needs. In Chapter 3, we found that stakeholders want to understand which interactions, in which context, led to the particular problems users reported. With the interaction events, they want also to create steps to reproduce, helping them to fix problems quicker.

8.1 Motivation

The market for mobile apps is highly competitive and dynamic. The two major app stores, *Apple App Store* and *Google Play Store*, include ∼4 million apps [239]. To fix bugs and try out new features, 14% of the app vendors release a new version at least bi-weekly [164]. Users who are unsatisfied with certain aspects or features of an app are likely to look for alternatives [14, 74, 256]. This can quickly lead to the fall of even previously popular and successful apps [139]. In this environment, continuously monitoring and understanding the changing needs and habits of users is indispensable for the successful survival and evolution of the app.

Emerging app analytics tools aim at coping with this challenge. They collect and process user or issue tracker data to derive actionable insights for stakeholders and vendors and to inform future decisions, e.g., on what to release next [80, 153, 163, 175, 249], App analytics tools can be divided into two categories based on what data they collect and analyze: user feedback or usage data [153]. *User feedback* analytics tools focus on what users say or write about the app and its features, e.g., in review platforms or social media. In recent years software engineering researchers and vendors have suggested many tools in this category, ranging from tools to classify comments into bug reports, feature requests, or uninformative text [39, 81, 151, 159, 249] to tools [8, 245] summarizing the user opinions about the various features of the apps [101, 143, 156]. *Usage data* analytics tools focus on how users actually use the app. State-of-the-art approaches either log low-level execution events (e.g., stack traces and method calls to augment crash reports) [89, 108], or collect and visualize general high-level usage information (such as the app activation time, location, and device) [27, 55, 61, 91, 251, 259]. Other studies use interaction data to make predictions about the next app used [241, 266] or tasks within one application [240].

This work targets an intermediate level of usage analytics, which focuses on *app features*, use cases, or other user-perceivable aspects of an app. These features are communicated to users, e.g., in release plans and app pages, [118] and discussed by them in review platforms and social media [107, 144]. They are often implemented across multiple components, classes, and methods. It is thus hard for stakeholders to precisely scope what (e.g., from the UI) exactly belongs to certain features such as "browse a catalog" or "send a message" as it can be used in many different ways, following many different execution paths. This makes it hard for stakeholders to monitor single events, denoting the usage of features. On the other hand, understanding how app features are used, whether they are used at all, by whom, and in which context can be informative to developers, analysts, and managers to scope the development work, understand users' requirements, and set up release priorities.

We suggest a novel, app-independent approach to enable app feature analytics. The main idea is simple: we log general user interaction events such as *scroll view* or *get a notification* and use them for training a machine learning model that relates the app features to the interactions. We conducted a crowdsourcing study with 55 participants who shared 55GB of interaction data and manually labeled 5815 feature usages of 170 unique apps for 18 days.

Our within-apps evaluation shows that we can achieve encouraging precision and recall values already with ten labeled feature usages. For certain popular features such as "browse newsfeed" or "send email" we achieved F1 values above 0.88, between-apps feature learning also seems feasible with F1 values up to 0.82.

The remainder is structured as follows: In Section 8.2, we introduce the overall research methodology, including the terminology, crowdsourcing study, research data, and machine learning experiment setup. In Section 8.3, we present a within-apps analysis, which details our results to RQ8.1. Section 8.4 answers RQ8.2 by reporting our between-apps analysis. Section 8.5 discusses the impact of our results on software engineering and shows the limitations. Section 8.6 concludes the paper.

8.2 Study Design

The main terms underlying this work – "interaction" and "features" – have been frequently used in literature with multiple definitions and interpretations. In this work, the term **app feature** denotes a label or a short English phrase that represents a user-visible aspect, a certain characteristic, or functionality of the app [125]. Particularly, we consider verb-noun pairs from the app pages such as "send email" and "watch video" as app features [73, 101, 105, 118]. A verb can be a verb-phrase or two-term verb or, e.g., "dial-down", and similarly for a noun, e.g., "tv-episode". Typically, app features are either listed by the vendors as bullet points or screenshots in the store's app pages or referred to in reviews and tweets by users [118, 177]. In this work, we defined **interaction events** as the interaction of a user with a mobile device (e.g., click and swipe on the screen), which can automatically be observed and which are not specific to certain apps (to enable an app-independent solution). In Chapter2 Table 2.2, we summarized common types of mobile interaction events, which can, e.g., be monitored using the Android accessibility framework or other generic UI frameworks. These are the interaction types we consider in this approach.

8.2.1 Research Questions

The main goal of our research is to learn app feature usages from the interaction events. This requires a large-enough dataset of app interaction data submitted by real app users and augmented with the names of the app features which have been used. We formulate the following research questions.

RQ8.1 To what extent can we identify the usage of certain app features based on interaction data of different users of an app? (within-apps analysis)

RQ8.2 Can we learn app feature usages between apps? (between-apps analysis)

8.2.2 Study Process

To answer our research questions, we followed a three-phased research methodology, as shown in Figure 8.1. In the first phase, we conducted a crowdsourcing study to collect research data. We developed a data collection and labeling app, conducted a pre-study to select the study participants on a crowdsourcing platform carefully, and finally executed the study. The second phase was concerned with machine learning experiments, including the understanding and analysis of the collected data, the clean-up and filtering of data useful for machine learning, as well as the implementation and testing of the machine learning pipeline and models, baselines, and evaluation metrics. Finally, in the third phase, we conducted a series of benchmark experiments, comparing multiple machine learning algorithms and configurations to answer RQ8.1 (within-apps analysis) and RQ8.2 (between-apps analysis). For all benchmark experiments, we perform a quantitative and qualitative analysis of the results.

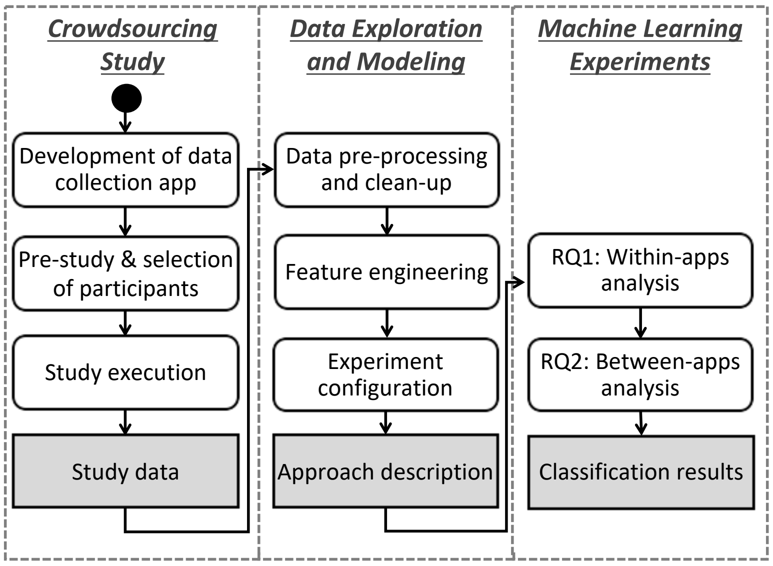

Crowdsourcing Study
Development of data collection app
Pre-study & selection of participants
Study execution
Study data
Data Exploration and Modeling
Data pre-processing and clean-up
Feature engineering
Experiment configuration
Approach description
Machine Learning Experiments
RQ1: Within-apps analysis
RQ2: Between-apps analysis
Classification results

Figure 8.1: Overview of the research methodology.

8.2.3 Crowdsourcing Study

Relying on supervised machine learning to predict app feature usage requires training a model with labeled data (truth/training set). To the best of our knowledge, no comparable dataset exists from literature. For gathering such data, we chose to ask crowd workers to label and share their interaction data with apps they regularly use without imposing any restrictions on them. Another app users' study would have been reasonable only if we had had direct access to many users of various apps – in order to get enough interaction data with enough labels (to train the model) corresponding to multiple apps with multiple features (for a realistic and generalizable setting). A synthetic dataset collected in experimental setups, e.g., with students or volunteers, would have seriously threatened the reliability and validity of our results as the user behavior should be as natural, spontaneous, and unbiased as possible.

We developed an **Android app**, which we distributed on the crowdsourcing platform Amazon Mechanical Turk (AMT). AMT is one of the largest crowdsourcing platforms, allowing access to a large sample of crowdworkers [6]. The app collects the interaction events listed in Table 2.2 of Chapter 2 based on Android's Accessibility Framework. It also allows users to label their usages with the app features they just used, as shown in Figure 8.2. The labeling dialog can either be activated by the users at any time or shown to them when they try to lock the screen or change the app. We targeted Android because of its high market share, particularly on the AMT, and of its APIs, which are less restrictive than those of iOS. Nevertheless, our approach can be used in iOS apps, too, based on UIKit APIs [9].

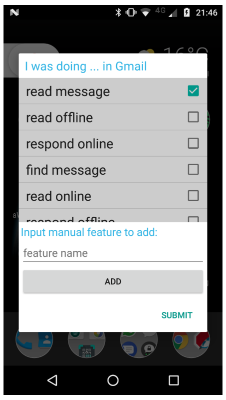


Figure 8.2: Study app to label interactions with features used.

The study **participant selection** is crucial for getting real, high-quality, reliable data. As a first step, we restricted the number of countries in which the study was available on ATM to 12, as we found that for some countries (such as India), it was difficult to retrieve data due to an unstable internet connection. We also allowed only crowdworkers with the highest rating (showing their past performance) to participate. Second, we distributed a questionnaire to the participants to understand which apps they are frequently using. This step is important as we had to manually extract the app features from app descriptions before starting the labeling study. Third, the participants installed the data collection app and used it in a pre-study for two consecutive days. This enabled us to test not only the app robustness but also a participant's ability to provide useful data. For this, we evaluated the following criteria:

Trustworthiness. The crowd-worker must use daily at least 70% of the frequently used apps stated in the questionnaire (must list apps). Moreover, at least ten unique apps should be used.

Datapoints. The mobile phone should be used at least five times a day, and the usage time should be at least 30 minutes daily.

App Features. The size of the intersection between *installed* apps and the master app list should be at least ten. Moreover, the number of intersection between their used apps during the trial and the master app list should be at least four.

By applying those selection criteria on 2-days to the pre-study participants, we sampled a total of 55 participants for the real study.

In the real study which we conducted in April 2017, each participant was paid a base fee of 2.50 USD and got an additional 0.05 USD for each label provided during the study. To ensure that a participant does not perform dozens of labels per day just to get a big payment, we have set an upper limit of 16 daily labels for which the participant will get paid. Therefore, each participant can earn a maximum of 16.90 USD. We also explained to participants that it is most efficient for them to simply use their phones as usual and label after each session. Trying to fake the usage would instead lead to more work since too short usage sessions were also not accepted (at least 1 minute). To get paid, the participants also had to provide at least 30 labels per week and at least 90 labels for the whole time frame. Figure 8.3 provides an overview of the crowdsourcing study setup. During the 18-day field study, the 55 sampled participants used the data collection app to label their device usage. Technically, the app referenced each label (app feature) with the collected interaction data. Whenever the participants used an app on their smartphone, we showed a dialog in which the participants were able to select the app features they used, or, if not specified in the list add app features by themselves.

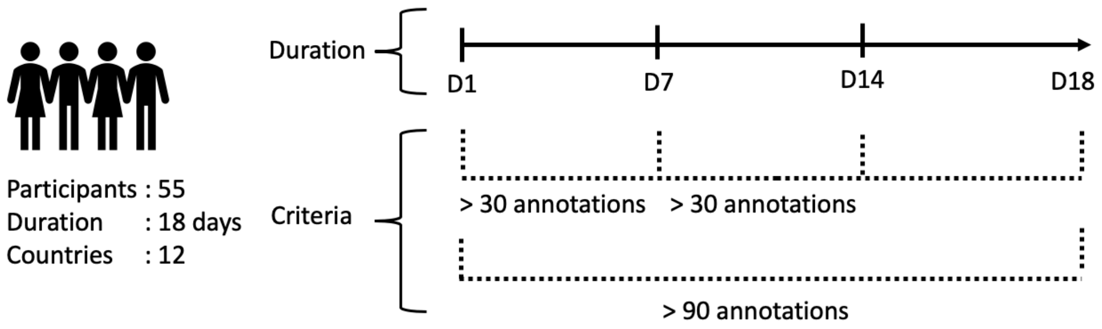


Figure 8.3: Design of the crowdsourcing study.

8.2.4 Study Data

The study resulted in a total of 55 GB of raw data. The participants used 170 unique apps and produced 27,569 sessions, of which 5,815 were labeled. We have about 4.7 times more sessions without labels, as we only allowed the participants to perform a certain amount of labels per day. We enforced this behavior as a quality criterion as the participants might otherwise be tempted to perform all required labels within one day—leading to many fake labels. On average, the participants used 6.85 apps during a single phone usage, which, on average, lasted about 7.6 minutes. The 170 apps in our data set cover 53 app categories, of which 29 belong to gaming subcategories.

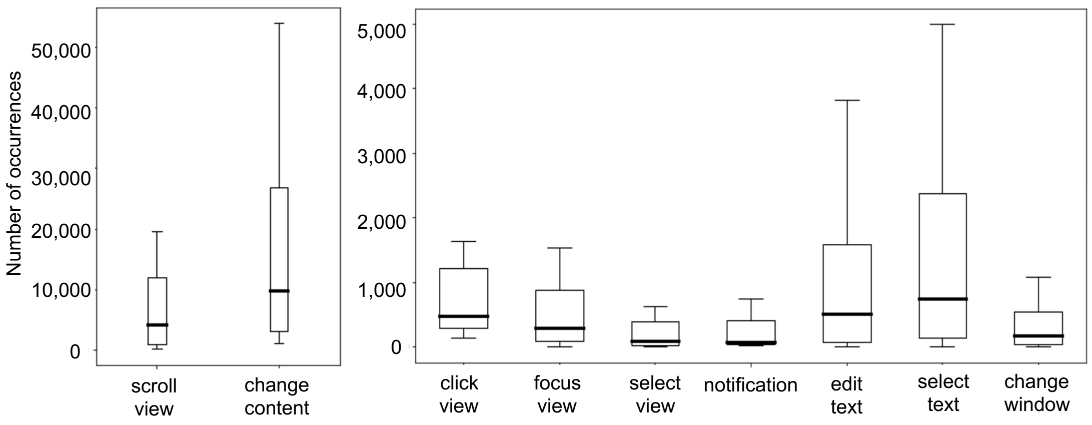


Figure 8.4: Interaction events per app in the study data.

Table 8.1: Mean number of interaction events per app session.

App Name	Click View	Notific.	Edit Text	Sel. Text	Scroll View	Ch. Content	Foc. View	Ch. Wind.	Sel. View
Facebook	9.34	4.51	99.09	57.76	306.11	408.80	9.64	29.66	13.33
Gmail	6.17	6.12	38.22	116.29	99.01	116.33	4.01	11.09	4.14
Fb Messenger	4.53	6.07	109.40	114.61	31.98	61.51	5.96	9.62	5.09
Snapchat	10.26	5.37	83.50	90.70	101.61	146.91	21.39	16.88	8.79
Google Drive	6.49	3.60	8.25	10.38	33.99	38.65	2.48	11.22	4.05
YouTube	6.37	14.97	38.70	44.51	74.67	96.19	8.90	13.38	7.63
Instagram	11.38	4.02	49.56	33.45	22.72	122.76	23.28	12.90	9.64
Twitter	9.44	3.19	93.96	53.09	94.77	248.91	7.04	18.49	18.88
Yahoo Mail	5.32	3.05	31.62	133.92	41.52	68.40	8.61	14.40	5.82

In total, we count 721 raw app features. As stakeholders describe similar app features differently (e.g., “take photo” and “capture photo”), and because we allowed the participants to add app features that were not covered by the label dialog the data set contains many noisy app features that are, for example, duplicates. After a manual analysis of the app features, we reduced the amount to 32 high-level app features.

Figure 8.4 depicts the number of interaction events per app. The figure shows that the most occurring interaction types are "scroll view" (median 4,119) and "change content" (median 9,754). The remaining interaction types are less frequent within the single apps. For example, the median number of "click view" events is 465.

Figure 8.5 shows the number of labels per app and high-level app feature over the whole dataset of 170 apps. Each dot on the y-axis represents one app feature for a certain app. Its position on the y-axis represents the number of labels we have for that certain app feature. The figure reveals that for the majority of apps, we have less than 50 labels per app feature. For nine apps, we have app features that have at least 50 labels. For instance, Gmail has 403 labels for "read email", 74 labels for "send email", 12 for "delete email", and 9 for "write email".

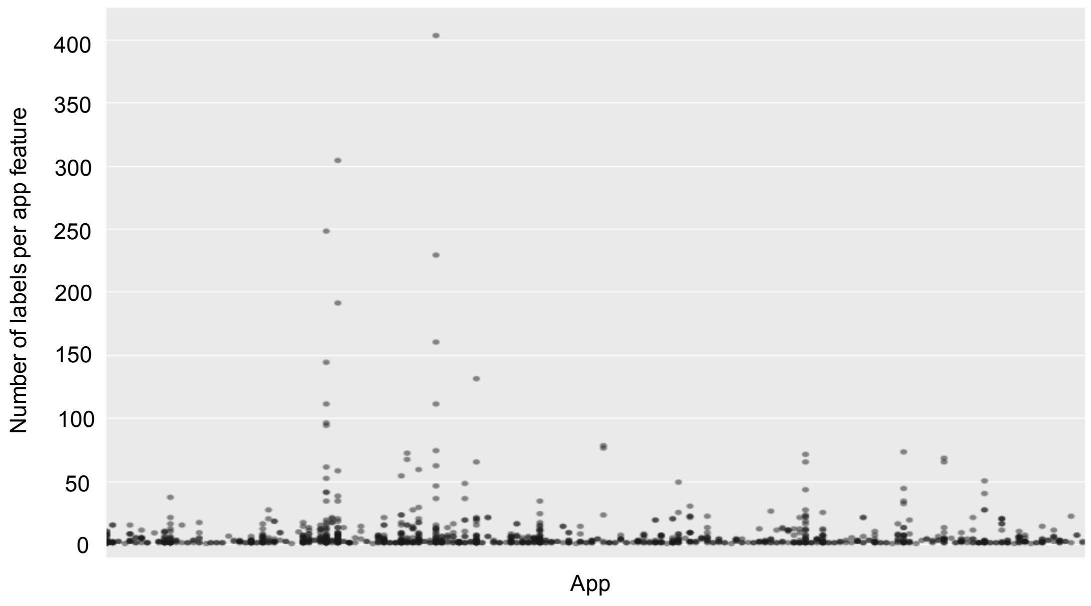


Figure 8.5: Frequency of app features labeled in the dataset.

8.2.5 Machine Learning Experiments

This section summarizes the use of the collected interaction data by explaining all technical steps, including the data pre-processing, feature engineering, and classification experiments.

Data Pre-Processing. One of the most extensive tasks in machine learning is the preparation of the input data before a machine learning model can learn from it. We performed several pre-processing steps for our supervised machine learning experiments. First, we removed the data of participants that did not match the quality criteria described in Section 8.2.3. Second, we merged the *scroll view* events because the accessibility framework fires this event every few milliseconds leading to dozens and sometimes hundreds of *scroll view* events, when, in fact, the participant only scrolled once. Third, we manually looked into the labels (app features) and found that we had to unify most of them because they were either (1) semantically identical (e.g., "capture photo", "take photo", "take picture", "make photo", "shot photo"), (2) duplication due to spelling mistakes, and (3) for some features we have too few, sometimes single labels which makes a classification approach impossible. To resolve the labeling issues, we manually iterated the list of all labels by resolving spelling mistakes grouping semantically similar features and extracting high-level features. As an example, "view picture", "view photo", and "view image" become the high-level feature "view photo".

Feature Engineering. Feature engineering is the process of using domain knowledge to represent data in a meaningful way for a specific machine learning task. In this paper, we classify which app feature a user uses during an app session. An app session comprises all interaction events that a user-triggered during the usage of a single app. For our classification experiments, we represent each app session as a machine-readable vector. The feature vector consists of nine numerical values based on the event types in Table 2.2. Each value describes its occurrence proportion of the whole app session. For example, if a participant uses an email client and triggers nine clicks and one scroll event during an app session, the machine learning feature value for *click view* is *0.9*, for *scroll view*, it is *0.1*, and *0.0* for all other machine learning features. As our machine learning features represent the proportion of each event type within an app session, all values are between 0 and 1. Therefore, we do not need to scale the feature values manually.

Classification Experiments. We describe how we created the classification experiments we conducted. For the within-apps app feature classification, we conducted experiments considering the four dimensions: (1) app, (2) app feature, (3) classifier, and (4) training data balancing strategy. The first dimension represents all 170 apps in our dataset. The second dimension concerns all high-level app features per app (varies per app). In the third dimension, we train seven different machine learning algorithms, which the Python sklearn library scikit-learn [194] supports: *LinearSVC*, *GaussianNB*, *DecisionTreeClassifier*, and the ensemble classifiers *AdaBoostClassifier*, *GradientBoostingClassifier*, *RandomForest*, and the *VotingClassifier* using soft voting. In the fourth dimension, we tested non-sampling vs. random undersampling before training the classifier, because a machine learning model tends to classify only the majority class when trained on a highly unbalanced training set. The combination of all four dimensions led to 844 experiments for the within-apps analysis.

For the between-apps analysis, we performed 448 experiments (32 high-level app features · 7 classifiers · 2 balancings). In total, we ran 1,292 experiments for this study. For each experiment, we performed 10-fold cross-validation and compared the results according to well-established metrics in related work—*precision*, *recall*, *F1*, and *ROC AUC*.

8.3 Results: Within-Apps Analysis

We report on the within-apps analysis, which automatically identifyies app feature usage for a particular app. We run our experiments on the top 30 apps (the most labeled) and report on the results in Table 8.2. Table 8.2 shows only the results of the nine apps for which we have enough labeled data to perform classification benchmarks for at least three app features. We decided for at least three app features because otherwise, we would not be able to show if we can distinguish between different features of an app. The table shows for each app the total number of app features it encompasses (*n_features*), the app features for which we have enough labels for performing 10-fold cross-validation, the best working machine learning algorithm (*estimator*), the evaluation metrics, as well as the number of labels used for training and evaluating the classifier. For example, Facebook counts a total number of 13 app features in our dataset, out of which we can identify the usage for seven. For all seven app features, balancing (*n_true* vs. *n_false*) the dataset leads to the best performing results. One additional observation is that having many labels does not necessarily lead to better

Table 8.2: Within-apps analysis: machine learning results for app feature usage identification.

App Name	n_features	App Feature	Estimator	Precision	Recall	F1	ROC AUC	n_true	n_false
Facebook	13	browse newsfeed	Gradient Boosting	0.85	0.92	0.88	0.85	12	12
		read post	Voting Classifier	0.63	0.88	0.74	0.68	49	49
		view event	Naive Bayes	0.55	0.83	0.66	0.53	109	109
		share media	Ada Boost	0.62	0.70	0.65	0.60	53	53
		get notification	Linear SVC	0.51	0.90	0.65	0.47	143	143
		like post	Linear SVC	0.49	0.89	0.63	0.46	94	94
		write comment	Random Forest	0.61	0.55	0.58	0.61	40	40
Fb Messenger	11	send message	Voting Classifier	0.79	1.00	0.88	0.61	252	69
		play game	Voting Classifier	0.75	0.86	0.80	0.71	14	14
		read message	Voting Classifier	0.62	0.88	0.73	0.55	17	17
		view media	Voting Classifier	0.64	0.69	0.67	0.65	36	36
Gmail	8	read mail	Linear SVC	0.68	1.00	0.81	0.51	354	170
		send mail	Ada Boost	0.65	0.65	0.65	0.66	69	69
		delete mail	Linear SVC	0.50	0.92	0.65	0.58	12	12
		search mail	Linear SVC	0.60	0.67	0.63	0.43	48	48
		get mails	Naive Bayes	0.50	0.81	0.62	0.54	160	160
Google Drive	4	edit file	Linear SVC	0.56	1.00	0.71	0.51	20	16
		view file	Linear SVC	0.47	0.90	0.62	0.54	10	10
		save file	Naive Bayes	0.39	1.00	0.56	0.83	7	29
Instagram	11	like photo	Voting Classifier	0.85	0.81	0.83	0.85	21	21
		watch story	Random Forest	0.65	0.68	0.67	0.71	19	19
		browse timeline	Linear SVC	0.51	0.94	0.66	0.45	33	33
Snapchat	8	send message	Naive Bayes	0.58	0.95	0.72	0.70	65	65
		read message	Naive Bayes	0.56	0.98	0.71	0.70	41	41
		save message	Ada Boost	0.67	0.73	0.70	0.65	11	11
		watch story	Linear SVC	0.53	0.75	0.62	0.37	12	12
		capture photo	Voting Classifier	0.73	0.50	0.59	0.64	22	22
Twitter	6	view feed	Voting Classifier	0.55	0.85	0.67	0.46	13	13
		read message/tweet	Decision Tree	0.65	0.66	0.65	0.65	73	73
		get live	Linear SVC	0.53	0.64	0.58	0.30	50	50
Yahoo Mail	4	send mail	Random Forest	0.94	1.00	0.97	0.97	16	31
		read mail	Decision Tree	0.86	0.90	0.88	0.88	20	20
		get notifications	Naive Bayes	0.65	1.00	0.79	0.60	11	11
YouTube	12	watch video	Linear SVC	0.55	1.00	0.71	0.47	131	108
		view latest videos	Naive Bayes	0.58	0.83	0.68	0.57	18	18
		browse recommendations	Naive Bayes	0.57	0.80	0.67	0.60	20	20
		rate video	Decision Tree	0.57	0.62	0.59	0.57	21	21
Mean				0.62	0.83	0.70	0.61	56.65	47.00

results; for example Facebook "browse newsfeed" vs Facebook "get notification". For the machine learning algorithms, it is more important if the usage between two app features is as diverse as possible. If a user is browsing through their Facebook "newsfeed", the interaction event *scroll view* may be more frequent and important than the interaction event *edit text* or any other.

One finding of our experiments is that using an ensemble classifier—such as *Random Forest*, which conducts several *Decision Trees* to conclude—does not necessarily lead to better results. For the app Gmail, the best performing (F1) app feature is "read mail". That app feature achieves an F1 score of .81 and is classified with *Linear SVC*. Generally speaking, in 15 out of 37 experiments, ensemble classifiers performed best. On average, we can identify app feature usage for 0.53 of all app features in Table 8.2.

However, the interpretation of the F1 score must be made with caution, as it might be misleading. Therefore, we also included the ROC AUC, which is a measure that compares our results to a baseline and helps to interpret how much better the models are compared to random models. In the following, we discuss three distinct cases that foster the interpretation of our results when considering F1 and ROC AUC.

Case 1: One example of a model with good classification results is Facebook with the app feature "browse newsfeed". It has a promising F1 score of 0.88, which is an indicator of a good performing model. The ROC AUC score of 0.85 is very similar and confirms that the model will achieve much better results than a random classifier (ROC AUC = 0.5).

Case 2: Gmail, on the other hand, has a misleading F1 score of 0.81 for the app feature "read mail". Its ROC AUC value of 0.51 reveals that the model is performing similar to a random model. When taking precision and recall into account, we find that the recall of 1.0 is the indicator for the misleading F1 score. A recall of 1.0 means that the model simply classifies all instances as "read mail" even though from the *n_false* columns, we can see that there are 170 labels for the negative case ("not reading mail").

Case 3: Twitter's app feature "get live" has a rather low F1 score of 0.58 (precision 0.53, recall 0.64). The ROC AUC value of 0.30 indicated that the model performs worse than a random classifier, which has a ROC AUC of 0.50. However, since we employed binary classifiers, a ROC AUC value below random indicates that the classification predictions could be inverted—resulting in a ROC AUC of 0.70.

As a consequence, we conclude that interpreting classification results should be done carefully by considering not only the F1 score but also the ROC AUC value.

Figure 8.6 shows how many of the nine apps in our experiment reach a certain threshold of the mean F1 and ROC AUC score across the app features we are able to classify. For example, it shows that for all nine apps, we reach the threshold of a mean F1 score of at least 0.60. In contrast, the mean ROC AUC values show that for three apps, we do not achieve a mean value of 0.60. For the three apps *Instagram*, *Fb Messenger*, and *Yahoo Mail*, the mean F1 scores of their app features are at least 0.70, while for *Yahoo Mail*, the mean F1 score of all its app features is at least 0.80. The colors on the heat map show for each app how

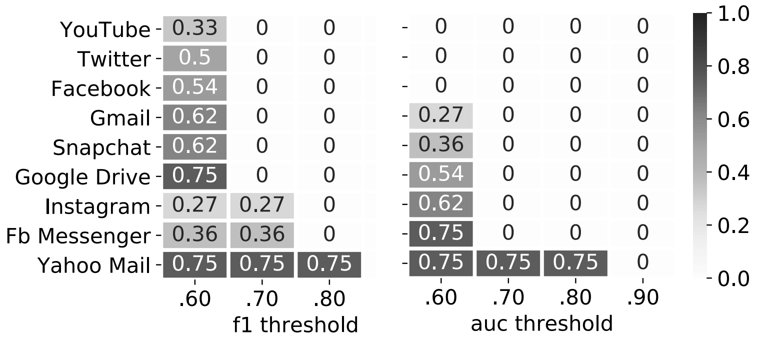


Figure 8.6: Within-apps analysis: the ratio of app features for which we reach a certain mean F1/ROC AUC threshold.

many of its app features we can successfully classify. For example, our models can successfully classify half of Twitter's app features with at least a mean F1 score of 0.60 (In Table 8.2 it shows that Twitter has six app features in our dataset out of which we can classify three). Figure 8.7 focuses on the max F1 and ROC AUC

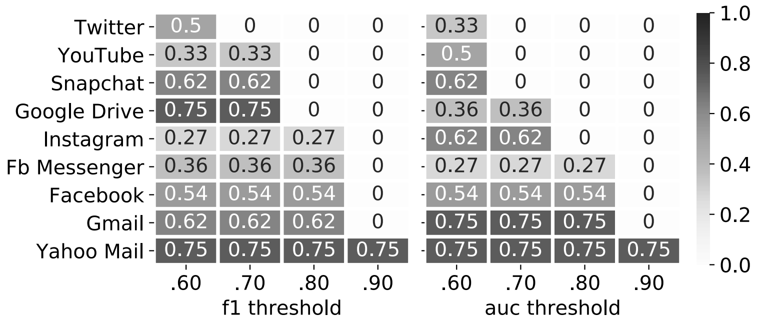


Figure 8.7: Within-apps analysis: ratio of app features for which we reach a certain max F1/ROC AUC threshold.

score we reach for classifying app features. The figure helps to better overview the best performing cases of Table 8.2. It reveals that for *Yahoo Mail*, we can achieve a max F1 score of above 0.90. Further, we can report that for five out of the nine apps, we have at least one app feature for which we achieve a high F1 score of 0.80. Similarly to the F1 score of *Yahoo Mail*, we achieve, for one app feature, a ROC AUC score of at least 0.90.

Table 8.3: Within-apps analysis: machine learning feature importance reporting χ^2 (chi-squared test).

App Name	App Feature	Click View	Notific.	Edit Text	Sel. Text	Scroll View	Ch. Content	Foc. View	Ch. Wind.	Sel. View
Facebook	share media	0.16	0.29	**2.17**	0.92	0.06	0.07	0.01	0.38	0.10
Gmail	send mail	0.38	**10.19**	0.01	8.49	0.00	0.27	0.24	0.48	0.14
Fb Messenger	send message	0.05	0.06	2.26	**2.49**	1.26	1.05	0.01	0.83	0.01
Snapchat	capture photo	0.20	0.12	0.28	0.22	0.17	0.00	0.49	**9.55**	0.05
Google Drive	edit file	0.03	**1.06**	0.02	0.02	0.02	0.02	0.01	0.23	0.00
YouTube	rate video	0.65	0.08	0.00	0.00	0.11	0.02	0.03	0.59	0.02
Instagram	send post	0.00	0.01	1.29	**1.32**	0.16	0.13	0.00	0.31	0.00
Twitter	read message	0.01	0.00	0.20	0.00	0.09	0.00	0.01	**1.35**	0.08
Yahoo Mail	send mail	0.15	0.14	0.14	1.35	0.13	0.10	0.05	**3.02**	0.02

Machine Learning Feature Significance

We analyze the machine learning feature significance quantitatively and qualitatively to gain insights into our trained machine learning models. Table 8.3 shows the significance (impact) of the machine learning features in our classification experiments based on the χ^2 scores [193]. The table lists the nine apps, which we also used for our classification experiments. The table shows the app feature that contains the highest χ^2 score. It is important to note that a high significance does not mean that this machine learning feature had the highest occurrence. Instead, it means that it is crucial information to classify the app feature. A high χ^2 score could also mean that a small value of a certain feature is informative.

Figure 8.8 shows the feature distribution of the most significant features. For example, the table reveals that the feature *edit text* is the most significant to classify the app feature "share media" in the Facebook app (2.17). We can see in the violin plot that users trigger the *edit text* event more often when they "share media". In the Gmail app, the feature *notification* reaches the highest χ^2 score. It is highly significant for classifying the app feature "send email" with a χ^2 score of *10.19*. This could be the case as the *notification* event is triggered more frequently through the toast message that indicates that the mail has been sent successfully. For the app Snapchat, the *change window* event is highly significant for classifying the "capture photo" feature. According to the violin plot, we can see that the change window event occurs more frequently when users capture a photo. We think that the switch between Snapchat and the camera app could cause this imbalanced number of "change window" events. We conclude that the significance scores and the distribution of the machine learning features can be aligned with an intuitive human understanding.

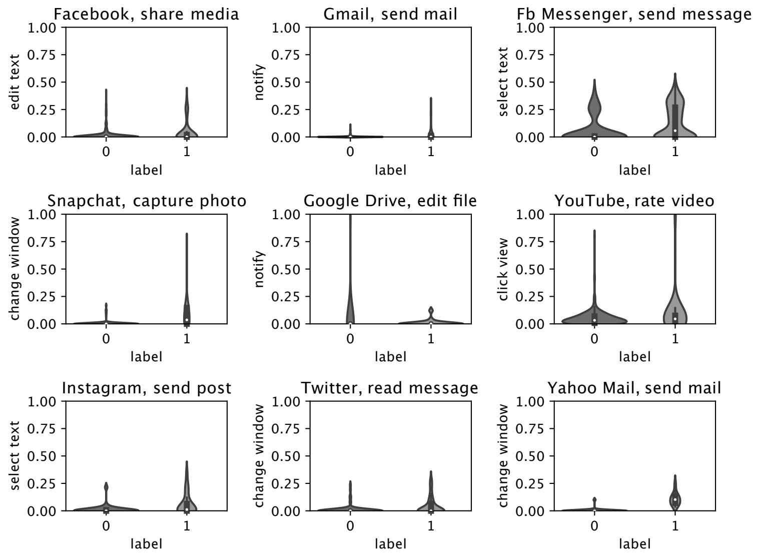


Figure 8.8: Within-apps analysis: violin plot for the apps with the most significant machine learning features.

8.4 Results: Between-Apps Analysis

Table 8.4: Between-apps analysis: the five top and worst machine learning results for app feature usage identification.

Estimator	App Feature	n_apps	Precision	Recall	F1	ROC AUC	n_true	n_false
Linear SVC	listen to music	4	0.82	0.91	0.86	0.91	44	44
Voting Classifier	delete	4	0.77	0.89	0.83	0.82	19	19
Voting Classifier	play game	10	0.75	0.75	0.75	0.80	85	85
Voting Classifier	earn money	14	0.73	0.76	0.75	0.80	83	83
Gradient Boosting	pay money	20	0.74	0.71	0.72	0.79	92	92
Naive Bayes	share	19	0.51	0.79	0.62	0.47	123	123
Gradient Boosting	search	37	0.62	0.61	0.62	0.65	308	308
Decision Tree	manage items	13	0.62	0.61	0.62	0.61	92	92
Naive Bayes	call	6	0.51	0.77	0.62	0.44	31	31
Decision Tree	write message	29	0.59	0.57	0.58	0.58	205	205
Mean		15.6	0.66	0.74	0.70	0.70	108.2	108.02

The goal of the between-apps analysis is to understand if we can re-use the labeled data, and therefore, the interaction data from multiple apps to identify the usage of similar app features. For example, Gmail and Yahoo Mail are both email clients that allow their users to draft, send, and receive emails. For example, to send an email, the user has either to write one or to select a draft, choose the recipients, define a subject, and eventually, click a send button. The combination of these steps describes the whole process to send an email, which may be similar for apps that offer the same functionality, as depicted in the example above. We run the between-apps analysis for all 32 high-level app features, which lead to 448 experiments as described in the experiment configuration.

Table 8.4 shows the results of the top-five and worst-five performing classification experiments. Our following analysis will focus on these ten examples of our results. In general, our classification results for the top-five app features look promising as we are able to achieve values above 0.70 for both F1 and ROC AUC, which is indicating a reliable classification model. The best performing app feature is "listen to music", which achieved a high F1 score of 0.86 and a high ROC AUC of 0.91 with only 44 true labels. In Table 8.4, we also report on the number of apps in which the listed features are labeled. This number is important because if e.g., there is only one app that implements a certain feature, we would report a within-apps analysis instead of between-apps. For example, for the app feature "listen to music" we count the four apps *Amazon Music*, *Pandora - Streaming Music, Radio & Podcasts*, *Spotify: Discover new music, podcasts, and songs*, and *Deezer Music Player: Songs, Playlists & Podcasts*. This shows that the model is able to learn the app feature identification by combining the interaction events of all four apps. In total, the mean number of apps that implement a certain feature is 15.6, while the mean for the top-five is 10.4 and 20.8 for the worst-five app feature classification experiments.

Machine Learning Feature Significance

We investigate the between-app feature classification also for the five best and five worst machine learning classification results, which we listed in Table 8.4. We highlight the most significant machine learning features for each app feature from Table 8.5. Figure 8.9 shows the distribution of the machine learning feature values in violin plots for each highlighted χ^2 score, which we interpret in the following. We can see that for the app feature "listen to music", the machine learning feature *select view* is strongly significant with the highest χ^2 score of 45.99. The plots reveal that the *select view* event occurs more frequently when users "listen to music", as they might select the next songs. The app feature "share" shows a high ratio of *click* events, as users have to select which content to share, via which app, and with whom they want to share it, which could be the reason for this high ratio. The app feature "search" shows a high ratio of *scroll view* events, which users might trigger by scrolling through the search result lists. Significant for the feature "earn money" is a high number of *click* events, which users might trigger by filling out surveys for which they get paid. When users "manage items", they trigger many *click* events, as they probably navigate through different menus to, e.g., edit their shopping cart or adjust their account settings. When users "call" on the phone, the ratio of *notification* is high, which could be caused by the fact that Android shows a notification for incoming calls. When users "pay money" or "write messages", the ratio of *edit text* events is often higher than usual. In the first case, users probably enter their payment details; in the second case, users write and edit the text message they want to send.

Table 8.5: Between-apps analysis: machine learning feature importance reporting χ^2 (chi-squared test).

App Feature	Click View	Notific.	Edit Text	Sel. Text	Scroll View	Ch. Content	Foc. View	Ch. Wind.	Sel. View
listen to music	0.28	0.55	1.52	1.69	4.18	0.42	0.15	0.17	**45.99**
delete	0.60	0.36	0.62	**0.67**	0.09	0.32	0.12	0.03	0.01
play game	0.08	0.82	1.57	1.96	2.03	0.68	0.20	0.93	**6.90**
earn money	**4.84**	0.06	1.52	1.44	0.38	0.05	3.13	0.12	4.44
pay money	3.41	0.12	**3.47**	2.69	3.05	0.36	0.01	1.64	0.00
share	**0.72**	0.11	0.12	0.49	0.14	0.01	0.01	0.62	0.19
search	1.87	0.22	3.71	2.19	**7.79**	0.05	0.00	1.08	0.26
manage items	**0.72**	0.00	0.58	0.26	0.05	0.00	0.14	0.02	0.46
call	0.00	**1.11**	0.62	0.00	0.65	0.03	0.22	0.36	0.26
write message	0.29	0.54	**6.10**	7.35	0.12	0.92	0.38	0.99	1.63

8.5 Discussion

We discuss the impact of our findings on the software engineering community, as well as the strengths and limitations of our approach and study.

8.5.1 Implications of the Results

Based on a survey with 607 responses from software engineers, Begel and Zimmermann reported [17] that the question "How do users typically use my application?" is the most essential and worthwhile. Our results show that we can learn app feature usages from high-level, app-independent interaction data: both by training classifiers with the data of other users of an app, or by using the usage data of different apps providing similar app features. In the following, we focus the

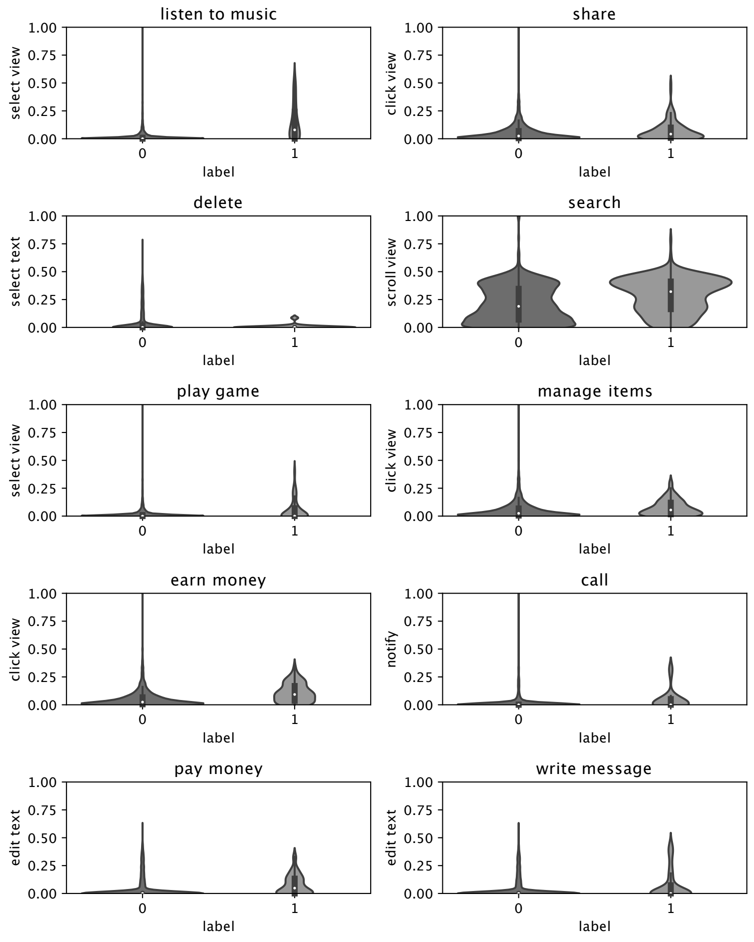


Figure 8.9: Between-apps analysis: violin plot for the app features with the most significant machine learning features.

discussion on how this can be used to support stakeholders, including developers, managers, and analysts. We summarize the following three main use cases for the discussion.

Objective app usage analytics. App analytics tools do a good job in reporting the app's "health" by analyzing the written reviews on either app stores or social media [151, 176, 256, 257]. These analyses usually classify the written reviews into bug reports, feature requests, or irrelevant [39, 81, 249]. They further show the opinion of users about certain aspects of the app, such as app features listed on app pages [79, 101]. However, these analyses only rely on the *subjective* experience of users, are non-representative (only those who submit feedback are represented), and often contain emotional and uninformative text [82, 187].

Based on our approach (analyzing the usage data), stakeholders can understand the actual *objective* usage of an app, independently of users' emotions. Usage data analytics (interaction data + app feature usage) enables answering questions such as: How long is my app or particular features of it being used? By whom? Which features are used together, and in what order? Depending on the collected usage data, more questions can be answered. If focusing on only interaction data + labeling, we can perform feature usage analytics. When adding additional context data such as location, device/hardware usage, and connectivity, we can answer questions such as: Where are our features being used? How much do they stress the hardware?. When also collecting data like views names and its content, we can answer advanced questions like: What views do our users stay longest and shortest? Where and at which point do my users close the app? The consequence of performing the suggested advanced analyses is to collect very pervasive and privacy intrusive data.

8.5.2 Field of Application

Data-driven release planning. Stakeholders usually keep track of open bugs, enhancement and feature requests in issue tracker systems. For release planning, stakeholders have to decide what to develop for which release – often a decision complex [36, 215]. Understanding popular, as well as unpopular app features, can support making this decision. For instance, bugs affecting popular app features might have a higher impact on users' satisfaction with the app [153]. Therefore, one recommendation to stakeholders could be to focus on bugs that affect popular app features as they would negatively impact the user more frequently. Similarly, feature popularity analysis can help prioritize testing, quality assurance, and documentation work.

Combining subjective with objective feedback. Software engineers describe their app features in the stores' app pages, which are also being addressed by users in their feedback and reviews [107, 118, 144]. A promising future area is the combination of both objective (what users do) and subjective (what users say) feedback. Written user reviews often report bugs but rarely state important information such as the affected device, software version, and steps to reproduce [22, 187]. Our work enables mapping written reviews with usage data to provide the missing information. For instance, spontaneous feedback mentioning a certain app feature can be used to label the interaction data and create the training set. On the other hand, if we precisely detect the feature the user has been using, we can, for instance, associate the star ratings (at least in part) to that feature.

This combination of information can help resolve bugs quickly and better understand the meaning of the written reviews. It may also give insights for rather uninformative reviews like "this app is trash!!" as the usage data could reveal pain points such as non-crashing bugs, views on which the user stayed much longer than others, or a view on which a user clicked several times on a button out of frustration.

8.5.3 Alternative Implementations from Related Work

Our study aimed at identifying app feature usage, which helps, e.g., in understanding how users use app features. However, we can implement the feedback to requirements activity for different ideas. In the following we summarize potential alternative implementations.

In the domain of mobile apps, one of the most important success factors is the user interface of the app. User interfaces designed in an intuitive fashion reduce the risk of frustrated users, who, in the end, might uninstall the app based on their negative experience. If we want to understand how users are using the UI in terms of navigation paths, time stayed on a view, or the number of performed actions, we can leverage user interaction data [55]. Deka, Huang, and Kumar [55] developed ERICA, an approach that mines user interaction data of mobile devices, which automatically detects, e.g., UI changes, records screenshots to create interaction traces. With these interaction traces, we can identify weaknesses in the UI like unintended navigation paths or gain insights into how to optimize the existing and intended navigation paths. Using interaction data for UI performance testing is helpful if we want to scale the tests. In particular, A/B testing, which aims at testing different versions with users, is time and cost-intensive as a substantial amount of users must be found to gain meaningful insights from the tests. With user interaction mining, we can scale this process to potentially all users of an app [54]. One of the next steps could be a data-driven design approach [53]. The data-driven design provides a database with example designs that helps decision-makers finding a design that we can use in our product. We can use the database to either identify best-practices or to get inspiration. With data-driven models, we can understand whether a design may be successful in achieving the specified goals [53]. In their paper, Deka et al. introduce an approach that leverages user interaction data and UI databases to enable applications like design search, UI layout generation, UI code generation, user interaction modeling, and user perception prediction. However, as Liu et al. [146] point out that these approaches are rather black-boxes that do not expose the design semantics of the UI. They, therefore, performed interactive coding to create a lexical database for 25 different types of UIs that allows us to understand what each element on the UI means. In combination with user interaction data, we can not only understand the semantic of the UI elements but also know how they are used. That kind of analysis provides a rich source of information in the requirements engineering process that we can use to test different UIs, optimize navigation flows, identify navigation flaws, and understand what the user is doing in the app.

8.5.4 Limitations and Threats to Validity

We employed supervised machine learning for identifying app feature usage, which depends on labeled data for training. Machine learning can only be as good as these labels—wrong labels, e.g., fake or accidentally wrongly selected app features in the label dialog introduce bias and noise to the algorithm. We mitigated this threat by 1) carefully selecting the study participants in a pre-study and 2) by limiting the number of labels participants can assign during the field study (see Section 8.2.3). We have confidence in the collected data because of these measures and our data, which includes many popular apps with reasonable labels.

The generalizability of this approach is another threat to the validity of our results as we rely on data from 55 participants. However, we chose to perform crowdsourcing instead of, e.g., student-based experiments to ensure real mobile phone usage and selected participants from 12 countries to increase participants' diversity.

Since app features are named in diverse ways, even though they are semantically or even technically the same, we rely on unifying these app features manually. This manual process is prone to bias as its correctness depends on our understanding of the app features. We address this potential bias with two steps. First, one of the authors with more than eight years of experience in app development performed and documented the first iteration of the unification. Second, two more authors checked the documented list of the unified app features and discussed the disagreements leading to the final list.

8.6 Summary

This chapter introduced an approach that learns app feature usage from interaction events such as scrolling and clicks. We performed an 18-day long data collection crowdsourcing study with a focus on collecting high-quality data. The study had 55 participants who labeled 5,815 app feature usages for 170 mobile apps. In total, we performed 1,292 machine learning experiments

Therefore, we created an approach that can address the stakeholder need to understand which interactions led to the particular problems users reported. We summarize our main findings.

We can learn app features with few interaction events. Our within-apps experiments show that we can identify app features with encouraging F1 and ROC AUC of ~0.60-0.70 already only with about ten labeled feature usages. In our experiments, we found that more labeled data does not necessarily increase the classification result.

Apps require similar interactions to use their features. The between-apps analysis achieved promising F1, and ROC AUC scores of up to 0.91, which shows that for some apps, interaction events are similar between-apps when performing the same feature. As a consequence, we may be able to reuse labeled data for similar apps, which reduces the overall amount of needed labels.

We can reach explainable conclusions with a few human-readable machine learning features. We analyzed the machine learning feature significance and found that a human interpretation of interaction events is often similar to what the classifier model seems to find important. For example, when writing an email, the *edit text* event is the most important machine learning feature.

Part III

Evaluation

Chapter 9

feed.ai—A Prototype for Requirements Intelligence

I enjoy thinking about ways to create something that other people have not even thought about, something no one has managed to achieve.

Shigeru Miyamoto

Publication. This chapter significantly extends our publication "Requirements Intelligence with OpenReq Analytics" [236], which won the best *poster and tool demo paper* award at the 27th IEEE International Requirements Engineering Conference. My contributions to this publication were conducting the research, developing the tool, and leading the writing.

Contribution. This chapter concerns the development of a requirements intelligence prototype, including its integrated interactive visualization of the requirements intelligence framework (see Figure 4.1). We contribute by stating the requirements for the prototype that we extracted from our two qualitative studies in Chapter 3. We further detail our technical decisions like the microservice architecture of the prototype and provide the dynamic models for some of the microservices. Eventually, we introduce the integrated interactive visualization by depicting screenshots from the prototype and by relating its features to stakeholder needs.

Addressed stakeholder needs. In Chapter 3, we found that stakeholders need automated tool support as the manual analysis of explicit and implicit user feedback is unfeasible. We also found that they do not know about automated tools that provide user feedback analytics, but they use tools to collect feedback or document it in, e.g., issue trackers. This chapter covers many of the stakeholder needs, which we aggregate and explain in the prototype requirements section.

9.1 Motivation

In Chapter 3, we analyzed the need for software requirements intelligence from the perspective of stakeholders. Stakeholders, as of today, feel overwhelmed when analyzing explicit and implicit user feedback for several reasons. For instance, stakeholders need much effort to understand their users because they receive feedback in large amounts from different feedback channels. As research shows, popular apps receive 4,000 reviews daily on app distribution platforms and about 31,000 daily feedback on Twitter [99, 187]. Analyzing that amount of user feedback is unfeasible, as the stakeholders in our study stated. In Villarroel et al. [249], a stakeholder stated that he worked two full days to analyze 1,000 app reviews, while a second stated that he dedicates 6-7 hours every week to analyze user feedback. Overall, the stakeholders in Chapter 3 desire tool support for automated analyses of user feedback.

We introduce feed.ai (user feedback analytics with artificial intelligence), a prototype demonstrating the software requirements intelligence framework. Here, we discuss the technical perspectives, such as the underlying architecture and the frontend of feed.ai. In Section 9.2, we list the requirements for the prototype, based on our findings from the interview study and literature review from Chapter 3. After that, we present the architecture of the prototype in Section 9.3 by starting with a general overview, followed by detailed sequence diagrams for some microservices. Next, we illustrate and discuss the integrated interactive visualization of the prototype by showing and explaining screenshots from the deployed version in Section 9.4. Afterwards, we discuss the implementation of the requirements intelligence framework in Section 9.5. Eventually, we summarize the chapter in Section 9.6.

9.2 Requirements

This section summarizes the requirements for feed.ai, which we based on our results from Chapter 3. To report the functional requirements, we chose *use cases*. Use cases are written text helping to discover and to record requirements [135]. In Table 9.1, we show an overview of all considered use cases. We grouped

Table 9.1: feed.ai use case overview (grey rows mark sub-systems).

ID	Name
—	Feedback source configuration
UC1.1	Specify feedback source
UC1.2	Specify feedback language
UC1.3	Specify feedback collection intervall
—	Performance indicators (frequencies)
UC2.1	See when users provide feedback
UC2.2	See trends in feedback
UC2.3	See the most urgent issues
UC2.4	See requirements-relevant feedback category distribution
—	In-depth feedback analysis (categories)
UC3.1	Read original user feedback
UC3.2	Read filtered feedback by feedback category
UC3.3	Read feedback addressing a certain feature or topic
UC3.4	Read context and interaction data data attached to feedback
—	Human control mechanisms
UC4.1	Evaluate automated filtering approaches
—	Comparison of multiple apps
UC5.1	Compare feedback category distribution
UC5.2	Compare sentiment
UC5.3	Compare trends in feedback

the use cases into the following five major categories/sub-systems.

Feedback source configuration

Chapter 3 revealed that only a few organizations collect and analyze implicit user feedback. However, all included stakeholders either try to utilize or want to analyze explicit user feedback. As some stakeholders stated opportunities in combing both feedback types, the tool should work with either one or both feedback types. When considering the individual feedback types, not every organization receives feedback from the same channels or devices. An app development organization might be interested in app reviews for explicit user feedback and app interaction data from smartphones, while a telecommunication organization might be solely interested in explicit user feedback from social media. As a consequence, the tool must be capable of handling both feedback types and multiple channels. We formulate the following use cases for the configuration of the feedback sources.

UC1.1 - Specify feedback source. Stakeholders need the ability to specify the feedback source to allow customized analysis results. In particular, they have to configure the explicit and implicit user feedback source. For example, stakeholders who want to analyze tweets must specify the Twitter profiles they are interested in analyzing.

UC1.2 - Specify feedback language. One challenge for global organizations is that they have users from across the world that speak different languages. Therefore, for explicit user feedback, stakeholders must specify the feedback languages they want to analyze.

UC1.3 - Specify feedback collection interval. Automated approaches need computation power and time for performing the analyses. In particular, machine learning intense systems need time to generate their results. Further, API limitations may restrict how often stakeholders can collect feedback. Therefore, stakeholders should be able to specify the interval for collecting and analyzing user feedback.

Performance indicators (frequencies)

The idea is to get a quick overview of predefined metrics, which stakeholders consider essential for the decision-making process. These metrics are individual for the organization using the tool but mostly report descriptive analytics. For example, if we consider the example of an app development organization, that organization is likely interested in seeing how users generally think about the app (for example, their sentiment), or the number of reported bugs since yesterday or last week. These metrics are an indicator for stakeholders to learn if they have to take immediate action. We formulate the following use cases for the overall performance indicators.

UC2.1 - See when users provide feedback. One challenge of stakeholders is to gather feedback from multiple platforms and channels. They try to support their users independent of the feedback channel they use. However, stakeholders do not know when users provide how much feedback. That knowledge can help to get time-related insights like when do users submit problem reports. Also, they can use time-related analysis for planning when they have to provide more support given they are interested in a quick resolution of user needs.

UC2.2 - See trends in feedback. One indicator of the product's performance is trend analysis. Trend analysis reports on a specified metric over time. For example, a simple trend can be the number of user feedback received since the last release. This information helps stakeholders in, e.g., understanding if they improve over time. They can perform trend analyses on either historical data only or empower it to predict future trends. In this requirement, the tool shall be able to present trends for problem reports, inquiries (total number of user feedback belonging to this category), and the overall sentiment (average sentiment). The trend shall cover the time frames since yesterday, since last week, and since last month.

UC2.3 - See the most urgent issues. Although user feedback is individual to the person writing it, there may be a set of users stating similar opinions or writing about the same topic. Therefore, it is important to show the most discussed topics within the requirements-relevant user feedback. If there is an unusual number of users reporting the same problem, the tool should highlight this issue to create attention toward the problem.

UC2.4 - See requirements-relevant feedback category distribution. As organizations may receive thousands of user feedback daily, they are interested in understanding how many are relevant and worth investigating further. However, there are different types of relevant user feedback. In this work, we focus on problem reports and inquiries as relevant user feedback. The tool shall show the number of received relevant user feedback for each feedback category.

In-depth feedback analysis (categories)

Our app review analytics mockup in Chapter 3 included a view that let stakeholders browse through actual user feedback. The interviews showed that stakeholders perceive this feature as helpful, specifically if the automated approach filters irrelevant feedback. In particular, developers found the list of user feedback helpful because users sometimes state information that allows the developers to isolate the location of an issue. Further, some users include context information, such as the software and hardware version, in their feedback. Therefore, the tool shall include a view that allows browsing and discovering requirements-relevant user feedback. We formulate the following use cases for the in-depth feedback analysis.

UC3.1 - Read original user feedback. Although stakeholders desire automated analyses of user feedback, they still need to read the original feedback. The reason is that analytics results only show part of the feedback like often addressed features and their count. But this kind of analysis results might miss important information. Therefore, the prototype shall display the original (raw) user feedback.

UC3.2 - Read filtered feedback by feedback category. For historical and potentially real-time analyses, the prototype should allow to monitor and analyze user feedback continuously. For example, agile software development projects need to get current user feedback to better support their tight release cycles as the feedback can either improve the release or help to understand the perception of the user of the released version. Generally, in software projects, continuously analyzing user feedback can help to identify and solve issues quicker [107]. Nonetheless, user feedback comes in large amounts. Therefore, the prototype shall provide a view only showing requirements-relevant (filtered) user feedback.

UC3.3 - Read feedback addressing a certain feature or topic. Filtering irrelevant user feedback helps stakeholders to concentrate on what they deem relevant. However, the relevant user feedback may still be too cumbersome to analyze as there is either too much or too long feedback to summarize it manually. Also, manually summarizing feedback is not an easy task as, during the reading process, the stakeholder has to identify common topics or software features addressed. For example, the tool identified 1,000 problem reports for the current day. Although all of them are problem reports, the actual problems of the user can be different. In this example, only one problem report is about the registration form in an app, 200 discuss issues with network features, 600 describe a crash on a specific view, and so on. This summarization helps to identify the most urgent issues without reading through all of the user feedback. Therefore, the prototype should be able to highlight the most discussed topics and features.

UC3.4 - Read context and interaction data attached to feedback. Chapter 3 shows that none of the 90 stakeholders solely use implicit user feedback, but all of them either consider explicit user feedback as useful or already analyze it. However, stakeholders find implicit user feedback as a rich source for information if attached to explicit user feedback. For example, if a user reports a problem with the software, the user rarely states technical information such as context data [157], which makes it hard for stakeholders to understand how the issue occurred. If we attach implicit feedback like the software version or interaction events, the automated approach can help to narrow down the location of the issue. This kind of information can help stakeholders in finding and solving issues quicker. Consequently, the prototype shall be able to display implicit user feedback attached to explicit user feedback.

Human control mechanisms

Machine learning algorithms are often a black box, and their predictions are often not traceable or human-understandable. The result is that stakeholders need to trust the output of the deployed machine learning models. However, machine learning models do not achieve 100% accuracy, which leaves space for wrong classification results. Therefore, stakeholders cannot trust these models completely and request control mechanisms that allow experts to check, and if needed, correct the machine learning model output. Another issue of, in particular, text-based machine learning models is that their performance may decay over time as the language of people or the product and service portfolio of an organization changes. These changes may introduce sentence structures, phrases, or words unknown to the machine learning model, which can result in a reduced accuracy over time. We formulate the following use case for the human control mechanism.

UC4.1 - Evaluate automated filtering approaches. Stakeholders do not completely trust automated approaches as they make errors or are generally not aligned with the stakeholders' opinions. Therefore, the prototype shall allow stakeholders to evaluate the machine learning models to improve them continuously.

Comparison of multiple apps

The use cases so far focussed on an introspective—the analysis of the products and services owned by the organization—but they do not consider potential competitors in the market. Our interviews in Chapter 3 revealed that stakeholders have to explore how their competitors perform in terms of the number of relevant feedback received, problems they have, and features requests their users have. The information gained from that analysis may be helpful to, e.g., find often requested features that could be part of the own product or to learn from their mistakes. They further want to understand how their own apps compare to each other. We formulate the following use cases for the comparison of multiple apps.

UC5.1 - Compare feedback category distribution. Our interviews in Chapter 3 showed that stakeholders, in particular, project managers, were interested in seeing how their own apps compare in terms of the number of problem reports and inquiries. The review of the literature further showed that stakeholders are also interested in understanding how their competitors perform. Therefore, the prototype shall compare the feedback category distribution of multiple apps.

UC5.2 - Compare sentiment. Similar to the distribution of feedback categories, stakeholders want to see how their users generally perceive the apps. With that information, they can answer questions like "are our customers happier than those of our competitors?". Therefore, the prototype shall compare the sentiment of multiple apps.

UC5.3 - Compare trends in feedback. Besides seeing the distribution of the feedback categories, stakeholders are interested in seeing how these categories developed over time. For example, they can get time-related insights like identifying negative or positive trends in each of the feedback categories. They can get these insights either for the feedback categories or for certain topics. Therefore, the prototype shall compare the trends in user feedback.

9.3 Architecture

This section describes the architecture of the requirements intelligence prototype feed.ai.

Overall, we decided in favor of a microservice-based architecture. As described on the Amazon technical blog post [7], microservices offer many benefits over monoliths. Microservices are autonomous, meaning that we can develop, deploy, operate, and scale them individually without affecting the other services. Further, microservices focus on solving one specific problem, which ensures that they have low complexity and are, therefore, easier to maintain.

We split the remaining architecture section into two parts. In the first part, we discuss the overall architecture and presenting an overview figure that highlights the communication flow of the microservices. The figure also marks some microservices which we describe in more detail in the second part of this section.

9.3.1 Overview

We group the requirements intelligence microservices into the following five conceptual layers:

Application Layer (AL). This is the topmost layer, and its purpose is to provide the graphical interface to the stakeholders using the prototype. feed.ai is a web-based application that is accessible from any browser and responsive for most of its features, which enables stakeholders also to use the prototype on mobile devices. The integrated interactive visualization of the requirements intelligence prototype is part of this layer.

Data Orchestration Layer (DOL). All microservices are accessible via well-defined APIs. The microservices in this layer are responsible for managing the communication flow between the microservices. The orchestrators are aware of

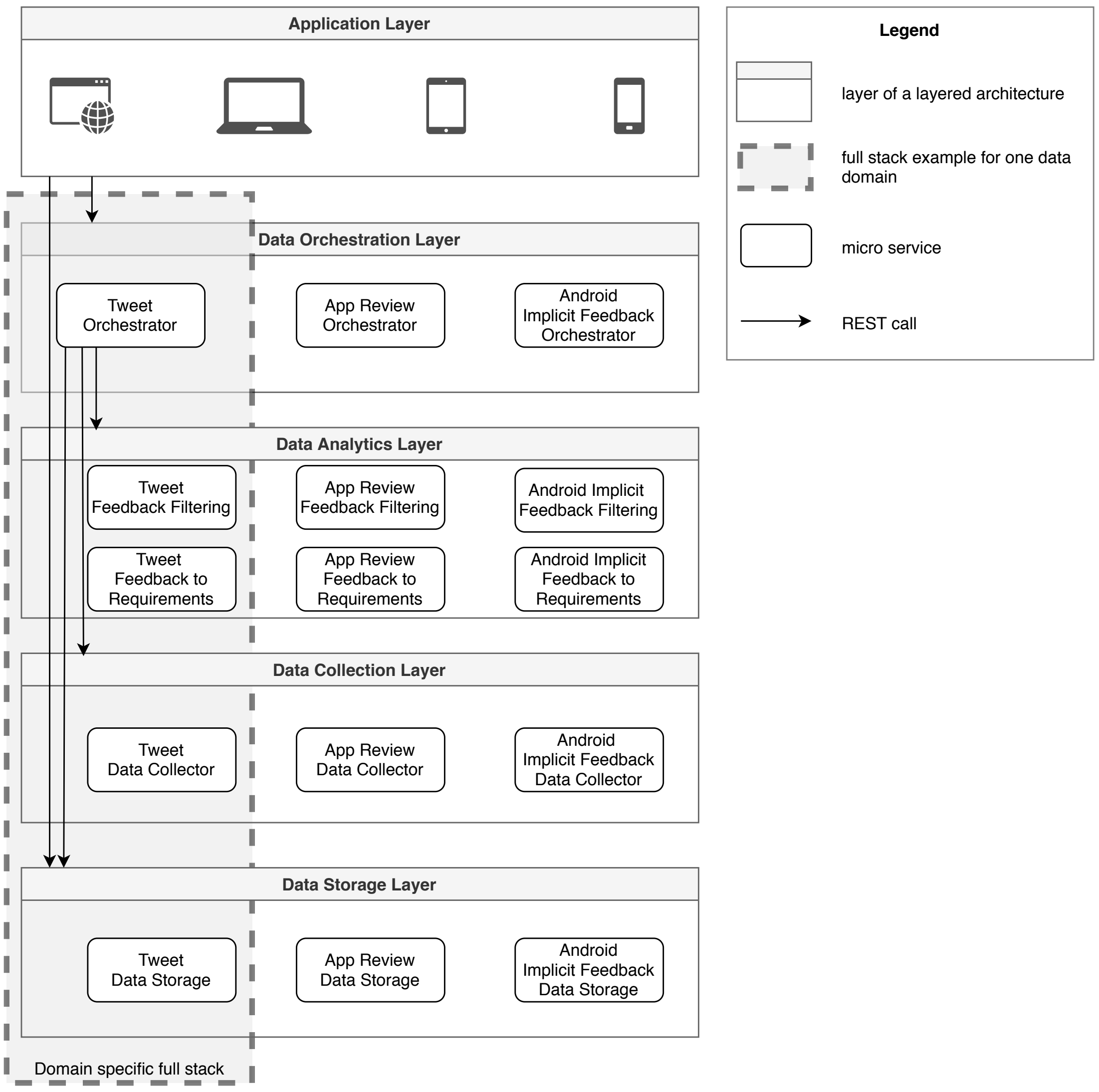

Application Layer
Legend
layer of a layered architecture
full stack example for one data domain
micro service
REST call
Data Orchestration Layer
Tweet Orchestrator
App Review Orchestrator
Android Implicit Feedback Orchestrator
Data Analytics Layer
Tweet Feedback Filtering
App Review Feedback Filtering
Android Implicit Feedback Filtering
Tweet Feedback to Requirements
App Review Feedback to Requirements
Android Implicit Feedback to Requirements
Data Collection Layer
Tweet Data Collector
App Review Data Collector
Android Implicit Feedback Data Collector
Data Storage Layer
Tweet Data Storage
App Review Data Storage
Android Implicit Feedback Data Storage
Domain specific full stack

Figure 9.1: feed.ai–architecture overview.

the other services and define the order, in which they have to call them. As per definition, we can use all microservices as standalone services in case third parties are interested in using only particular functionality, such as the app review filtering classification. However, feed.ai is a prototype showcasing the combination of all developed microservices and, therefore, needs orchestration for managing the communication flow. We designed the architecture in a way that there is one responsible orchestration microservice for each platform, i.e., Twitter and app stores. The idea is to reduce the framework's complexity by introducing conceptual groups of microservices that belong together.

Data Analytics Layer (DAL). This layer is a collection of microservices that processes raw user feedback. As for the microservices of the *DOL*, the *DAL* layer includes microservices dedicated to each single data domain. Despite having microservices for each domain, we further created a microservice per requirements intelligence activity. Therefore, the *DAL* contains microservices for filtering user feedback and for transforming feedback to requirements (e.g., extracting topics) per domain.

Data Collection Layer (DCL). The goal of this layer is to collect explicit and implicit user feedback. We collect explicit feedback with crawlers while monitoring implicit feedback from within the software, such as sensors (see Section 2.3). We have to develop custom data collection microservices and libraries for each data source and each type of user feedback.

Data Storage Layer (DSL). The goal of this layer is to persist data. For that, we store data not only coming from the collection layer but also store it for each processing step to avoid a potential loss of information. As explained in the *DAL*, we split the microservices based on their domain. For that reason, Figure 2 shows three microservices.

Figure 9.1 depicts one full-stack example for a single domain—highlighted by the box with the dashed line. Again, the example highlights that we designed the architecture in a way that we group the microservices per data domain. Within each domain and layer, there can be several microservices. For example, the analytics layer may perform several different machine learning approaches to generate insights from explicit user feedback. Following the idea behind microservices, there is one dedicated microservice for each machine learning approach. By following this procedure, the microservices are easier to maintain as the technical processes are different for each data source and feedback type (see Chapter 5). Further, we can support horizontal and vertical *scaling*, which is vital because by experience computation-heavy machine learning intense microservices need to scale earlier than, e.g., data storage based microservices.

9.3.2 Dynamic Models

In this section, we continue the example depicted in Figure 9.1 and present the sequence diagrams of the microservices for the Twitter domain. Here, we focus on briefly summarizing the microservices and their sequence diagram. A complete description of all dynamic models [64] will be published by the European Commission as part of the European Horizon 2020 project OpenReq [182]. The project further provides a detailed description of the APIs [181]. We show the sequence diagrams in the same order, as illustrated in Figure 9.1 (from top to down).

Overall Communication Flow

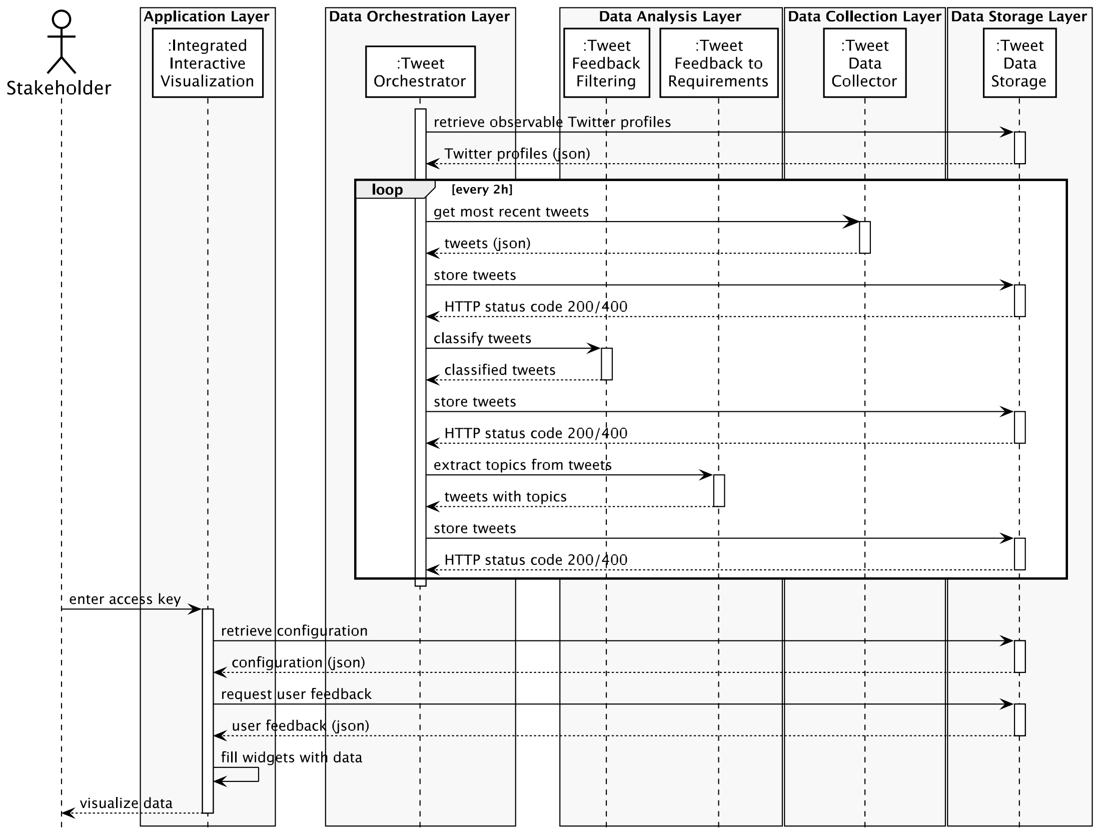


Figure 9.2: feed.ai–overall microservice communication flow.

Figure 9.2 summarizes the overall communication flow of the microservices for the Twitter domain. It highlights that the *Tweet Orchestrator* is the main responsible microservice for managing the communication between all Twitter-related microservices. Starting the *Tweet Orchestrator* initiates requesting all configured Twitter profiles that we want to analyze continuously. The next step sets up a cron job for each observable Twitter profile that triggers every two hours. Stakeholders can configure the observation interval for each Twitter profile individually so that the orchestrator asks the data collection microservice to crawl popular Twitter profiles more often. For describing this example, we decided on a two-hour interval cron job, which triggers the following process.

1. The *Tweet Orchestrator* requests a list of newly posted tweets from the *Tweet Data Collector*.
2. Store the retrieved tweets in a database by sending them to the *Tweet Data Storage* microservice.
3. Classify (filter) tweets to either *problem report*, *inquiry*, or *irrelevant*.
4. Store the filtered tweets.
5. Extract concrete topics and features such as "network".
6. Store the extracted topics.

This process happens in a continuous background task and does not affect the performance of the prototype's dashboard because it only accesses the processed data from the database and does not rely on, e.g., synchronous machine learning tasks.

Additionally, the bottom of Figure 9.2 highlights that stakeholders can access the dashboard only with a valid access key. If the stakeholder enters a valid access key, the dashboard requests a configuration file that includes the data sources the prototype analyzes and visualizes.

Tweet Data Orchestrator

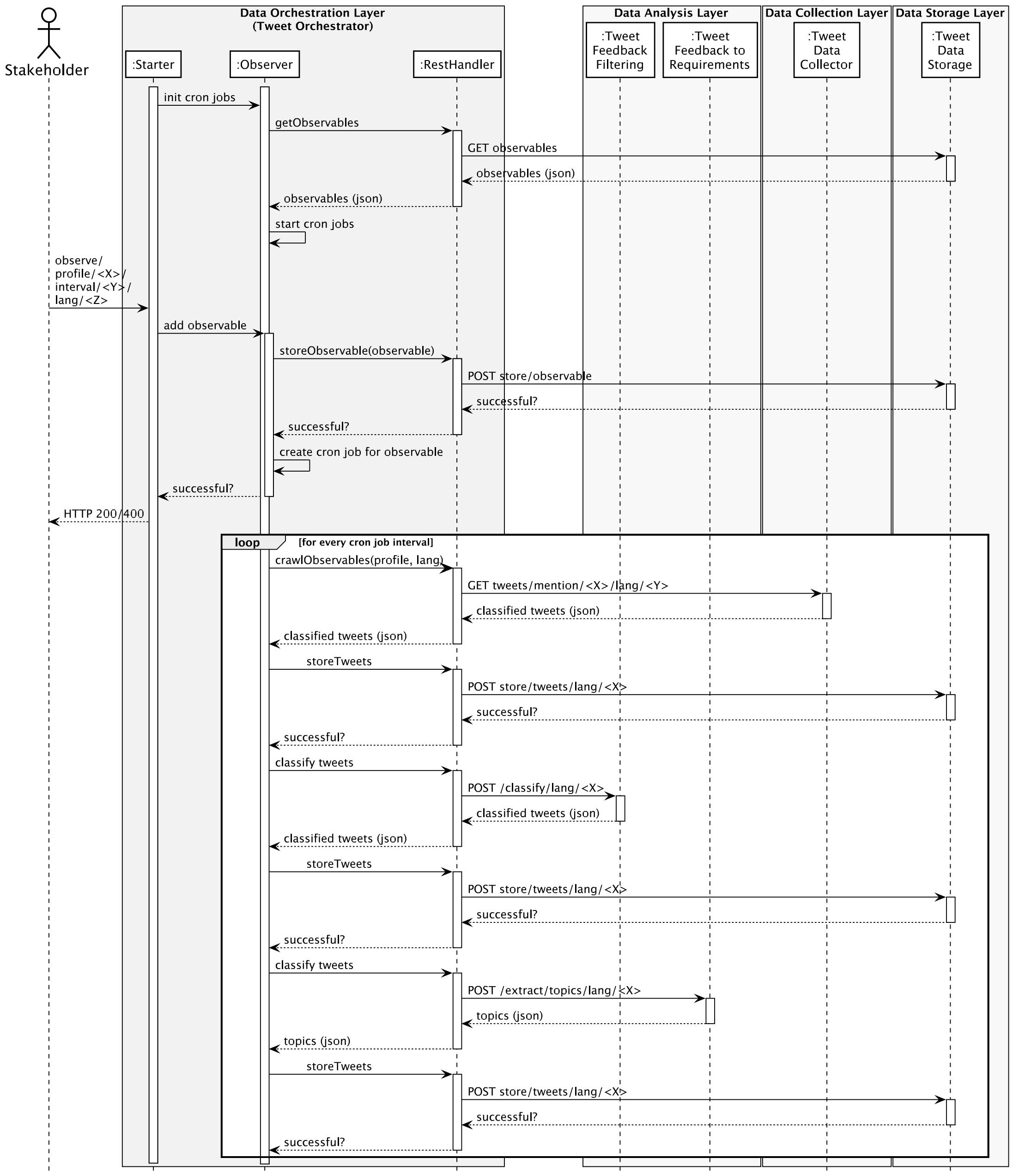


Figure 9.3: Tweet orchestrator communication flow.

The *Tweet Orchestrator* is the microservice responsible for managing the communication between other microservices. Though, each microservice exposes its own API, and is therefore accessible as a standalone service, the idea of feed.ai is to bundle all microservice. As we bundle the microservices and make them accessible on a dashboard, we reduce the effort for stakeholders to interact with only one point of access. As Figure 9.3 shows, the orchestrator has two main communication flows. On top of the figure, we visualize the initialization of the microservice, which triggers automatically upon startup. The initialization process first loads all observables, which in this exemplary case are all configured Twitter profiles. For each observable, the *Observer* class will create a cron job based on the configuration of the observable (target Twitter profile, the language of the feedback, and interval). Whenever a cron job triggers, it performs the same steps as described in the *Overall Communication Flow.* To summarize, the process crawls tweets from the configured Twitter profiles, then classifies them into *problem reports*, *inquiries*, and *irrelevant*, and finally, extract the tweet topics. There is an intermediate step that stores the result of each step. We decided to store the results of each step to avoid losing data in case any microservice is unreachable. At midnight, the microservice checks if there are tweets that did not run through the full process and performs all missing steps if needed.

Additionally, the orchestrator provides endpoints for adding and removing observables to cover the stakeholders' changing needs. Figure 9.3 depicts the process of adding a new observable. If we add a new observable, the microservice stores it in the database first. Afterwards, if storing the observable was successful, the service creates a new cron job based on the configuration in the initial request. The added observable is then automatically part of the loop in Figure 9.3; therefore, it will undergo the same procedure to get and process feedback sent towards that observable.

Tweet feedback Filtering

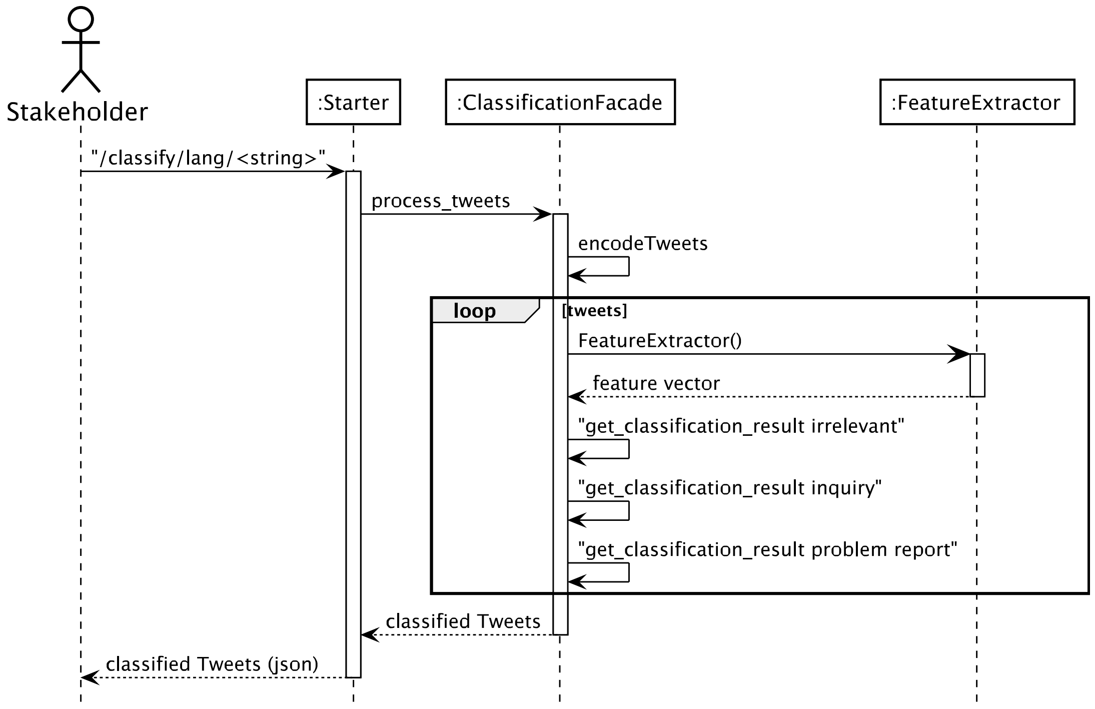


Figure 9.4: Tweet feedback filtering communication flow.

Figure 9.4 depicts the sequence diagram for applying machine learning on tweets to extract requirements related information, i.e., bug reports, inquiries, and irrelevant tweets. The microservice expects a list of tweets as a payload containing at least the tweet text body, as this is the input for the machine learning model. After ensuring that the text is in the UTF-8 format, the microservice iterates over each tweet and classifies it. For each tweet, the first step is to create a machine learning feature vector. Then the microservice processes it with one and up to three binary classifiers. As the binary classifier for identifying irrelevant tweets has much higher accuracy than the other classifiers, we perform this classification first. In case the model classifies the tweet as irrelevant, and the classifier has a confidence score of above 75%, the code immediately returns that classification result for this tweet. Otherwise, we insert the feature vector of this tweet to the problem report binary classifier and evaluate its result based on the same metrics mentioned above. Again, if the irrelevant and the problem report classifier both output that the currently processed tweets do not belong to these classes, the code employs the inquiry classifier. However, if the inquiry classifier returns that the input tweet is not an inquiry, we assign the irrelevant class as the default. We decided on this approach to avoid misclassifications based on the lower accuracy of the problem report and inquiry classifiers and to increase computation performance by performing fewer classification tasks.

Tweet Feedback to Requirements

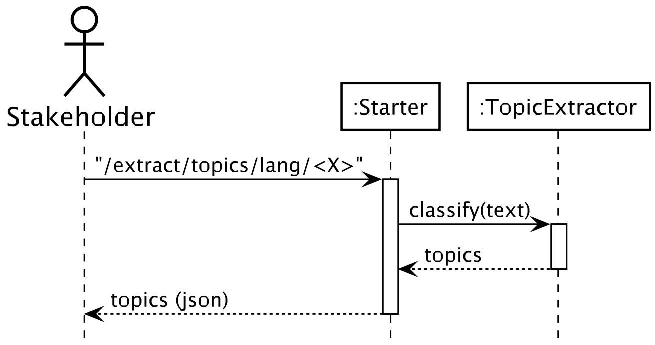


Figure 9.5: Tweet feedback to requirements communication flow.

Tweets may come in large amounts, which makes it difficult to analyze and understand them manually. This microservice aims at identifying topics in tweets by employing a supervised machine learning model. Therefore, we perform the requirements refinement step of the requirements intelligence framework in this service. For identifying topics, we provide the service with requirements-related tweets. As this service employs supervised machine learning, it only supports a fixed set of topics. Currently, the service is available for Italian tweets and supports the topics "billing", "churn", "network", and "offer". The machine learning model handles tweets as a multi-label problem, which means that a tweet can belong to multiple topics. For example, the tweet: “@Profile I cannot connect to the network anymore. This is a frustrating issue I face for several months. This is not what the contract and my bill state, and I want to cancel it!!!” may belong to the topics “network” and “billing”. The endpoint of this microservice expects a single tweet as an input and returns a JSON array, which contains the name two topics that have the highest probability, as well as the probability score itself.

Tweet Data Collector

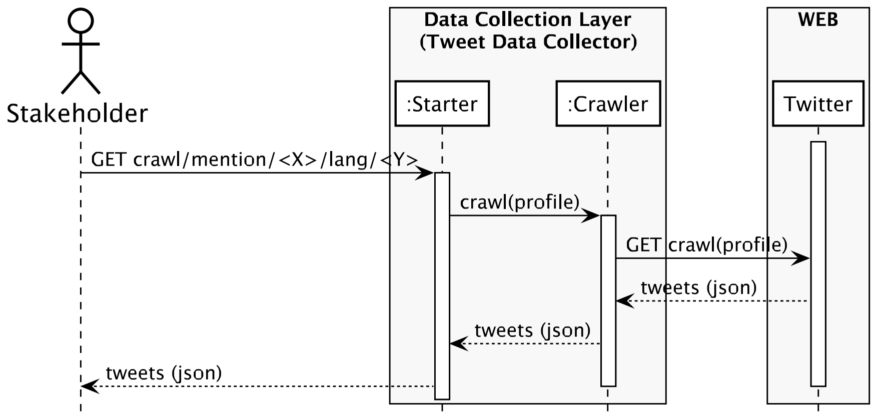


Figure 9.6: Tweet data collector communication flow.

The *Tweet Data Collector* microservice provides a simplified interface to the official Twitter API. As it used the official Twitter API, we first have to get the API keys from the official Twitter developer website. Therefore, before we can this service, we have to configure it with the Twitter API keys as environment variables or add them directly in the config file located in the codebase. Twitter provides different types of API keys, each coming with different limitations and rules. For example, we can either choose between Twitter's premium or free features. There are multiple versions of the premium API key, and depending on which premium version we select, the API allows us to crawl more tweets at once and further back into the history of the Twitter profile. The official Twitter website contains more information about the API limitations [246]. The API of this microservice expects the language of tweets we are interested in, as well as the Twitter profile name, which contains the tweets we are interested in (e.g., a support account as shown in Section 2.2.3). The microservice will then crawl as many tweets as the API key allows and returns a list of the crawled tweets.

Tweet Data Storage

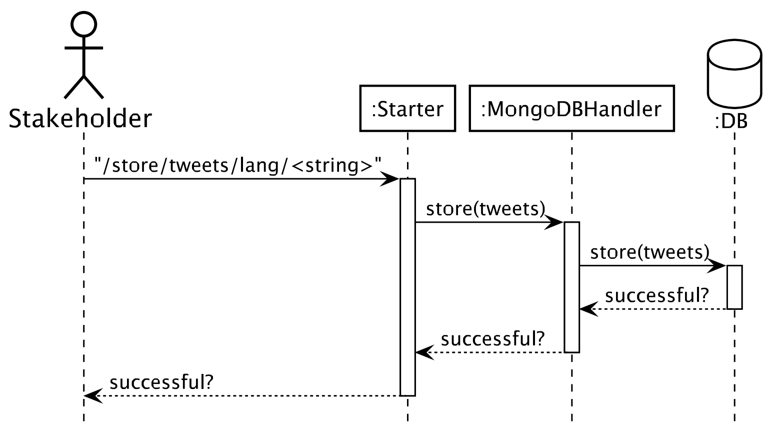


Figure 9.7: Tweet data storage communication flow.

This microservice is an API interface to a MongoDB instance. Based on the data that the endpoint receives, the microservice decides in which collection (table) to store the data. We decided for MongoDB as this technology allows us to better store and retrieve unstructured data. The advantage here is that if the Twitter platform may change the metadata or any other field we can receive from the endpoint, we do not have to find a solution for all historical data entries immediately, but can continue working with the data we have.

9.4 Integrated Interactive Visualization

This section describes the integrated interactive visualization, which visualizes the analysis results of the microservices introduced in the previous sections. It is the single point of access for stakeholders using the requirements intelligence prototype.

In the following, we present a subsection for each of the views the integrated interactive visualization provides. In each subsection, we show a screenshot and describe what it shows. We further detail our reasons for the illustrations as well as give insights into how stakeholders can use the widgets of the prototype.

9.4.1 Access to the dashboard

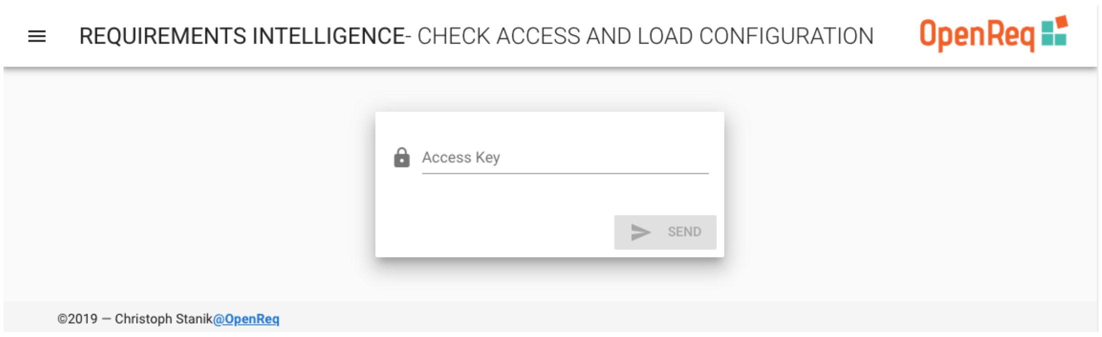
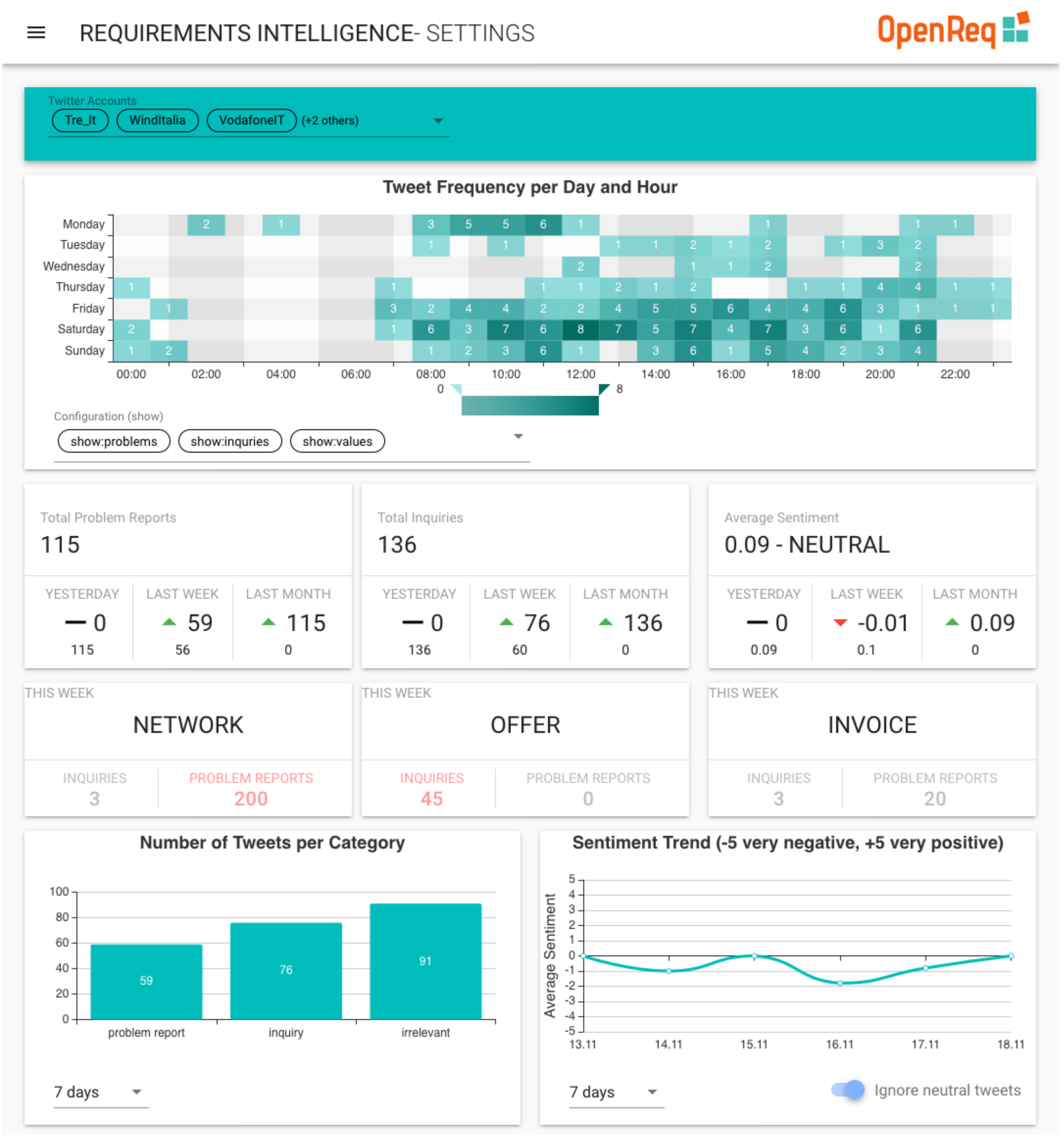

REQUIREMENTS INTELLIGENCE- CHECK ACCESS AND LOAD CONFIGURATION
OpenReq
Access Key
SEND
©2019 — Christoph Stanik@OpenReq

Figure 9.8: feed.ai—access key.

Figure 9.8, shows a view presenting an input form that requests an access key. It is the first view a stakeholder sees when using the prototype. The access key is mandatory and, therefore, the prototype does not allow to open any other view before the access key was validated. We introduced this view for several reasons. First, it provides preliminary protection of the underlying data. This functionality is an essential non-functional requirement from stakeholders, as they cannot allow other organizations to access their data and analysis results. Second, using access keys, we can create shared accounts for organizations that have the same configuration. For example, an organization decides using this prototype and requests access for several stakeholders that analyze the same data. Instead of including a user management system for individual stakeholders, we decided to simplify this process with access keys. Access keys, therefore, allow us to create shared configuration files for groups of stakeholders, although we can also create an access key for individual stakeholders. The third reason, which is related to the first, is that the prototype includes human control mechanisms for the machine learning models. With access keys, we can define who can correct the classification results of the machine learning models.

9.4.2 Dashboard



REQUIREMENTS INTELLIGENCE- SETTINGS

Twitter Accounts: Tre_It, WindItalia, VodafoneIT (+2 others)

Configuration (show): show:problems, show:inquries, show:values

Total Problem Reports		
115		
YESTERDAY	LAST WEEK	LAST MONTH
— 0	▲ 59	▲ 115
115	56	0

Total Inquiries		
136		
YESTERDAY	LAST WEEK	LAST MONTH
— 0	▲ 76	▲ 136
136	60	0

Average Sentiment		
0.09 - NEUTRAL		
YESTERDAY	LAST WEEK	LAST MONTH
— 0	▼ -0.01	▲ 0.09
0.09	0.1	0

THIS WEEK	INQUIRIES	PROBLEM REPORTS
NETWORK	3	200
OFFER	45	0
INVOICE	3	20

7 days

7 days — Ignore neutral tweets

Figure 9.9: feed.ai—dashboard.

The core of the integrated interactive visualization is the dashboard, whose goal is to give a quick overview of how an organization performs with respect to the opinions of its users. The dashboard in Figure 9.9 has four horizontal layers that we describe in the following.

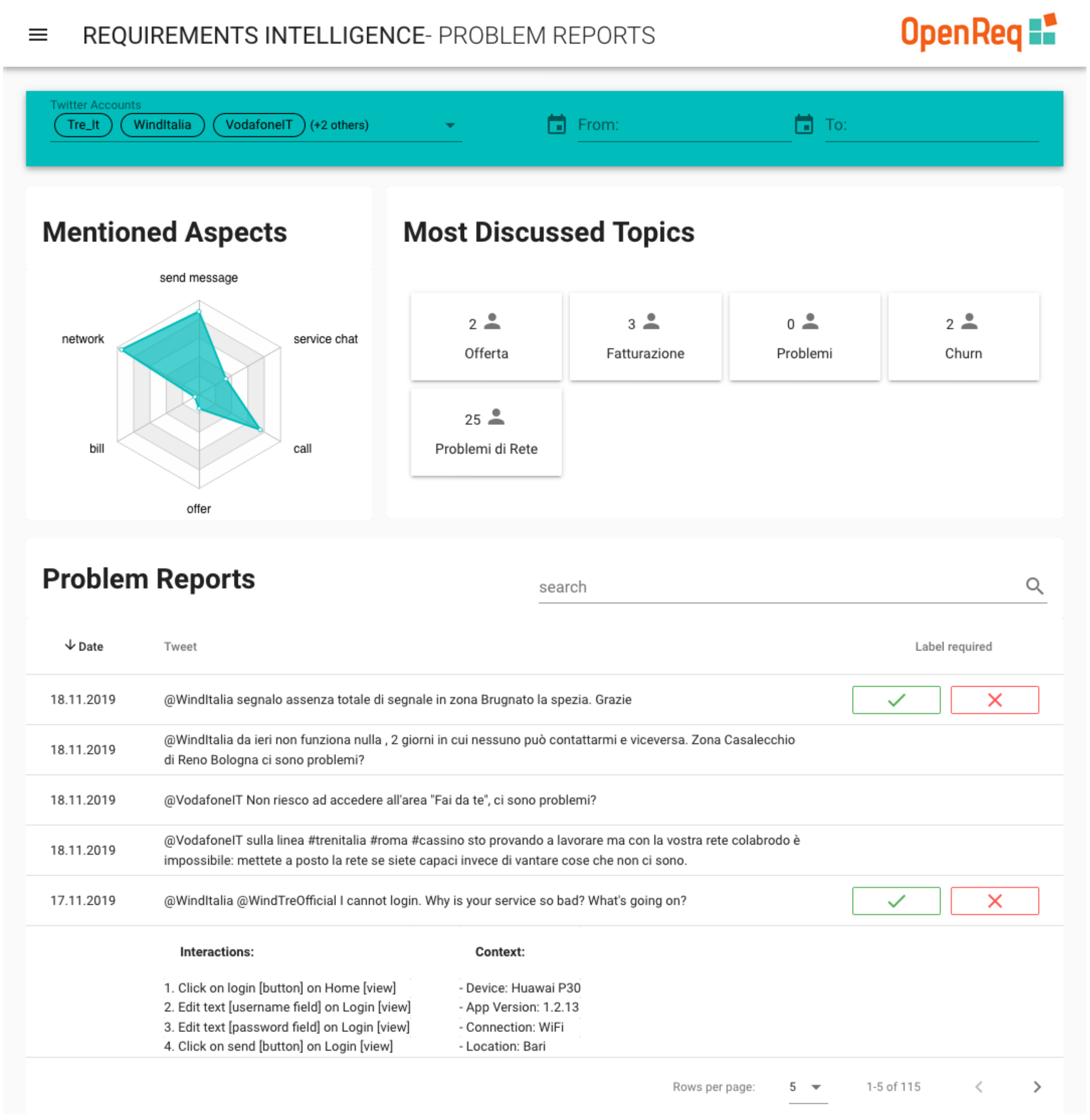
The top layer shows a heatmap that visualizes when users write feedback (use case UC2.1). The goal of this widget is to understand when users provide requirements-related feedback (such as bug reports and inquiries), when the organization's customer support needs to be more active, and when problems appear.

The second layer shows three distinct trend reports (use case UC2.2). Each trend report has the goal to show changes concerning different times in the past—i.e., changes since yesterday, last week, and last month. These views are particularly helpful to grasp if, for example, a new release of a feature introduced problems or the general performance increases or decreases. The outer right trend widget shows the overall sentiment of the user base. Values above 1.0 mean that users write positive feedback, while values below -1 means that users are unsatisfied—values between 1 and -1 indicate a neutral sentiment.

The third layer shows the three most discussed topics in the current week (use case UC2.3). Attached to the topic names are the number of inquiries and problem reports associated with that topic. If the number of reports reaches a specified threshold, they are marked in red to raise attention. For instance, the left-most topic is called "network". There had been three inquiries regarding the network topic in the current week, as well as 200 problem reports associated with it. As in this example, we set the threshold for raising an alert to 20 mentions of the topic or feature, the widget displays the lower right part of the card in red. Therefore, the organization may want to pay attention to that topic and analyze it further in one of the other views of the prototype that shows the actual feedback related to the topic.

The bottom layer shows two widgets. The left widget shows the proportions of how often users give either problems, inquiries, or irrelevant feedback (use case UC2.4). On the right, there is a user sentiment analysis. This analysis helps to visualize the sentiment for single days and puts it in relation to a given time frame. For example, if a newly released app feature contains a critical bug affecting many users, the sentiment is most likely to be negative. Therefore, we can use this chart to understand how urgent an issue may be, and when users are again in good standing with the organization. Both widgets on that layer enable the stakeholders to filter for specific time frames, such as the last seven days, 30 days, 90 days, this year, all time, or a custom time frame.

9.4.3 Focus Views


REQUIREMENTS INTELLIGENCE- PROBLEM REPORTS

Twitter Accounts: Tre_It, WindItalia, VodafoneIT (+2 others) — From: — To:

Mentioned Aspects

Most Discussed Topics

2	Offerta
3	Fatturazione
0	Problemi
2	Churn
25	Problemi di Rete

Problem Reports

search

Date	Tweet	Label required
18.11.2019	@WindItalia segnalo assenza totale di segnale in zona Brugnato la spezia. Grazie	✓ ✗
18.11.2019	@WindItalia da ieri non funziona nulla , 2 giorni in cui nessuno può contattarmi e viceversa. Zona Casalecchio di Reno Bologna ci sono problemi?	
18.11.2019	@VodafoneIT Non riesco ad accedere all'area "Fai da te", ci sono problemi?	
18.11.2019	@VodafoneIT sulla linea #trenitalia #roma #cassino sto provando a lavorare ma con la vostra rete colabrodo è impossibile: mettete a posto la rete se siete capaci invece di vantare cose che non ci sono.	
17.11.2019	@WindItalia @WindTreOfficial I cannot login. Why is your service so bad? What's going on?	✓ ✗

Interactions:

1. Click on login [button] on Home [view]
2. Edit text [username field] on Login [view]
3. Edit text [password field] on Login [view]
4. Click on send [button] on Login [view]

Context:

- Device: Huawai P30
- App Version: 1.2.13
- Connection: WiFi
- Location: Bari

Rows per page: 5 — 1-5 of 115

Figure 9.10: feed.ai—focus view.

The requirements intelligence prototype supports stakeholders in identifying user needs, such as the problems they face and the requests they have. While the dashboard provides a rather general view on how the organization performs, the focus view helps to browse and discover the written feedback. As described in Chapter 3, stakeholders need the functionality for browsing actual user feedback, as otherwise, they cannot understand act upon feedback. Therefore, we introduced two focus views for requirements-related feedback, one for problem reports and one for inquiries (use case UC3.2). Both focus views look the same and contain the same functionality. They only differ in the user feedback they show. Therefore, we only show and describe one of the focus views, as depicted in Figure 9.10. Each focus view has three horizontal layers.

On the top, there is a filter bar in which stakeholders can select the organizations and software products for which they want to analyze user feedback. The filter bar also allows filtering user feedback for specific releases, days, or custom time frame.

The second layer contains two widgets. On the left, there is a radar chart that summarizes the number of mentions of certain aspects such as topics, software features, or services with respect to the selected filters (use case UC3.3). The chart gives a brief overview of the trending aspects of user feedback and helps summarize the overall discussion (use case UC3.3). For instance, Figure 9.10 shows that the"network" topic and the features "send message" and "call" are the most discussed aspects. Since it is the problem report focus view, this example raises attention to a network issue that affects features associated with the network. The right side of the second layer shows predefined topics that are part of the access key configuration, as discussed in Section 9.4.1. The widget presents, for each topic, its name and its amount of user feedback available in the prototype. Stakeholders can click on the topics to further filter the list of user feedback they are interested in reading.

The bottom shows a table of user feedback that allows for browsing and discovering feedback with a keyword search (use case UC3.1). The table also supports pagination to not overwhelm stakeholders with a list of thousands of rows of user feedback. Each row contains the date of the user feedback, the feedback text body, and two buttons in specific cases. As this view is about problem reports, it shows the user feedback classified as such, using state-of-the-art approaches. As machine learning-based classification approaches do not provide perfect results and sometimes differ from what a human domain expert thinks, we introduced two buttons to either agree or disagree with the classification. To not overwhelm the stakeholders and to give meaningful input for the classification algorithm, we only show the buttons for the user feedback for which the classification algorithm is less than 70% confident. We chose this confidence threshold to balance the labeling effort of the stakeholders to improve the model and the impact the labeled data has on the machine learning model. We developed the prototype iteratively and evaluated it with a major telecommunication company. In each iteration, we updated the classification models utilizing the input of the human domain expert (use case UC4.1). If a stakeholder clicks on one of the problem reports, the row expands to show some analysis results of the implicit user feedback analysis if available. On the bottom of Figure 9.10, the example shows that the expanded row includes the order of interaction events on the left and further provides context information on the right side. This information can be valuable to understand, e.g., steps to reproduce, and to narrow down the location of an issue such as the view in the software on which a problem occurred (use case UC3.4).

9.4.4 User Interaction Data Insights

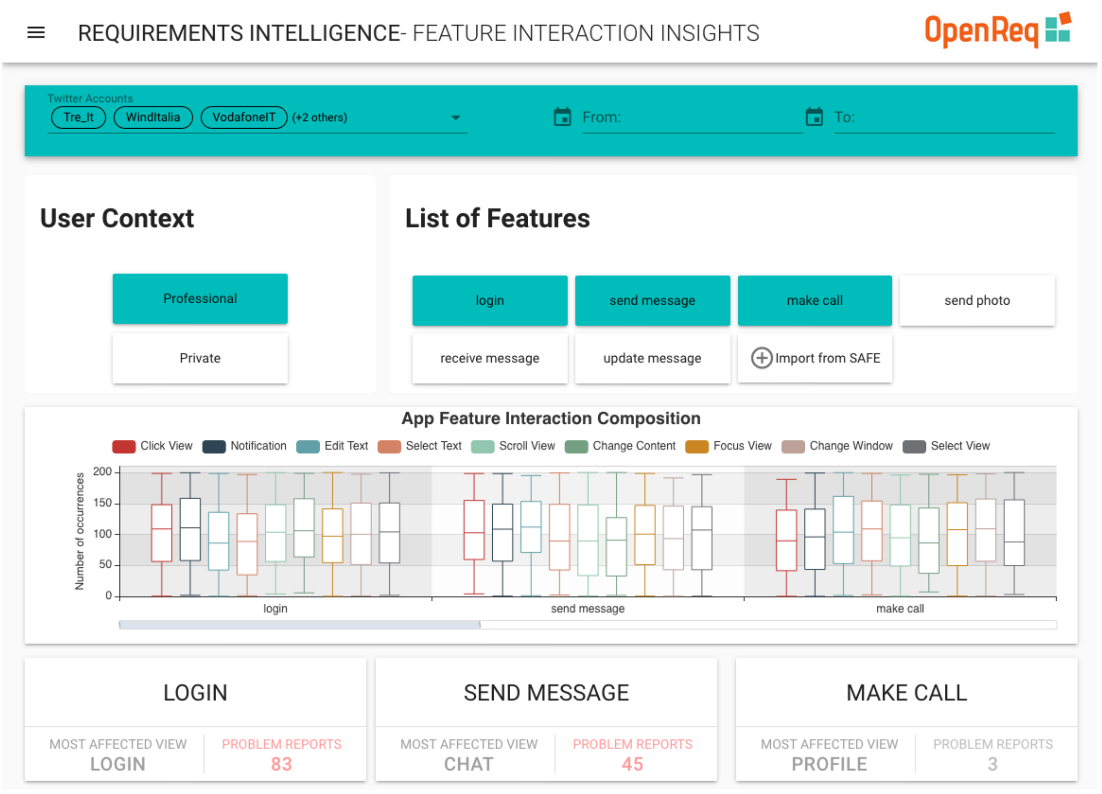


Figure 9.11: feed.ai—user interaction data insights.

The *user interaction data insights* view focuses on implicit user feedback. Figure 9.11 shows the screenshot of this view, which has four horizontal layers. As the top layer contains the same filter bar as the focus views, we do not describe it again (see Section 9.4.3).

The second layer contains two widgets. The first widget is a filter for the user context. In Chapter 7, we presented a study that analyzes context information, such as the used apps and the location of the user, to distinguish between private and professional device usage (use case UC3.2). This classification helps stakeholders to analyze features and requirements for specific situations. The stakeholders can use the widget to either filter one context type or keep both. In future studies, we could consider other context types such as sitting, walking, or driving, which is useful information for an organization focussing on navigation apps. The second widget is a list of software features. Stakeholders can either manually create the list and enhance or entirely create it with the SAFE approach from Chapter 6. Ultimately, the list is also a filter for the analysis of concrete user interactions and features. The selection in both widgets impacts the data visualization of the lower layers.

The third layer shows a group of box plots for each software feature we selected in the previous layer. Figure 9.11 depicts the box plot groups for the software features "login", "send a message", and "make a call" (use case UC3.1). Within each group, there is a box plot for each recorded user interaction event. It gives insights into how users are using the software features. For instance, a simple login view typically contains three control elements—two input fields (username and password), as well as a button to send the login information to a server. If users, on average, click more than once or twice per control element, there may be some issues related to the UI. Similarly, if users try to scroll often on a view which should contain its information directly visible on the device's screen, stakeholders need to investigate the issue further.

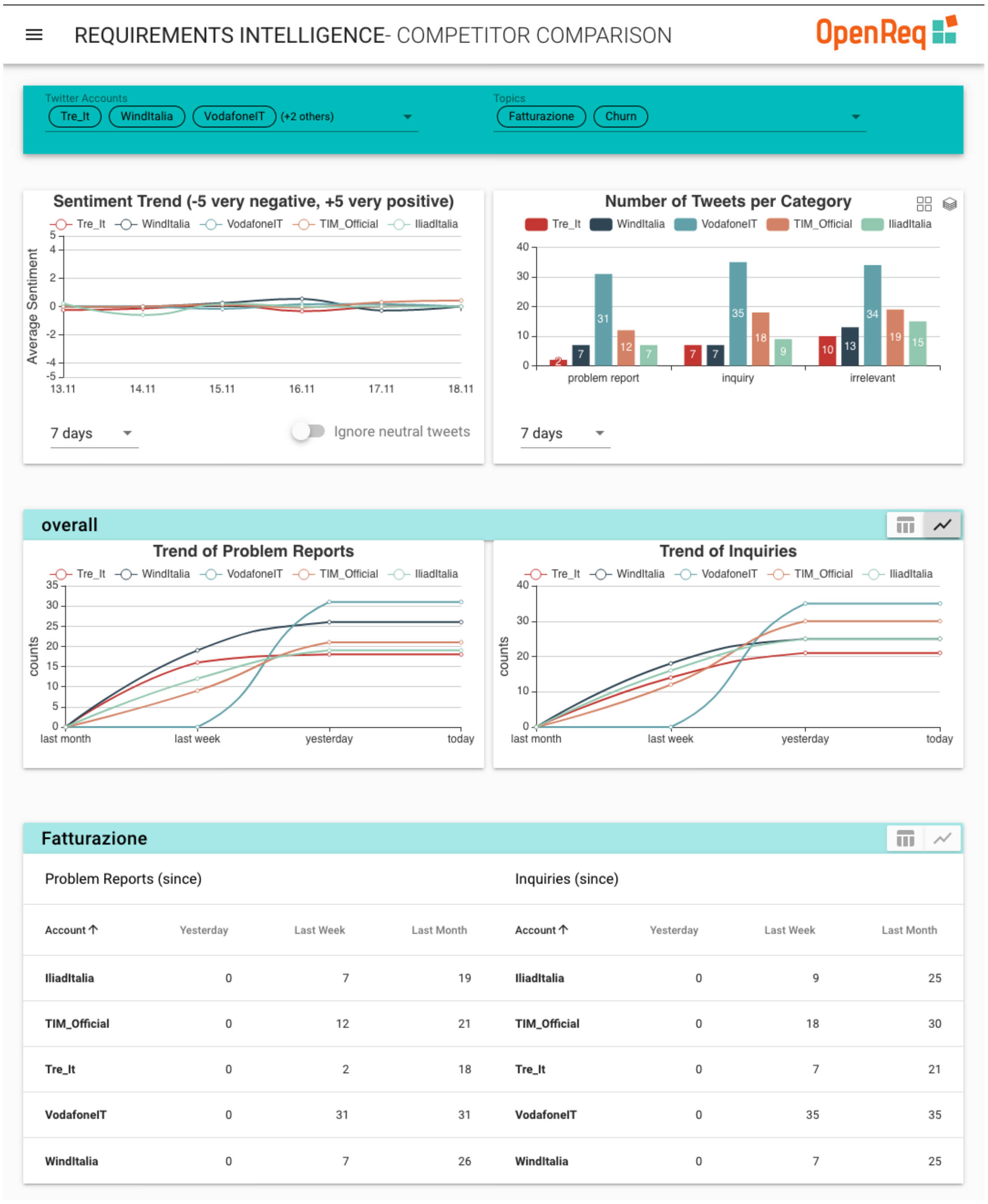
The bottom layer shows three cards, each belonging to a specific software feature (use case UC3.3). The idea of these cards is to show the number of reported problems for the filtered software features. Each card states the name of the software feature, indicates the most affected view of the software on which the problem occurred, as well as the number of users who wrote a problem report concerning this software feature. The most affected view is the result of the implicit user feedback analysis, while the number of problem reports comes from the explicit user feedback analysis. The card can raise an alert in case a specified number of problem reports toward this feature exist. Stakeholders can specify that threshold. Like the alert in the dashboard (Section 9.4.2), the number of problem reports is marked in red if the number of problem reports exceeds the specified threshold. In this example, the prototype shows the three software features from the selected filters and two cards that contain an alert, as more users reported problems as defined by the threshold.

In general, the widgets presented in the *user interaction data insights* view are indicative and may require further and more detailed analyses.

9.4.5 Competitor Comparison

The *competitor comparison* view arose from the need of stakeholders to learn how their product performs compared to the market. In feed.ai's competitor comparison view, we provide stakeholders with similar functionality as we presented in the focus views, but it also allows us to put the analysis results of several organizations and apps in one chart.

Figure 9.12 shows the competitor comparison. On the top left, the stakeholders can see the sentiment trend for different organizations and products (use case UC5.2). It helps in understanding how users perceive and discuss the products. For example, if a telecommunication organization faces a network outage, we can see if it also affected the competitors. If a network outage affected several organizations, the issue might be outside the own organization, while it is most likely within the organization if only they are affected. Further, stakeholders can use the view to track, e.g., the effect of advertisement campaigns like giveaways. On the top right, the user can compare the number of problem reports, inquiries, and irrelevant feedback users report (use case UC5.1). The bar charts allow understanding which organization receives more inquiries or problem reports. By combining the three bars of the feedback categories, we can see which organization receives more feedback in general. If an organization owns multiple products or apps, stakeholders can also use the competitor comparison view to compare their own products' performance.



Problem Reports (since)

Account ↑	Yesterday	Last Week	Last Month
IliadItalia	0	7	19
TIM_Official	0	12	21
Tre_It	0	2	18
VodafoneIT	0	31	31
WindItalia	0	7	26

Inquiries (since)

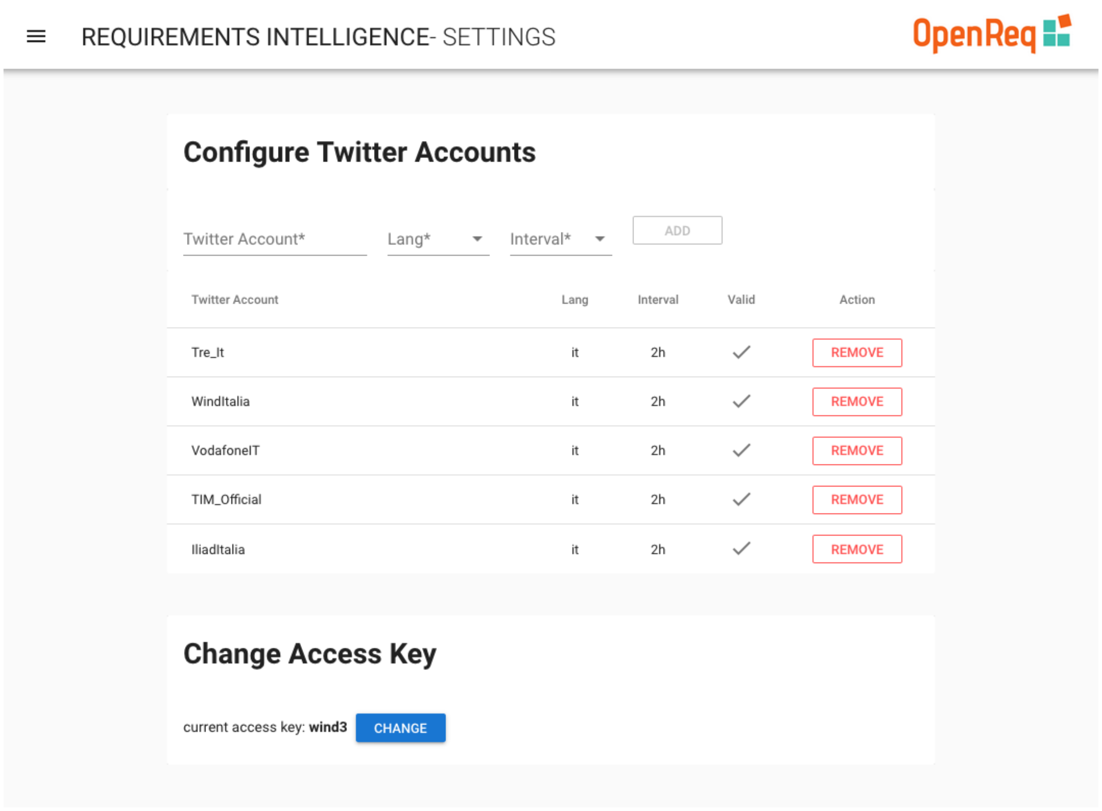
Account ↑	Yesterday	Last Week	Last Month
IliadItalia	0	9	25
TIM_Official	0	18	30
Tre_It	0	7	21
VodafoneIT	0	35	35
WindItalia	0	7	25

Figure 9.12: feed.ai—competitor comparison.

Below the sentiment and feedback category charts, the prototype shows additional layers. The third layer with the title *overall* is always visible and reports on the problem reports and inquiries trends. However, sometimes, this categorization may be too coarse-grained for stakeholders and needs additional insights.

Therefore, the filter bar on the very top allows selecting more fine-grained topics of interest. If, for example, a classification model for identifying network-related issues exists, stakeholders can select it in the filter bar. For each selected topic, the prototype generates one additional horizontal layer on the bottom. These analysis layers allow stakeholders to switch between a table and a multi-line chart widget. The layer contains the trend analysis for the selected topic regarding the number of problem reports and the inquiries users reported (use case UC5.3).

9.4.6 Settings


REQUIREMENTS INTELLIGENCE- SETTINGS
OpenReq
Configure Twitter Accounts
Twitter Account*
Lang*
Interval*
ADD

Twitter Account	Lang	Interval	Valid	Action
Tre_It	it	2h	✓	REMOVE
WindItalia	it	2h	✓	REMOVE
VodafoneIT	it	2h	✓	REMOVE
TIM_Official	it	2h	✓	REMOVE
IliadItalia	it	2h	✓	REMOVE

current access key: **wind3**

Figure 9.13: feed.ai—settings.

Although the stakeholders using feed.ai may know what user feedback types and channels they want to analyze, their needs may change over time. For that reason, we introduced a view allowing stakeholders to update the configuration of the access key. The settings view (see Figure 9.13) allows adding new feedback channels by specifying the name of the Twitter profile (use case UC1.1), the language of the feedback (use case UC1.2), and the interval specifying how often the prototype crawls and processes user feedback (use case UC1.3). The example in the figure shows that the stakeholders can add, edit, and remove Twitter profiles they want to monitor. Therefore, the settings view lists all Twitter profiles that are currently part of the access key. Whenever a stakeholder adds a valid Twitter profile or removes an existing profile, the prototype updates the access key. In case stakeholders have several access keys, they can change it on the bottom of the page.

9.5 Discussion

As discussed earlier, the requirements intelligence framework (see Chapter 4) contains a set of activities. In this chapter, we transformed the results of our scientific studies into one possible requirements intelligence prototype. The feed.ai prototype automatically analyzes implicit and explicit user feedback to support stakeholders such as project managers and developers. In practice, users of feed.ai only have to incorporate the data sources of interest, such as the configuration of Twitter profiles shown in Section 9.4.6. Then, the underlying microservices perform the data preparations and analyses that the web-based integrated interactive visualization presents. feed.ai covers various stakeholder needs, such as a dashboard that summarizes the overall product health. In Chapter 3, we discuss that, in particular, project managers are interested in getting a quick overview of the product's performance to understand if they have to consider immediate actions. Despite the dashboard, we showed views that provide more in-depth results by, e.g., including machine learning supported filters to omit irrelevant feedback and to discover frequently discussed topics and features. Two views focus on either problem reports or inquiries. They are particularly interesting for developers and other stakeholders that have to read user feedback. The view concerning user interaction data gives insights into how users use the software, e.g., an app or website, and reports on how the usage is composed. Further, we show examples of combining implicit and explicit user feedback by, e.g., attaching user interactions and context data to explicit feedback to better understand how an issue arose. Most views cover an introspective perspective, which focuses on the own products and services, but we also show how an extrospective perspective by illustrating an example for competitor analysis. As feed.ai is only a prototype demonstrating how we can implement the requirements intelligence framework, we also provided a reference architecture helping interested parties in either extending or re-implementing the framework. We further foster using, extending, and implementing microservices of the requirements intelligence framework as they are open-source. The open source repositories contain documentation for explaining the deployment, and if necessary, provide hints for developers for extending the service. The repositories are available on the GitHub page of OpenReq [183].

9.6 Summary

This chapter introduced feed.ai, a prototype based on the requirements intelligence framework from Section 4.3. After motivating the prototype, we summarized the requirements for feed.ai, which we based on our results from Chapter 3. In total, we defined five sub-systems and formulated 15 functional requirements in the form of use cases. We presented an overview of the requirements and detailed them afterwards. Then, we discussed the overall microservice architecture of feed.ai, which we split into five layers. The architecture contains microservices for each requirements intelligence activity. After that, we gave an example of how we implemented the microservices for the Twitter platform by illustrating and describing their dynamic models. Our intention of showing the overall architecture and describing a complete example for one feedback domain is to help understanding how feed.ai works, as well as how others can use and extend it. Afterwards, we presented screenshots of the integrated interactive visualization, explained them, and described how they cover feed.ai's requirements and stakeholder needs. Eventually, we discussed the implications of the feed.ai.

Chapter 10

feed.ai in Practice

I have always believed that technology should do the hard work - discovery, organization, communication - so users can do what makes them happiest: living and loving, not messing with annoying computers! That means making our products work together seamlessly.

Larry Page

Contribution. This chapter concerns the evaluation of the requirements intelligence prototype feed.ai in a case study. We developed feed.ai in five iterations in cooperation with a major Italian telecommunication company over 12 months. After each iteration, we employed methods like semi-structured interviews and surveys to receive feedback from stakeholders. We report on the feedback and give insights about how feed.ai helped stakeholders in their work with user feedback.

Addressed stakeholder needs. This chapter addresses stakeholder needs regarding the analysis of explicit user feedback. We cover the need for automated tool support for continuously collecting and preprocessing tweets from Twitter, feedback filtering, and identifying topics and features users address. The integrated interactive visualization provides a web-based interface for stakeholders.

10.1 Motivation

In Chapter 3, we described the need of stakeholders for software requirements intelligence. Our qualitative studies show that stakeholders anticipate less manual effort for analyzing user feedback. They desire tool support for automatically analyzing user feedback to understand user needs and support their decision-making process. The requirements intelligence bases its analyses on explicit and implicit user feedback, as described in Chapter 4. We presented approaches for the explicit and implicit user feedback analysis activities *feedback filtering* and *feedback to requirements*. Chapter 5 (activity feedback filtering) and Chapter 6 (activity feedback to requirements) are dedicated studies introducing approaches for analyzing explicit user feedback. For analyzing implicit user feedback, we also present evaluated approaches in Chapter 7 (activity feedback filtering) and Chapter 8 (activity feedback to requirements). We empirically evaluated all studies in this work in their dedicated chapters. However, the research results of the chapters are standalone studies. To understand if their results are useful, we created the requirements intelligence prototype feed.ai, which combines the approaches and visualizes user feedback insights in an integrated interactive visualization. The feed.ai prototype provides automated tool support for the activities presented in the requirements intelligence framework in Chapter 4. In Chapter 9, we presented our envisioned requirements intelligence prototype. In this chapter, we present an iteratively and fully implemented requirements intelligence prototype. The prototype development was part of the Horizon 2020 project OpenReq [182]. We evaluated the prototype with a major telecommunication company for 12 months. This chapter summarizes the evaluation process of the prototype and its results. In Section 10.2, we discuss the design of the evaluation, including a description of the evaluation context, setting, and evaluation questions. Section 10.3 reports on our results for each of the five iterations. In Section 10.4, we discuss the results of the evaluation and summarize the chapter in Section 10.5.

10.2 Design

Here, we describe the overall evaluation design. First, we detail the context of the evaluation in Section 10.2.1. Then, we present the evaluation setting in Section 10.2.2. In Section 10.2.3, we state our evaluation questions, while Section 10.2.4

describes our research methodology.

10.2.1 Evaluation Context: OpenReq

We introduce the Horizon 2020 project OpenReq [182], which is the evaluation context.

OpenReq is about researching, developing, and evaluating intelligent recommendations and decision technologies to support communities and individual stakeholders in the gathering and management of software requirements. OpenReq bridges the gap between the development and usage of software-enabled products and services: by taking the user community as part of the innovation process and by continuously observing and involving stakeholders and users and tightening their commitment in the decision-making. OpenReq aims at developing an integrated approach that continuously monitors and adjusts requirements knowledge instead of having a waterfall process. The project encompasses the following objectives and activities.

Requirements gathering, reuse of requirements & stakeholder management. Stakeholders gather requirements in the early project phases. OpenReq supports stakeholders in effectively gathering and cooperatively screening the relevant requirements. The gathering activity is a continuous process over the whole project's lifetime. The project further screens the previous requirements knowledge of users and their communities for identifying reusable requirements. OpenReq further helps to identify stakeholders who should work together on certain requirements and supports new stakeholders in effectively acquiring knowledge about already existing requirements. The project also assists users in participating in the requirements engineering process.

User needs & screening. The project involves users implicitly and explicitly during the whole project's lifecycle. We acquire implicit feedback by observing the actual usage of the software at run-time through OpenReq. Software organizations and service providers will better and easier assess the priorities of particular features and requests. We further collect explicit feedback through social media channels and specialized E-Participation platforms to derive emergent needs, usage trends, and new requirements.

Analysis, dependency detection & diagnosis. The screened requirements are analyzed and documented in detail. OpenReq supports stakeholders in identifying and managing dependencies between requirements and identifying (preferred) adaptations in the case of inconsistencies. We identify dependencies on a semantic level based on different types of natural language processing techniques. OpenReq supports easy requirements adaptation after changing a requirement by automatically identifying related requirements that presumably have to change as well.

Intelligent release planning. Openreq supports the creation of release candidates and the group-based decision-making about candidates that stakeholders should consider in the requirements negotiation phase. The project filters relevant requirements based on the stakeholders' preferences, user preferences, time constraints, and budget. These filtered requirements are candidates in the release planning activity, which stakeholders decide about in a guided discussion.

This work addressed the **user needs & screening** activities of OpenReq.

10.2.2 Evaluation Setting

We performed the evaluation in cooperation with a leading Italian telecommunication company that serves about 30 million customers (in the following called users) for both mobile networks and fixed-lines. They have an annual revenue of 5.6 billion euros and about 6,800 employees. The company owns the following three brands:

- **Brand A**[1]. It targets the consumer segment (mainly families) by providing services for mobile and fixed-lines. The brand stands for two characteristics. First, it aims at being as close as possible to its users. Second, it intends to offer clearly and easily described plans.
- **Brand B**. Addresses younger people who need more flexibility in their plans and frequently updated services. The brand focusses on mobile services and fosters innovative solutions, diverse features, and high performance.

[1]anonymized

- **Brand C**. The business segment covers both human and machine to machine (M2M) SIMs. The vision of the brand is to be a trusted partner for business clients supporting their daily business operations and offering innovative solutions to them.

The goal of our evaluation partner is to address the increasing popularity of social media and the pervasiveness of connected devices. In particular, the company strives to attract and retain users that are socially connected and highly informed. For attracting and retaining users, they want to offer new innovative and software-enabled products and services corresponding to the users' expectations. They, therefore, look for automated solutions helping them to understand their users better. They further want to use such a tool to increase the interactions between them and their users.

In discussion with the evaluation partner, we decided to focus on the explicit user feedback analysis activities of the requirements intelligence framework. We excluded the implicit user feedback analysis activities in the evaluation as the evaluation partner's policies do not allow us to collect and analyze implicit user feedback from their apps.

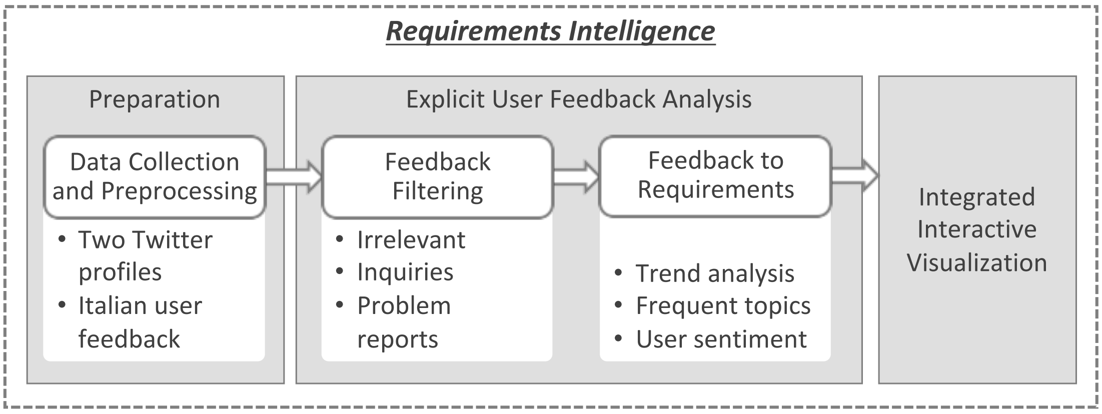


Figure 10.1: The evaluated activities of the requirements intelligence framework, including an overview of the evaluated activity results.

Figure 10.1 details the scope of the evaluation. For the *Data Collection and Preprocessing* activity, we target the social media platform Twitter. The reason for that decision is that the evaluation partner identified it as the most actively used platform of their users. They own two Twitter profiles that they actively use to provide support to their users and to advertise current and upcoming products and services. The first Twitter profile has about 166,000 followers and targets *Brand B*. The second Twitter profile has 122,000 followers and targets *Brand A*. As the evaluation focuses on the Italian market, we only consider Italian tweets.

The activity *feedback filtering* uses the approach introduced in Chapter 5. The telecommunication company in our evaluation required the automated identification of relevant and irrelevant user feedback from Twitter. We further classify the requirements-relevant user feedback to problem reports and inquiries. Therefore, we introduce a two-step filtering approach, which first filters irrelevant feedback and then performs the second classification.

In the *feedback to requirements* activity, we identify frequently discussed features and topics around the product and services of the company, such as network availability and product offers. Further, the evaluation partner required a trend and sentiment analysis, which gives an overview of the company's performance in social media. Eventually, the experts at the evaluation partner's side requested a mechanism to correct the analysis results. The further asked for analytics insights, such as how many users report a particular feature/topic and to browse and filter the actual tweets. They need this functionality so that they can export the user feedback to another tool that supports the discussion and prioritization of requirements-relevant tweets.

Finally, the integrated interactive visualization should be a web-based dashboard that visualizes the analysis results and the original tweets.

10.2.3 Evaluation Questions

The objective of the evaluation was twofold. First, we wanted to iteratively develop a prototype of the requirements intelligence framework that integrates state-of-the-art research results, which stakeholders can use in practice. Second, we wanted to understand the overall usefulness of the developed prototype from the perspective of stakeholders. To achieve this, we collaborated with a major telecommunication company that used the prototype in their daily workflow. We aligned their needs with the idea of requirements intelligence and adapted the prototype based on frequent feedback loops.

We formulate the following evaluation questions to validate our objectives.

RQ10.1 Usefulness: How do stakeholders assess the usefulness of each activity and visualization of the requirements intelligence prototype?

RQ10.2 Improvements: What visualization needs improvements? Why does it need improvements, and how should we improve it?

RQ10.3 Impact: Which department can benefit from the requirements intelligence prototype? How does it help stakeholders in their daily work?

10.2.4 Evaluation Methods and Timeline

We developed the prototype in five iterations and evaluated each iteration with either a survey (including open and closed questions) or semi-structured interviews. Table 10.1 depicts the timeline of the evaluation process. It shows that

Table 10.1: Timeline of the evaluation and the used evaluation methods. Each row represents the evaluation of one developed iteration.

OCT 18 ●	Survey (2 participants)
DEC 18 ●	Survey (3 participants)
MAR 19 ●	Interviews (3 participants)
JUL 19 ●	Email/phone (2 participants)
SEP 19 ●	Survey (10 participants)

we developed and evaluated the prototype for 12 months. Depending on the state of the prototype, we employed the evaluation method for feedback that was most useful for us. Before we started the development of the first iteration, we discussed intensively with the evaluation partner what their needs are. We documented and used their challenges and envisioned solutions as a basis for the prototype. Beginning of October 2018, we finished the first prototype iteration, which was also the first subject for our evaluation. For the evaluation of the first iteration, we designed closed and open questions about every single widget visible in the integrated interactive visualization. Each widget represents one result of the requirements intelligence activities and the extracted analytical insights. Based on the survey results, we improved the prototype and created a second iteration. In the second iteration, we had one additional participant from a different department (as detailed in the results). Afterwards, we decided to perform interviews with the third iteration to gain in-depth qualitative insights by talking to three stakeholders using the prototype. We used closed questions to get more information about their role in the company and the department they work for. With open questions, we aimed at 1) getting more detailed use case scenarios and 2) to better understand the pain points they had when using feed.ai. In July 2019, we deployed our fourth iteration of the prototype, which was the basis for an agile development process utilizing direct contact with the evaluation partner via phone and email. The result of this agile development process is the final evaluated prototype (see Figures in Appendix B.5). For the evaluation of the final iteration, the partner distributed a survey to 10 potential stakeholders within the company who anonymously answered the survey.

10.3 Results

Chapter 3 focuses on the general needs of stakeholders for analyzing user feedback, while Chapter 9 reports the technical aspects of the envisioned prototype. Here, we report on a fully implemented open-source prototype that helped us creating the feed.ai prototype in Chapter 9. This section shows screenshots of the integrated interactive visualization of the developed prototype. Each step in the timeline in Figure 10.1 evaluated a single iteration of the prototype. For each iteration, we summarize its functionality and give a brief overview of the evaluation feedback we received. For each iteration, we discuss the findings to the evaluation questions from Section 10.2.3.

10.3.1 Iteration 1

Iteration	1 (OCT 18)
Screenshots	Appendix B.1
Method	Survey
Participants	• Senior Innovation Manager (P1) • International Relations and Assistant Innovation Manager (P2)

Functionality. In the first iteration, we developed a requirements intelligence prototype that analyzed one single Twitter profile. The prototype included microservices for continuously collecting the tweets from the given Twitter profile. A second microservice employed the feedback filtering activity to identify requirements-relevant tweets. The integrated interactive visualization presented three analysis widgets. First, it showed a bar chart that displayed the proportion of *problem reports*, *inquiries*, and *irrelevant* user feedback. Second, the tool showed a sentiment line chart that depicted the average sentiment per day of all user feedback in the current week. Third, it contained a table with three columns. Each column had a list of the original tweets belonging to either *problem reports*, *inquiries*, and *irrelevant*.

Feedback. We evaluated the first iteration with a survey. We had two participants, a senior innovation manager (P1) and international relations and assistant innovation manager (P2). Both survey participants stated that they used the prototype for about one hour while P1 used it on two days, P2 used the tool on a total of six days. Both agree that filtering the tweets is useful for them. P1 highlighted the usefulness of filtering irrelevant user feedback, while P2 also found the user feedback belonging to inquiries as most useful at this stage. However, both agree that the high-level filtering approach had only limited value for them. P2 stated that it would be more helpful to focus on specific topics and features like problems regarding the network. P1 said that tags summarizing the user feedback would help to grasp the tweet's central message quicker. Both participants reported that the first iteration could have a significant impact on the company but that they could not benefit from it.

10.3.2 Iteration 2

Iteration	2 (DEC 18)
Screenshots	Appendix B.2
Method	Survey
Participants	• Senior Innovation Manager (P1) • International Relations and Assistant Innovation Manager (P2) • Data Scientist (P3)

Functionality. Based on the feedback, we re-designed and re-implemented the prototype for the second iteration. We developed a dashboard for the integrated interactive visualization (see B.2), which included four types of widgets for the analysis. First, it included a heatmap to detail on when users tweet to the company. Second, we kept the bar chart for the three filtered feedback categories. Third, we extended the sentiment line chart with a comparison of the users' perception of the current and the last week. Fourth, we developed widgets showing trends for *problem reports*, *inquiries*, and users' *sentiment*. The widgets report descriptive analytics concerning predefined time frames. In particular, they show the delta from today, compared to yesterday, last week, and the last month. Besides the dashboard, we further introduced a comparison view (see Figure B.3), which allowed filtering user feedback based on selected time frames. The comparison view also included the proportions of the tweet categories, as well as basic sentiment analysis. Further, we re-designed the tweet list (see B.4) to include the date of the tweets, as well as additional filters like time frames and keyword searches. As the evaluation partner criticized the performance of the automated filtering approach, we included a human control mechanism allowing the partner to correct the machine learning results (see B.5). We used corrections of the evaluation partner to improve the classification models incrementally in the future iterations.

Feedback. In addition to P1 and P2, a data scientist (P3) joined the evaluation. All stakeholders agreed that the control mechanism to correct the filtering approach is very helpful and that they see that their input improves the machine learning model's performance. The stakeholders evaluated the usefulness of the heatmap differently. P2, who also worked in public relations, found this information very helpful (5 out of 5 points), while P1 and P3 did not find the information helpful for their work. The stakeholders did not find much value in the sentiment analysis as it was neutral for the whole time of the evaluation (they could not observe any positive or negative peaks). One of the reasons for that was that the underlying approach averaged the sentiment of all tweets. After we updated the classification models, the participants perceived this feature more positive. The new views, specifically the comparison view and the tweet list view, were seen as helpful to very helpful. The stakeholders stated that adding user information to the tweets such as the number of followers and, as users with more followers, may have a bigger impact (P3) as a point for improvement. However, P1 mentioned in the first iteration, that user information should be anonymized. Overall, the stakeholders saw more potential in the tool and agreed that it improved compared to the previous iteration. For instance, P1 stated: “I think the tool overall has potential, however it needs refinement in the way information is conveyed. For example I would exclude irrelevant tweets from most views and neutral tweets from sentiment analysis barchart. I think the tool will benefit the most Marketing and CVM departments. Users from those departments should/could request customized dashboards and classification categories for very specific needs”. The stakeholders gave at least 4 out 5 points for the usefulness to keep the machine learning models up-to-date (Figure B.5). However, the stakeholders also stated that continuously labeling all user feedback is unfeasible. The stakeholders found that the second iteration of the prototype may create the most benefit in the marketing, operations, customer value management, and big data departments.

10.3.3 Iteration 3

Iteration	3 (MAR 19)
Screenshots	Appendix B.3
Method	Semi-structured Interviews
Participants	• Senior Innovation Manager (P1) • Technology Enterprise Architect (Big Data and Cloud) (P4) • Technology Innovation Manager (P5)

Functionality. The third iteration included minor changes to the dashboard and two major changes. Regarding the changes in the dashboard (see Figure B.6, we introduced new filtering mechanisms by including two switches in the heatmap. The first switch toggled between highlighting all user feedback or showing only requirements-relevant user feedback. The second switch could toggle between showing or hiding the absolute number of user feedback for any given time and day. Additionally, we changed the sentiment analysis by only showing one line for a defined time frame instead of comparing two weeks. Both the bar chart for the feedback categories and the sentiment line chart introduced time frame filters. Each widget could have its own time frame.

The first major change in this iteration was the introduction of *focus views* for *problem reports* and *inquiries* (see Figure B.7). The reason behind that decision was to make clear what kind of analytics insights the stakeholders see. This change merged the feedback list and the feedback labeling task list from the previous iteration. Within the focus view, the stakeholder could filter for different Twitter profiles, time frames, and search for keywords of interests. Further, stakeholders do not have to correct all machine learning results in the human control mechanism, but only the user feedback for which the machine learning approach is uncertain. This decision had two benefits. First, it reduced the effort for the stakeholders. Second, machine learning models improve most if they get new labeled data for uncertain cases.

The second major change is the introduction of the *access key* view. The purpose of this view was to define profiles for the stakeholders. Stakeholders could decide from which Twitter profile they wanted to get analysis results. For example, the development team may only have and interest in their own app. The marketing department, however, might like to compare their app with those of the competitors. Both could define their own access key, which is associated with a configured profile customized to the stakeholders' needs.

Feedback. P4 is a technology enterprise architect whose department is in close contact with the marketing and product development department. The technology enterprise architects of the evaluation partner are particularly interested in products introduced within the next six to twelve months and, therefore, are mostly using the inquiry focus view. P5, who is a technology innovation manager, was interested in innovative products that are being introduced in the next five to ten years. Therefore, P5 was interested in analyzing both problem reports and inquiries. Problem reports are a good indicator to validate if the current products meet the needs of the users. The stakeholder found the inquiry focus view useful because it helped her understanding if the mentioned features already exist in the products. It further helped her identifying if the features were part of the products, but their users did not know about them. However, the prototype did not allow them to filter user feedback for fine-grained topics and features and relate them to specific products or services. The stakeholders stated that getting that kind of granularity is necessary to reduce the effort of reading all user feedback or finding the right keywords to filter the list of feedback. All participants again highlighted the importance of continuously updating the machine learning models. In particular, they liked that they do not have to label all user feedback (see Figure B.7), as this reduces their effort and makes the approach more feasible.

10.3.4 Iteration 4

Iteration	4 (JUL 19)
Screenshots	Appendix B.4
Method	Email/Phone
Participants	• Senior Innovation Manager (P1) • Data Scientist (P3)

Functionality. The fourth iteration included two major changes. First, the focus views introduced the opportunity for filtering user feedback by predefined topics. For instance, the evaluation partner was particularly interested in identifying issues related to the network, the billing system, and offers. For that, we included cards in the focus views, stating the topic, the number of users reporting it, and its average sentiment. Stakeholders could click on any of the topics to filter the user feedback list below. Identifying features and topics belongs to the third activity of the requirements intelligence framework, feedback to requirements. The second major change was the introduction of the *settings* view, which allowed the evaluation partner to add, remove, or update Twitter profiles for their analyses. The settings view allowed stakeholders to define the Twitter profiles they are interested in, the language of the user feedback, as well as the data collection and analysis interval.

Feedback. This iteration did not follow an empirical method to evaluate the prototype, but it embraced agile communication to include improvements and changes to the prototype quickly. The goal was to prepare the prototype for the final evaluation with multiple stakeholders from the evaluation partner. During the period of one month, we had frequent contact with two stakeholders in the form of phone calls and email exchanges. The communication led to many performance improvements. Besides that, the participants stated that they found it difficult to compare the analytics results of both specified Twitter profiles. The prototype was limited to either aggregate the analyses results of all defined Twitter profiles or to present the results of one Twitter profile. The first case made the comparison impossible, while the latter was cumbersome as the stakeholders could only see the analysis results of one Twitter profile at the time. The stakeholders, therefore, suggested introducing a view for comparing the feedback analysis results of multiple Twitter profiles. One stakeholder said that using feed.ai helped them to save 70% of their time for analyzing user feedback.

10.3.5 Iteration 5

Iteration	5 (SEP 19)
Screenshots	Appendix B.5
Method	Survey
Participants	• 10 anonymous participating stakeholders • Three from the marketing department • Four from the customer care department • Thee from the architecture department

Functionality. Based on the previous feedback, we included a *competitor comparison* view in the fifth and final iteration of the evaluation. In combination with the settings view of the previous iteration, the stakeholders of the prototype could add competitors to analyze and compare their performance in the newly introduced view. The competitor comparison view (see Figure B.16) allowed stakeholders to compare the overall sentiment of their users with a line chart covering a predefined time frame. A bar chart allowed stakeholders to compare the feedback filtering categories (problem report, inquiry, and irrelevant). Further, the stakeholders could see and compare the trends for the feedback filtering categories and the topics across multiple Twitter profiles. For example, if the company has a network issue, they could check if they were the only company experiencing this issue or if others were facing the same problem.

Feedback. The final survey included nine closed questions to rate the functionality and usability of the final prototype (see Appendix B.5. We constructed all nine questions as a 5-point Likert scale (I fully disagree to I fully agree). The results of this survey are shown in Figure 10.2. We anonymized the roles and the name of the participating stakeholders but asked them in which department they work. Three stakeholders were part of the customer care department, four belong to the marketing department, and three are part of the architecture department. The overall impression of the participants was positive. The evaluation shows that they evaluated the sentiment analysis with 3 out of 5, meaning that it did not provide meaningful insights for most of the stakeholders. As the sentiment analysis presents the average score of all user feedback per day, the sentiment was often neutral and did not provide much information. We tried to mitigate this effect by allowing stakeholders to remove neutral tweets from the analysis, which increased the impact of positive and negative tweets. The focus views were the most positive perceived feature of the integrated interactive visualization. The focus views allowed stakeholders to browse and filter actual user feedback. Regarding usability, all participants agree that the prototype was easy to use. Most were satisfied with the design of the dashboard and agreed that they would use the tool frequently. However, the features of the tool could have been better integrated, but as the survey only considered closed questions, we cannot derive the rationale.

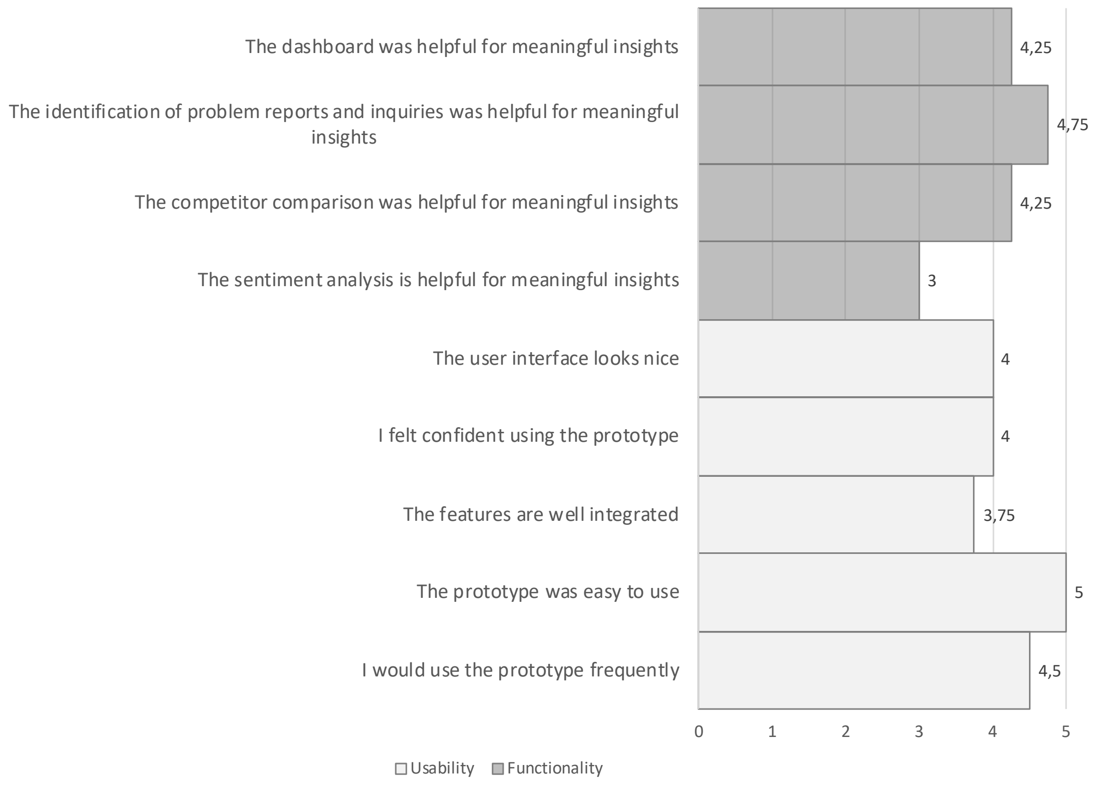


Figure 10.2: Final survey results.

10.4 Discussion

In the following, we aggregated the findings of the evaluation, including all iterations.

The stakeholders think that the requirements intelligence prototype can benefit diverse departments. During our evaluation, the stakeholders stated that the requirements intelligence prototype could benefit the departments: marketing, customer relationship, customer value management, operations, and big data. All of the mentioned departments include user feedback in their workflow. One stakeholder revealed that the company currently moves from waterfall product development to agile product development. In particular, the agile teams started looking at user feedback from social media to get inspiration for innovations and to find problems with existing products. The customer relationship management can benefit from the prototype by reporting on the descriptive statistics of the dashboard and enrich that report with examples from actual tweets. The marketing department can analyze if, for example, advertisement campaigns improved the sentiment of the users or to check how many users are discussing them.

Requirements intelligence can support the discovery of innovation. The prototype was evaluated and used in departments that are responsible for creating innovative products. In particular, P4 searches for innovative features and products the company can release within the next six to twelve months. This stakeholder was particularly interested in the inquiry focus view, which helped him identify what their users wish. He reported that he was looking at the focus view regularly to check if users came up with new ideas or to check if some ideas got reported repeatedly. P4 used the dashboard as an indicator to see how many inquiries were added since the last time he checked them to understand if it is worth reading them.

Requirements intelligence helps to validate if the existing products meet user needs. Once a feature, product, or service got released in the market, it is crucial to understand how users perceive that release. For example, P5 is a technology innovation manager who is responsible for developing innovation in the course of five to ten years. In that long-term perspective, the stakeholder said that they look at both focus views that the requirements intelligence prototype offers. In particular, the stakeholder found the problem report focus view valuable as it helped her validating if existing products meet the needs of the users. From these insights, she can get ideas for long-term improvements or new products.

The dashboard helps to understand the general performance of the company quickly. Stakeholders found that the dashboard, we introduced in the second iteration, is meaningful for getting a quick overview of how the company performs from the perspectives of their users. They generally liked the descriptive statistics we displayed in the trends, as well as in the bar charts to indicate how many additional problem reports and inquiries they received. However, stakeholders criticized the sentiment analysis, which could not provide any meaningful insights as the general user sentiment was neutral most of the time of the evaluation. A more fine-grained sentiment analysis that considers emotions like anger or joy could provide meaningful insights.

Including a human control mechanism is meaningful if stakeholders do not get overwehlmed. Multiple stakeholders stated that filtering the user feedback into irrelevant, problem reports, and inquiries are valuable for them. They particularly highlighted that having control over the machine learning experiments is vital, as they do not agree with all of the classification results. However, the stakeholders felt overwhelmed with the human control mechanism we introduced in the second iteration. The reason was that the tool asked the stakeholders to either agree or disagree for all collected tweets. Therefore, limited the number of tweets stakeholders should label to those that have the biggest impact on improving the machine learning model. The stakeholders found that change crucial and found that it made the approach feasible for them.

Improving machine learning models over time, based on stakeholder's input, improved their acceptance of the automated approach. The stakeholder stated that they like that they are in control of the machine learning models. In particular, they stated that they could see that the machine learning models were improving over time, which gave them higher confidence in the approach. The prototype tracked if the stakeholders agreed or disagreed with the classification results and found that with the first machine learning model, they agreed with 74% of the classifications. Based on their corrections, we re-trained the machine learning model and integrated the improved version in the prototype. The stakeholders agreed with 82% of the classifications of the improved machine learning models. In the 12 months evaluation period, the stakeholders provided 1,800 labels for the machine learning results. In the final machine learning model, which included all 1,800 corrections, the stakeholders agreed with 92% of all classifications. Therefore, we could improve the models by 18% compared to the first deployed machine learning model, that we trained with crowd-labeled data (for more information see Chapter 5).

10.5 Summary

This chapter aimed to report on the evaluation of the requirements intelligence prototype feed.ai, which we based on the stakeholder needs reported in Chapter 3. We evaluated the prototype with a major telecommunication company for over 12 months. We developed five iterations of feed.ai based on the feedback of the evaluation partner's stakeholders. We gathered their feedback by either using surveys, performing semi-structured interviews, or phone meetings. The stakeholders had diverse roles and came from different departments, like public relations and innovation management.

Based on the stakeholder needs and their privacy policies, we limited feed.ai's evaluation to the explicit feedback analysis activities and the integrated interactive visualization. The prototype continuously collected explicit feedback from the two Twitter profiles our partner operates. These two profiles provide support to their users and advertise its product portfolio.

We evaluated the final iteration in a survey with ten stakeholders working in customer care, marketing, and the architecture department. Overall, they perceived the prototype positively in both its functionality and usability. In the following, we highlight some findings.

- Requirements intelligence benefits departments working with user feedback.

- The dashboard helps to get a quick overview of the overall performance of the company.
- Identifying inquiries can foster the innovation process.
- Problem reports can support stakeholders to see if the company meets users' needs.
- Enabling stakeholders to correct the automated approaches increases stakeholder's trust in them and improves the approach over time.
- feed.ai's user feedback analytics insights helped the evaluation partner to save 70% of their time.

Generally, we were able to develop a prototype that meets stakeholder needs. We could also confirm that implementing an approach for the stakeholder needs from Chapter 3 is useful for them in their actual work.

Chapter 11

Conclusion

The aim of an argument or discussion should not be victory, but progress.

Joseph Joubert

This chapter summarizes the contributions of this work by detailing topics around requirements intelligence. The chapter further discusses directions for future work.

11.1 Summary of Contributions

Here we discuss the contributions of this work for the topics *requirements intelligence*, *core enablers for requirements intelligence*, and *the feed.ai prototype*.

11.1.1 Requirements Intelligence

We present our findings and conclusions from Chapter 3, which clarifies stakeholder needs for automated analyses of user feedback and Chapter 4, which introduces requirements intelligence.

Stakeholder Needs for Automated User Feedback Analysis. Traditional user involvement in requirements engineering encompasses approaches like workshops, interviews, and observations. Stakeholders often perform these approaches in the early phases of a project. Although requirements engineering activities such as the elicitation activity benefit from involving users with these approaches, they are challenging to carry out continuously and involve only a limited number of users (See Section 2.1). In recent years, stakeholders discovered that written online user feedback, as well as usage data of an app, are continuous sources for requirements-related information. Yet, stakeholders rarely use these sources in their decision-making process because manually analyzing user feedback is unfeasible. Written online user feedback (explicit user feedback) and usage data (implicit user feedback) come in large, unfiltered, noisy amounts. Stakeholders need automated approaches to guide their decision-making based on user feedback. For that, they desire tools that filter user feedback to reduce its overwhelming amount. Further, stakeholders see the most potential if they can combine explicit and implicit user feedback because usually, non-technical users write this feedback and, therefore, misses important information like the usage context. Stakeholders want to stay informed all the time and desire a web-based standalone tool. Within the tool, they want to be in control over the automated approaches and correct them if needed. There are more details on the contributions in Chapter 3.

Requirements Intelligence Framework. Requirements Intelligence is a framework that continuously collects, preprocesses, filters, and transforms explicit and implicit user feedback to requirements to generate analytical insights for stakeholders in an integrated interactive visualization.

We motivated the definition of requirements intelligence from the fields of requirements engineering, business intelligence & analytics, as well as software analytics. These three fields have similar goals in mind Business intelligence & analytics and software analytics are well-established fields to provide support in the decision-making process with analytical insights into data. Requirements intelligence is a framework that generates analytics insights for requirements stakeholders based on explicit and implicit user feedback. The two core enablers of the framework, the explicit and implicit user feedback analyses, are composed of the activities *data collection and preprocessing*, *feedback filtering*, and *feedback to requirements*. The framework continuously collects user feedback and prepares it for machine learning-based activities, feedback filtering, and feedback to requirements. Stakeholders have access to the results of the core enablers in an integrated interactive visualization. There are more details on the contributions in Chapter 4.

11.1.2 Core Enablers of Requirements Intelligence

We summarize our findings regarding the explicit and implicit user feedback analyses activities, *feedback filtering* and *feedback to requirements*. For the explicit user feedback analyses, we address Chapter 5 and Chapter 6. For the implicit user feedback analyses, we address Chapter 7 and Chapter 8.

Explicit user feedback analysis: feedback filtering. The objective of this activity is to filter the large amounts of written user feedback. Research shows that popular apps receive daily about 4,000 app reviews and about 31,000 tweets. This feedback contains valuable, requirements-related information like problem reports and inquiries. However, most of the feedback is noisy and irrelevant to requirements stakeholders. Therefore, we propose feedback filtering, which is a requirements intelligence activity that automatically classifies explicit user feedback into the categories problem report, inquiry, and irrelevant. In a crowd-based study, we labeled a total number of 31,000 explicit user feedback entries. After extensive machine learning experiments, we achieved F1 scores of up to .89. Stakeholders can use this approach to filter irrelevant feedback and to distinguish between the two requirements-relevant categories problem report and inquiry. Chapter 5 provides more information about our results.

Explicit user feedback analysis: feedback to requirements. This activity performs analyses of the previously filtered user feedback, which reduces the manual analysis effort for stakeholders. Although we filtered the explicit user feedback into the requirements-relevant categories problem report and inquiry, the amounts of feedback within these categories remain challenging to analyze manually. We, therefore, introduce an approach that can do two things. First, it can extract the features (requirements) users address in explicit user feedback, as well as the features stakeholders, document on app pages. The extraction allows stakeholders to identify the particular features users address and, in combination with the filtering activity, know if the feature belongs to a problem report or an inquiry. Second, we can match the features from the app reviews with the features documented on app pages, which allows stakeholders to understand if users address existing or potential new features. We an extract the stakeholder documented features with an F1 score of .46, and the user described features with an

F1 score of .36, both outperforming existing approaches from the literature. For the matching of the user feedback, we achieved an F1 score of .63 for identifying true positives, and an F1 score .90 for the true negatives. Chapter 6 provides more information about our results.

Implicit user feedback analysis: feedback filtering. Implicit user feedback reports how the user is interacting with an app and puts that interaction into the context. A few users can already generate gigabytes of data and millions of, e.g., click events within an app. In contrast to explicit user feedback, implicit user feedback is not natural language but a collection of categorical and numerical values and, therefore, difficult to interpret by humans. This feedback filtering activity allows stakeholders to filter the feedback based on the usage context, such as the device (e.g., smartphone vs. TV) or the connection (e.g., Wi-Fi vs. mobile). We presented an approach that distinguishes between private and professional device usage automatically. Stakeholders can use this approach to gain insights on how users are using their apps in different contexts. In a crowd-based labeling study, we collected 88,000 implicit user feedback containing context data. Our supervised machine learning approach achieved an F1 score of .94 in a within-users analysis (one classification model per user). For our between-users classification (one classification model trained with several users which we applied on new users), we achieved an F1 score of .95. Chapter 7 provides more information about our results.

Implicit user feedback analysis: feedback to requirements. The next activity of the implicit user feedback analysis is an approach that focuses on user interactions to learn about how users use features. Again, we performed a crowd-based labeling study, in which users provided us the list of features they used within each app session. We performed a within-apps analysis that takes all the labeled interaction events of the users for one app to train and evaluate a machine learning model. We found that depending on the app and the number of labeled data we have, we achieve, on average, an F1 score of .70. The between-apps analysis combined the labeled data of all apps to understand if the user interactions per feature are similar between apps. We found that combining the training data of the apps enabled us to identify the features users use with an F1 score of up to .86. Stakeholders can use the output of the previous, and this activity to understand how users use features in different contexts. Chapter 8 provides more information about our results.

11.1.3 feed.ai

We summarize our contributions regarding the feed.ai prototype (Chapter 9), the integrated interactive visualization (Section 9.4), and its evaluation (Chapter 10).

The requirements intelligence prototype feed.ai. feed.ai is the prototype that we developed based on the requirements intelligence framework. We based the development of feed.ai on requirements that we extracted from Chapter 3, in which we performed qualitative studies to understand stakeholder needs. We documented the requirements in the form of use cases. Besides presenting the requirements, we contribute with an architectural overview of feed.ai, which shows how we realized the requirements intelligence framework. It explains how we can leverage microservices to, e.g., support multiple programming languages that were necessary for feed.ai's development. We further detail on the dynamic models of the microservices for the feedback platform Twitter to illustrate the communication flow between and within the microservices that allow continuous analyses. Stakeholders can extend and deploy the prototype on their own infrastructure (as explained in Chapter 10) as the whole source, and the microservice documentation is open-source [183]. Chapter 9 documents all technical decisions and outcomes of feed.ai.

Integrated interactive visualization. The integrated interactive visualization is the interface for stakeholders using the requirements intelligence prototype feed.ai. The integrated interactive visualization stands for combining both types of user feedback in one single place (integrated), which allows stakeholders to filter for, e.g., specific time frames or the usage context (interactive) in a standalone web-based tool (visualization). The main features are a dashboard, focus views, a user interaction view, as well as a competitor comparison. It presents a dashboard with descriptive analytics showcasing, for example, when and how often do users provide feedback. We also included one view dedicated to problem reports and one dedicated to inquiries (focus views) to allow stakeholders getting in-depth analytics insights into, e.g., the original user feedback, the features users mention, and control mechanisms to correct the automated approaches. In the user interaction view, stakeholders can filter for the usage context (implicit user feedback filtering activity) and automatically extracted features (explicit user feedback to requirements activity) to analyze how users use certain features. The competitor comparison view allows stakeholders to compare multiple apps to understand how users perceive the apps by presenting descriptive analytics results. Stakeholders can either compare multiple apps of their organization or compare their apps with those of competitors. The integrated interactive visualization of Section 9.4 is not fully implemented and an extension of the prototype presented in Chapter 10, which is completely open-source.

Evaluation of feed.ai. Chapter 10 presents the evaluation of feed.ai within the European Horizon 2020 project OpenReq. Within that project, we iteratively developed and evaluated the feed.ai prototype over 12 months with a major telecommunication company based in Italy. During that time, we were able to improve feed.ai in five iterations and evaluated it with methods like surveys and interviews. In total, we could include 15 stakeholders in the evaluation (depending on the iteration, we had two to ten stakeholders available). During the evaluation, we focused on getting qualitative insights to understand why the stakeholders liked or disliked particular features of feed.ai. After the fifth iteration, we report the quantitative results of a survey, including ten anonymous stakeholders from the marketing, customer care, and network departments. The survey results show that the stakeholders rated *feed.ai*'s functionality, on average, with 4.1/5 and its usability with 4.3/5. Overall the stakeholders found that feed.ai presents meaningful information. The stakeholders agreed with 92% of the automated filtering results indicating high accuracy. They further found requirements intelligence beneficial for departments working with user feedback like customer care, marketing, and technology innovation. As a result, *feed.ai* helped stakeholders to reduce 70% of their time spent on analyzing user feedback, indicating a high effectiveness of our approach. We describe the evaluation context, setting, and its results for the five iterations in more detail in Chapter 10.

11.2 Future Work

We introduce some ideas for future work concerning research directions and prototype improvements.

Research directions. Requirements intelligence arose from requirements stakeholder needs. Future work could research how stakeholders can combine traditional requirements engineering and requirements intelligence. We discussed how stakeholders could use requirements intelligence in agile requirements engineering activities but did not research if that integration is feasible nor if stakeholders find that beneficial.

The chapters concerning the explicit and implicit user feedback activities utilized crowdsourcing studies and machine learning approaches to achieve their objectives. For each activity, we presented one empirically evaluated approach, which future work can improve by, e.g., implementing new techniques. Yet, there are several ways to achieve these activities. For that reason, we discussed alternative ways of realizing the activity found in related work.

For filtering explicit feedback, we can imagine automated approaches that categorize user feedback into other categories than problem reports and inquiries, such as user experience and rating. Alternatively, future work could introduce more fine-grained categories or topic analyses. Researchers could also investigate the usefulness of feedback from other platforms like blogs, forums, and Reddit comments.

For filtering implicit feedback, future research could consider other user context scenarios meaningful for stakeholders. One example is to distinguish between computing environment contexts like filtering feedback regarding software's performance under stress. Stakeholders could then analyze how the software performs and how users experience it during sales or holidays.

The activity explicit feedback to requirements focuses on functional requirements in user feedback and app stores. Future research could integrate automated approaches for non-functional requirements and matching the features addressed in user feedback with sources like issue trackers or requirements specification documents.

For the implicit feedback to requirements approach, future research could further develop and integrate automated approaches for UI performance testing, which would provide insights into non-functional requirements like usability. These approaches could generate automated reports validating non-functional requirements.

Prototype improvements. The current state of the open-sourced prototype does not include all the features we introduced. Future work can extend the feed.ai by, for example, completing the implementation and including the previously reported examples for alternatives implementations we discussed in the part *Core Enablers for Requirements Intelligence.*

We evaluated feed.ai with a major telecommunication company instead of an app development company as we only had long-term access to stakeholders from that domain. We applied requirements intelligence in that domain successfully. Nonetheless, the consequence of that decision is that we could not evaluate all facets of the requirements intelligence framework, i.e., the implicit user feedback analyses. Future work should apply the requirements intelligence framework in a company that can use the whole framework for a more complete evaluation.

Part IV

Appendencies

Appendix A

Implicit User Feedback Analysis: Feedback Filtering

A.1 Complete Machine Learning Benchmarks

Table A.1: Accuracy of evaluated classifiers for all participants using the full feature set with 10-fold cross validation.

		Decision Tree			Decision Table			Naive Bayes			LibSVM		
p#	**Session Type**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
	Private	0.997	0.998	**0.998**	0.994	0.999	0.997	0.990	0.996	0.993	0.977	1.000	0.988
p1	Professional	0.927	0.884	**0.905**	0.943	0.767	0.846	0.758	0.581	0.658	0.000	0.000	0.000
	Private	0.997	0.999	**0.998**	0.996	0.998	0.997	0.995	0.960	0.977	0.991	0.996	0.993
p2	Professional	0.969	0.941	**0.955**	0.947	0.919	0.933	0.494	0.885	0.634	0.904	0.786	0.841
	Private	0.965	0.958	**0.962**	0.863	0.931	0.895	0.714	0.784	0.748	0.872	0.805	0.837
p3	Professional	0.974	0.978	**0.976**	0.954	0.907	0.930	0.857	0.804	0.829	0.884	0.926	0.904
	Private	0.917	0.972	**0.944**	0.898	0.983	0.938	0.947	0.806	0.871	0.920	0.914	0.917
p4	Professional	0.891	0.721	**0.797**	0.922	0.646	0.760	0.583	0.857	0.694	0.733	0.748	0.741
	Private	0.984	0.998	**0.991**	0.962	0.995	0.978	0.969	0.953	0.961	0.951	1.000	0.975
p5	Professional	0.957	0.739	**0.834**	0.833	0.378	0.520	0.413	0.521	0.461	1.000	0.193	0.324
	Private	0.989	0.997	**0.993**	0.973	0.995	0.984	0.985	0.864	0.920	0.970	1.000	0.985
p6	Professional	0.861	0.646	**0.738**	0.462	0.125	0.197	0.117	0.583	0.195	0.000	0.000	0.000
	Private	0.970	0.994	**0.982**	0.952	0.983	0.967	0.949	0.887	0.917	0.945	0.990	0.967
p7	Professional	0.963	0.832	**0.893**	0.885	0.726	0.798	0.543	0.737	0.625	0.929	0.684	0.788
	Private	0.988	0.989	**0.988**	0.957	0.988	0.972	0.872	0.797	0.833	0.906	0.978	0.941
p8	Professional	0.961	0.959	**0.960**	0.952	0.845	0.895	0.456	0.592	0.516	0.893	0.648	0.751
	Private	0.955	0.948	**0.951**	0.888	0.948	0.917	0.674	0.672	0.673	0.916	0.728	0.812
p9	Professional	0.972	0.976	**0.974**	0.971	0.936	0.954	0.826	0.827	0.826	0.870	0.965	0.915
	Private	0.997	0.999	**0.998**	0.994	0.997	0.996	0.987	0.909	0.946	0.981	0.998	0.989
p10	Professional	0.989	0.946	**0.967**	0.952	0.899	0.925	0.328	0.795	0.465	0.938	0.654	0.771
	Private	0.997	1.000	**0.999**	0.978	1.000	0.989	0.980	0.980	0.980	0.983	0.987	0.985
p11	Professional	1.000	0.990	**0.995**	1.000	0.913	0.954	0.923	0.923	0.923	0.948	0.933	0.941
	Private	0.983	0.994	**0.988**	0.969	1.000	0.984	0.963	0.955	0.959	0.945	0.994	0.969
p12	Professional	0.947	0.863	**0.903**	0.998	0.750	0.857	0.669	0.713	0.690	0.925	0.549	0.690
	Private	0.978	0.987	**0.983**	0.952	0.985	0.968	0.919	0.880	0.898	0.946	0.949	0.946
Mean	Professional	0.951	0.873	**0.908**	0.902	0.734	0.797	0.581	0.735	0.626	0.752	0.591	0.639

Table A.2: Accuracy of evaluated classifiers for all participants using the minimized feature set (app, day of the week, time of the day, Wi-Fi encryption, and number of available Wi-Fi networks) with 10-fold cross validation.

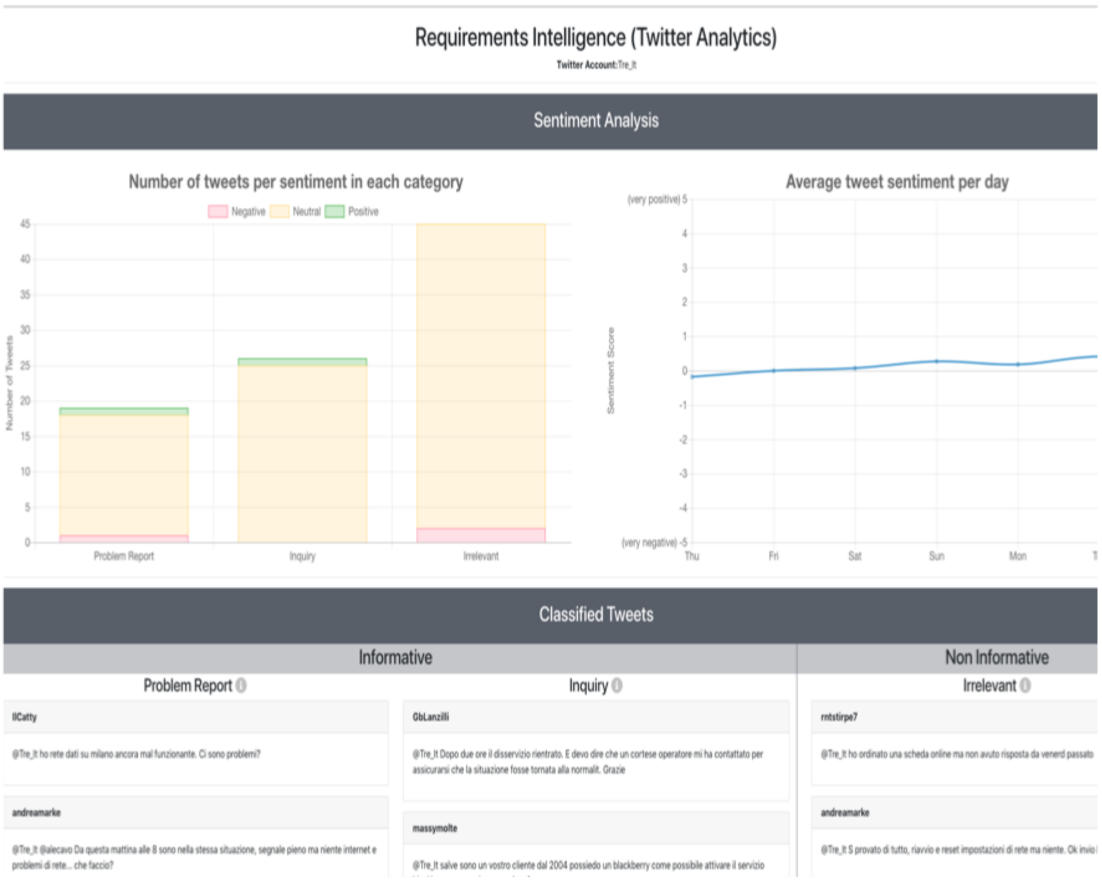
		Decision Tree			**Decision Table**			**Naive Bayes**			**LibSVM**		
p#	**Session Type**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
	Private	0.997	0.998	**0.998**	0.994	0.998	0.996	0.989	0.997	0.993	0.987	1.000	0.994
p1	Professional	0.927	0.884	**0.905**	0.889	0.744	0.810	0.793	0.535	0.639	1.000	0.465	0.635
	Private	0.998	0.998	**0.998**	0.997	0.997	0.997	0.995	0.969	0.982	0.993	0.995	0.994
p2	Professional	0.949	0.946	**0.947**	0.927	0.935	0.931	0.566	0.901	0.695	0.884	0.837	0.860
	Private	0.964	0.951	**0.957**	0.851	0.919	0.884	0.730	0.740	0.735	0.919	0.886	0.902
p3	Professional	0.970	0.978	**0.974**	0.947	0.900	0.923	0.836	0.829	0.833	0.930	0.951	0.941
	Private	0.916	0.970	**0.942**	0.909	0.972	0.940	0.926	0.894	0.910	0.919	0.907	0.913
p4	Professional	0.883	0.721	**0.794**	0.887	0.694	0.779	0.699	0.776	0.735	0.719	0.748	0.733
	Private	0.975	0.999	**0.987**	0.959	0.997	0.977	0.956	0.981	0.968	0.954	0.999	0.976
p5	Professional	0.973	0.597	**0.740**	0.867	0.328	0.476	0.493	0.294	0.368	0.968	0.252	0.400
	Private	0.990	0.997	**0.994**	0.974	0.995	0.984	0.973	0.977	0.975	0.972	1.000	0.986
p6	Professional	0.868	0.688	**0.767**	0.467	0.146	0.222	0.146	0.125	0.135	1.000	0.083	0.154
	Private	0.965	0.994	**0.979**	0.944	0.973	0.958	0.958	0.952	0.955	0.952	0.989	0.970
p7	Professional	0.962	0.800	**0.874**	0.823	0.684	0.747	0.745	0.768	0.756	0.920	0.726	0.812
	Private	0.986	0.989	**0.988**	0.955	0.989	0.972	0.859	0.906	0.881	0.934	0.981	0.957
p8	Professional	0.961	0.952	**0.957**	0.955	0.839	0.893	0.593	0.480	0.531	0.919	0.760	0.832
	Private	0.959	0.952	**0.956**	0.889	0.944	0.916	0.738	0.689	0.713	0.921	0.774	0.842
p9	Professional	0.975	0.978	**0.977**	0.969	0.937	0.953	0.840	0.870	0.855	0.889	0.965	0.926
	Private	0.997	0.999	**0.998**	0.994	0.998	0.996	0.984	0.994	0.989	0.986	0.997	0.991
p10	Professional	0.985	0.938	**0.961**	0.956	0.896	0.925	0.876	0.705	0.781	0.925	0.755	0.831
	Private	0.993	1.000	**0.997**	0.989	1.000	0.995	0.974	0.984	0.979	0.980	0.987	0.983
p11	Professional	1.000	0.974	**0.987**	1.000	0.959	0.979	0.936	0.897	0.916	0.947	0.923	0.935
	Private	0.990	0.994	**0.992**	0.970	1.000	0.985	0.958	0.962	0.960	0.964	0.994	0.979
p12	Professional	0.948	0.919	**0.933**	1.000	0.756	0.861	0.693	0.675	0.684	0.936	0.709	0.807
	Private	0.977	0.987	**0.982**	0.952	0.982	0.967	0.920	0.920	0.920	0.957	0.959	0.957
Mean	Professional	0.950	0.864	**0.901**	0.890	0.735	0.792	0.685	0.655	0.661	0.920	0.681	0.739

Appendix B

Evaluation

B.1 First Prototype Iteration

Screenshots



llCatty

@Tre_It ho rete dati su milano ancora mal funzionante. Ci sono problemi?

andreamarke

@Tre_It @alecavo Da questa mattina alle 8 sono nella stessa situazione, segnale pieno ma niente internet e problemi di rete... che faccio?

GbLanzilli

@Tre_It Dopo due ore il disservizio rientrato. E devo dire che un cortese operatore mi ha contattato per assicurarsi che la situazione fosse tornata alla normalit. Grazie

massymolte

@Tre_It salve sono un vostro cliente dal 2004 possiedo un blackberry come possibile attivare il servizio

rntstirpe7

@Tre_It ho ordinato una scheda online ma non avuto risposta da venerd passato

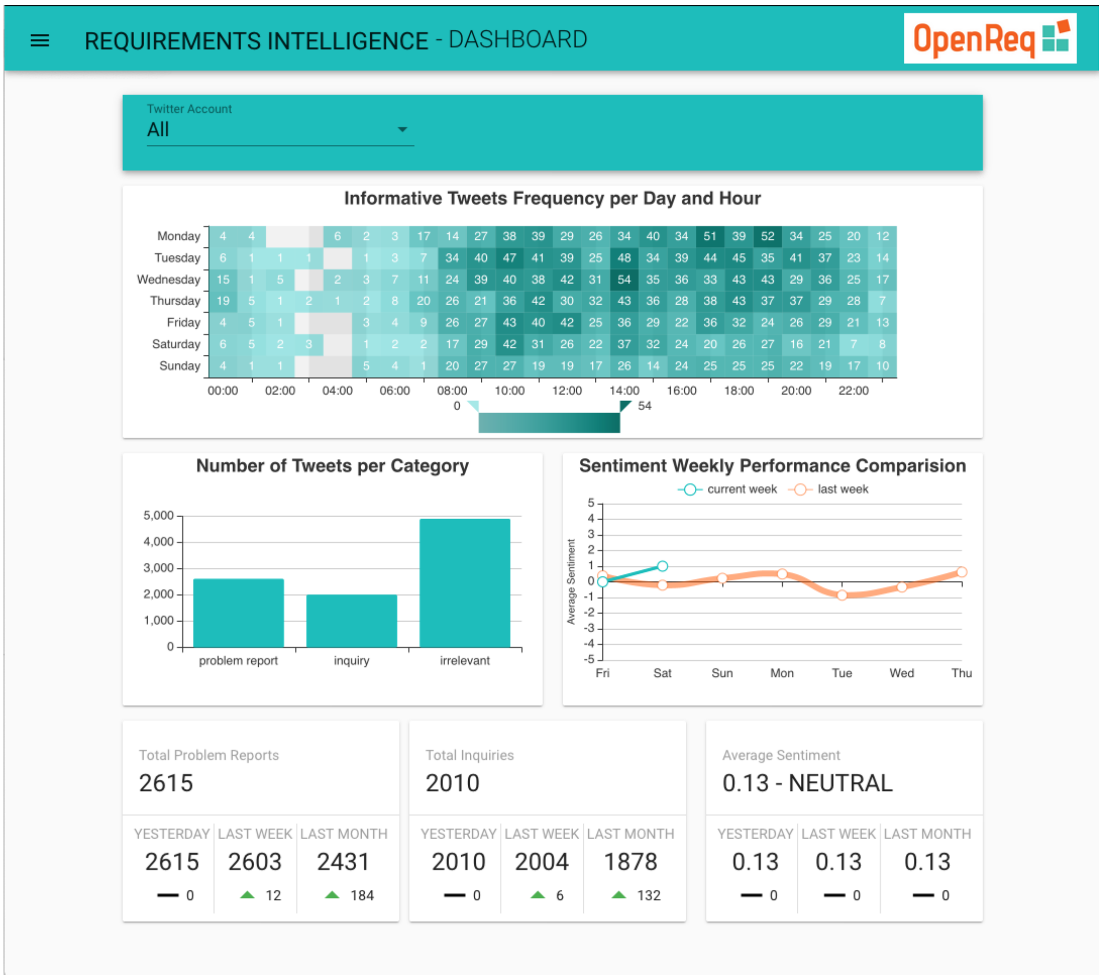
andreamarke

@Tre_It S provato di tutto, riavvio e reset impostazioni di rete ma niente. Ok invio

Figure B.1: First iteration of the requirements intelligence prototype.

B.2 Second Prototype Iteration

Screenshots



Twitter Account
All

Informative Tweets Frequency per Day and Hour

	00:00		02:00		04:00		06:00		08:00		10:00		12:00		14:00		16:00		18:00		20:00		22:00	
Monday	4	4			6	2	3	17	14	27	38	39	29	26	34	40	34	51	39	52	34	25	20	12
Tuesday	6	1	1	1		1	3	7	34	40	47	41	39	25	48	34	39	44	45	35	41	37	23	14
Wednesday	15	1	5		2	3	7	11	24	39	40	38	42	31	54	35	36	33	43	43	29	36	25	17
Thursday	19	5	1	2	1	2	8	20	26	21	36	42	30	32	43	36	28	38	43	37	37	29	28	7
Friday	4	5	1			3	4	9	26	27	43	40	42	25	36	29	22	36	32	24	26	29	21	13
Saturday	6	5	2	3		1	2	2	17	29	42	31	26	22	37	32	24	20	26	27	16	21	7	8
Sunday	4	1	1			5	4	1	20	27	27	19	19	17	26	14	24	25	25	25	22	19	17	10

0 54

Total Problem Reports
2615

YESTERDAY	LAST WEEK	LAST MONTH
2615	2603	2431
— 0	▲ 12	▲ 184

Total Inquiries
2010

YESTERDAY	LAST WEEK	LAST MONTH
2010	2004	1878
— 0	▲ 6	▲ 132

Average Sentiment
0.13 - NEUTRAL

YESTERDAY	LAST WEEK	LAST MONTH
0.13	0.13	0.13
— 0	— 0	— 0

Figure B.2: Second iteration: dashboard view.

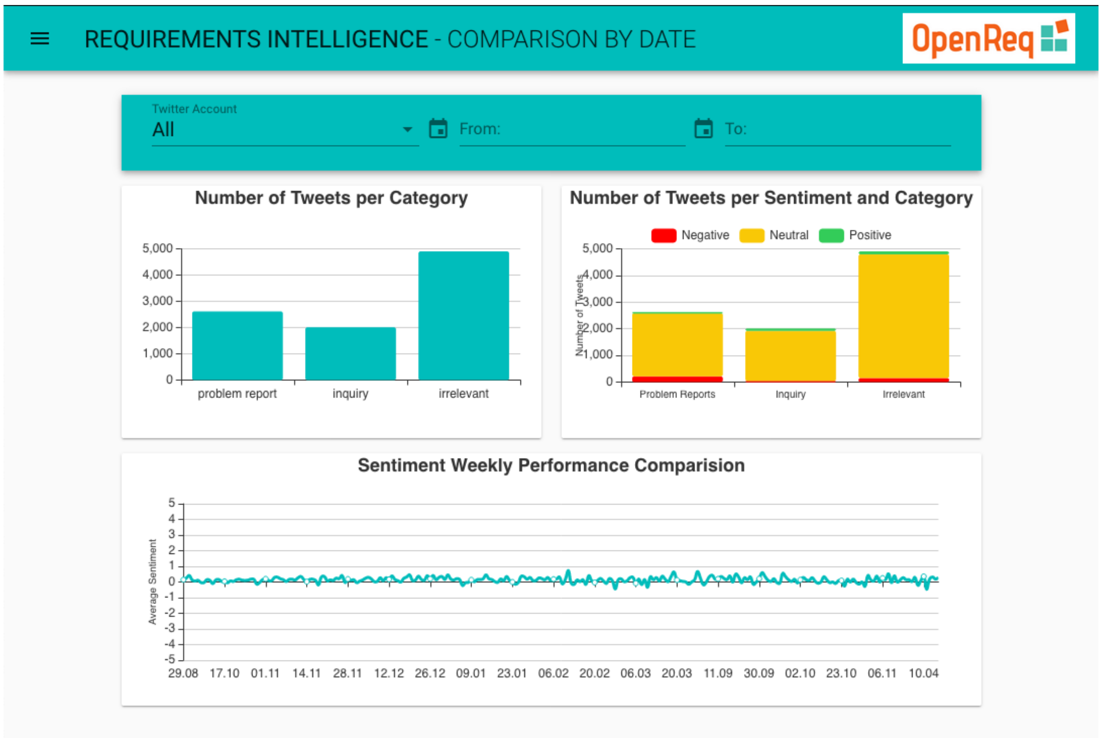
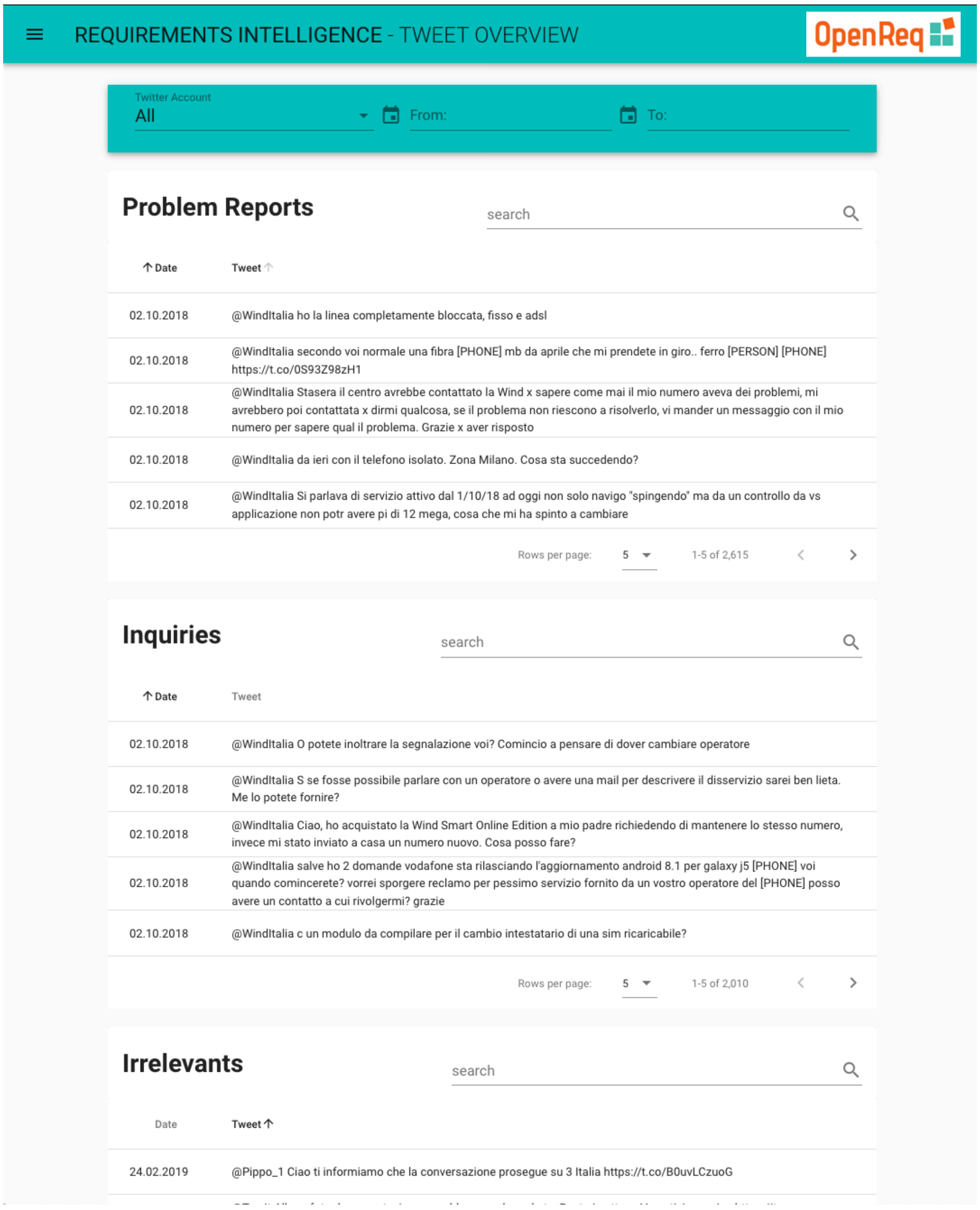

REQUIREMENTS INTELLIGENCE - COMPARISON BY DATE
OpenReq
Twitter Account
All
From:
To:
Number of Tweets per Category
problem report
inquiry
irrelevant
Number of Tweets per Sentiment and Category
Negative
Neutral
Positive
Number of Tweets
Problem Reports
Inquiry
Irrelevant
Sentiment Weekly Performance Comparision
Average Sentiment

Figure B.3: Second iteration: comparison view.



Problem Reports

Date	Tweet
02.10.2018	@WindItalia ho la linea completamente bloccata, fisso e adsl
02.10.2018	@WindItalia secondo voi normale una fibra [PHONE] mb da aprile che mi prendete in giro.. ferro [PERSON] [PHONE] https://t.co/0S93Z98zH1
02.10.2018	@WindItalia Stasera il centro avrebbe contattato la Wind x sapere come mai il mio numero aveva dei problemi, mi avrebbero poi contattata x dirmi qualcosa, se il problema non riescono a risolverlo, vi mander un messaggio con il mio numero per sapere qual il problema. Grazie x aver risposto
02.10.2018	@WindItalia da ieri con il telefono isolato. Zona Milano. Cosa sta succedendo?
02.10.2018	@WindItalia Si parlava di servizio attivo dal 1/10/18 ad oggi non solo navigo "spingendo" ma da un controllo da vs applicazione non potr avere pi di 12 mega, cosa che mi ha spinto a cambiare

Rows per page: 5 1-5 of 2,615

Inquiries

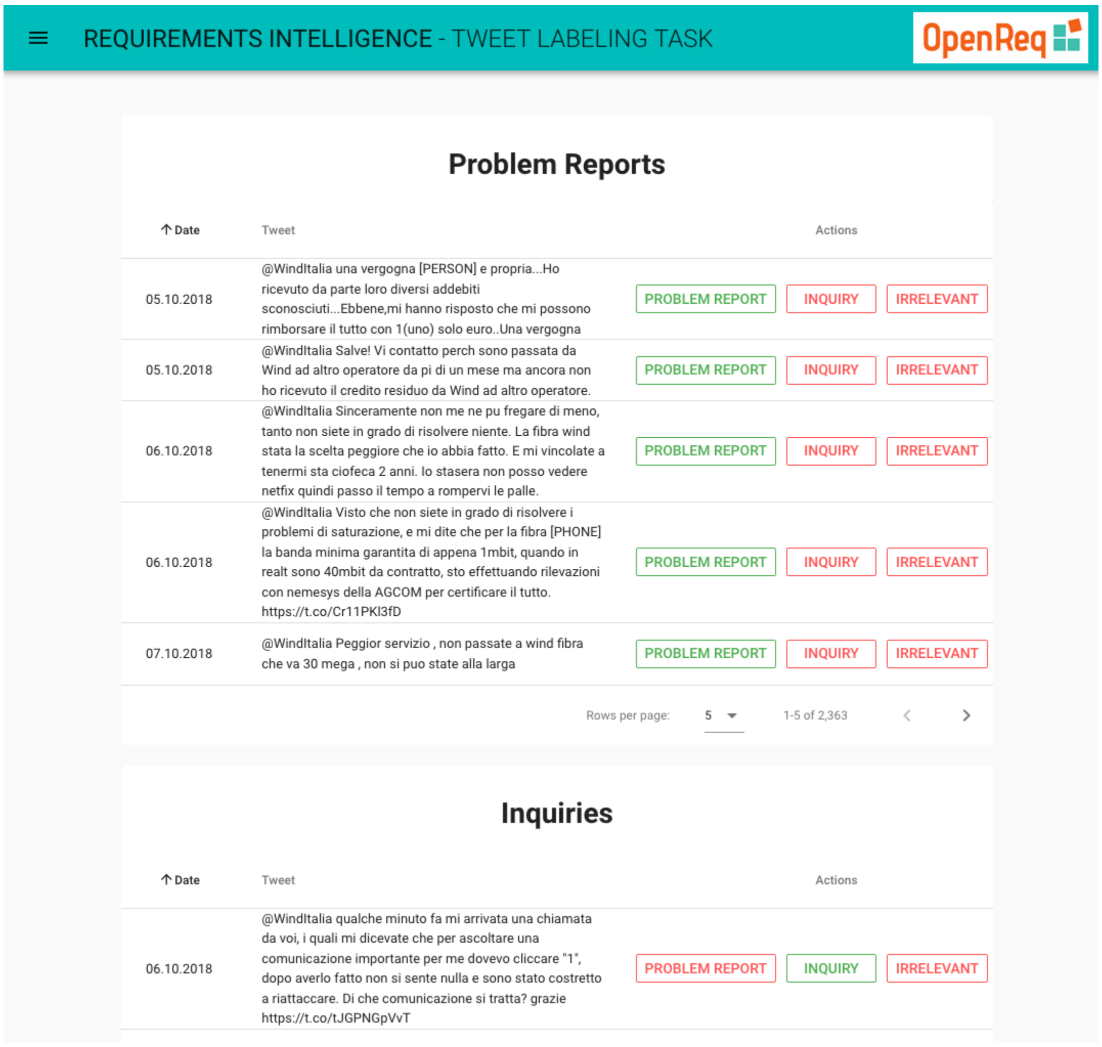
Date	Tweet
02.10.2018	@WindItalia O potete inoltrare la segnalazione voi? Comincio a pensare di dover cambiare operatore
02.10.2018	@WindItalia S se fosse possibile parlare con un operatore o avere una mail per descrivere il disservizio sarei ben lieta. Me lo potete fornire?
02.10.2018	@WindItalia Ciao, ho acquistato la Wind Smart Online Edition a mio padre richiedendo di mantenere lo stesso numero, invece mi stato inviato a casa un numero nuovo. Cosa posso fare?
02.10.2018	@WindItalia salve ho 2 domande vodafone sta rilasciando l'aggiornamento android 8.1 per galaxy j5 [PHONE] voi quando comincerete? vorrei sporgere reclamo per pessimo servizio fornito da un vostro operatore del [PHONE] posso avere un contatto a cui rivolgermi? grazie
02.10.2018	@WindItalia c un modulo da compilare per il cambio intestatario di una sim ricaricabile?

Rows per page: 5 1-5 of 2,010

Irrelevants

Date	Tweet
24.02.2019	@Pippo_1 Ciao ti informiamo che la conversazione prosegue su 3 Italia https://t.co/B0uvLCzuoG

Figure B.4: Second iteration: tweet list view.



Problem Reports

↑ Date	Tweet	Actions		
05.10.2018	@WindItalia una vergogna [PERSON] e propria...Ho ricevuto da parte loro diversi addebiti sconosciuti...Ebbene,mi hanno risposto che mi possono rimborsare il tutto con 1(uno) solo euro..Una vergogna	PROBLEM REPORT	INQUIRY	IRRELEVANT
05.10.2018	@WindItalia Salve! Vi contatto perch sono passata da Wind ad altro operatore da pi di un mese ma ancora non ho ricevuto il credito residuo da Wind ad altro operatore.	PROBLEM REPORT	INQUIRY	IRRELEVANT
06.10.2018	@WindItalia Sinceramente non me ne pu fregare di meno, tanto non siete in grado di risolvere niente. La fibra wind stata la scelta peggiore che io abbia fatto. E mi vincolate a tenermi sta ciofeca 2 anni. Io stasera non posso vedere netfix quindi passo il tempo a rompervi le palle.	PROBLEM REPORT	INQUIRY	IRRELEVANT
06.10.2018	@WindItalia Visto che non siete in grado di risolvere i problemi di saturazione, e mi dite che per la fibra [PHONE] la banda minima garantita di appena 1mbit, quando in realt sono 40mbit da contratto, sto effettuando rilevazioni con nemesys della AGCOM per certificare il tutto. https://t.co/Cr11PKI3fD	PROBLEM REPORT	INQUIRY	IRRELEVANT
07.10.2018	@WindItalia Peggior servizio , non passate a wind fibra che va 30 mega , non si puo state alla larga	PROBLEM REPORT	INQUIRY	IRRELEVANT

Rows per page: 5 1-5 of 2,363

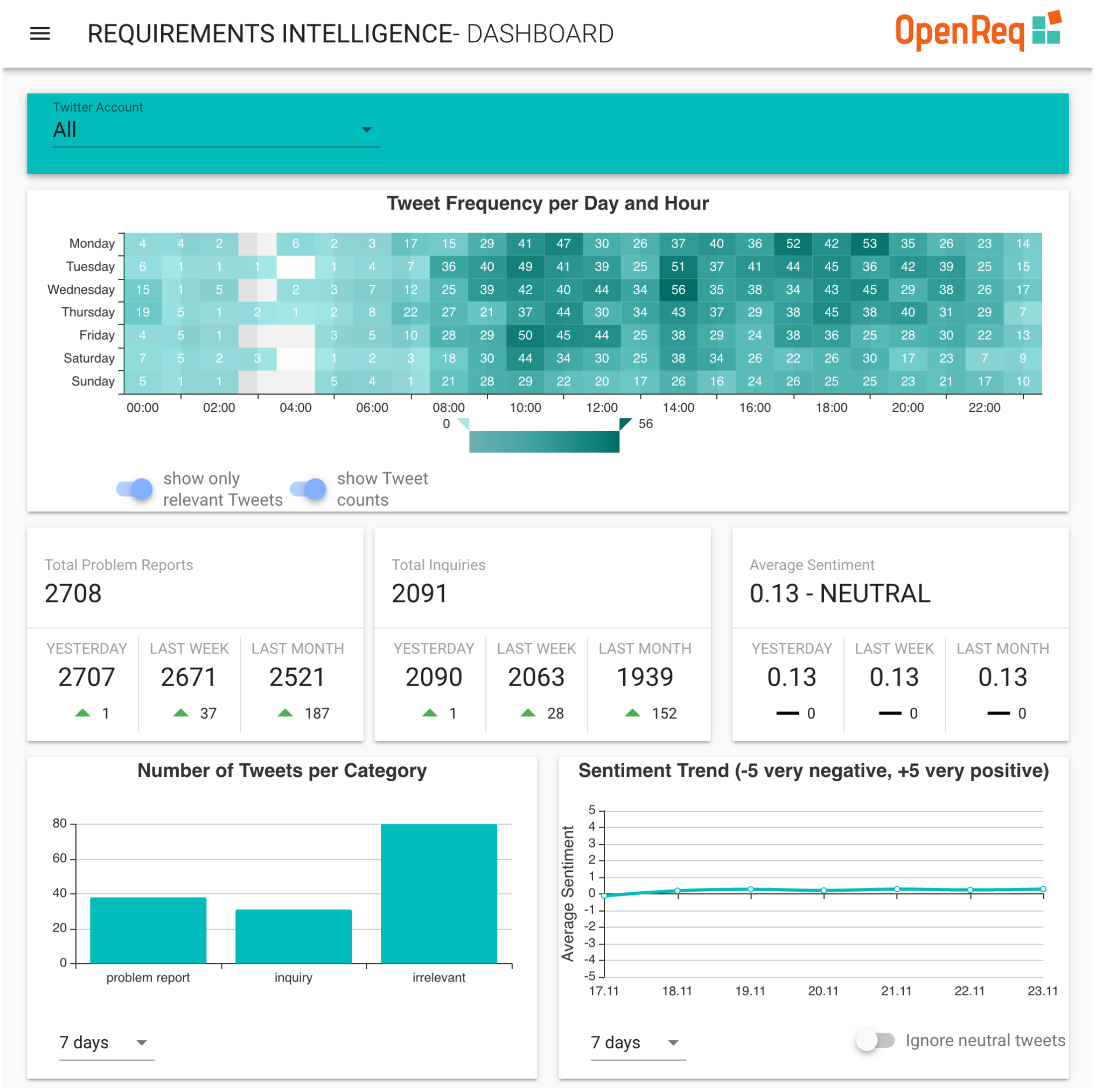
Inquiries

↑ Date	Tweet	Actions		
06.10.2018	@WindItalia qualche minuto fa mi arrivata una chiamata da voi, i quali mi dicevate che per ascoltare una comunicazione importante per me dovevo cliccare "1", dopo averlo fatto non si sente nulla e sono stato costretto a riattaccare. Di che comunicazione si tratta? grazie https://t.co/tJGPNGpVvT	PROBLEM REPORT	INQUIRY	IRRELEVANT

Figure B.5: Second iteration: human control mechanism.

B.3 Third Prototype Iteration

Screenshots



Tweet Frequency per Day and Hour

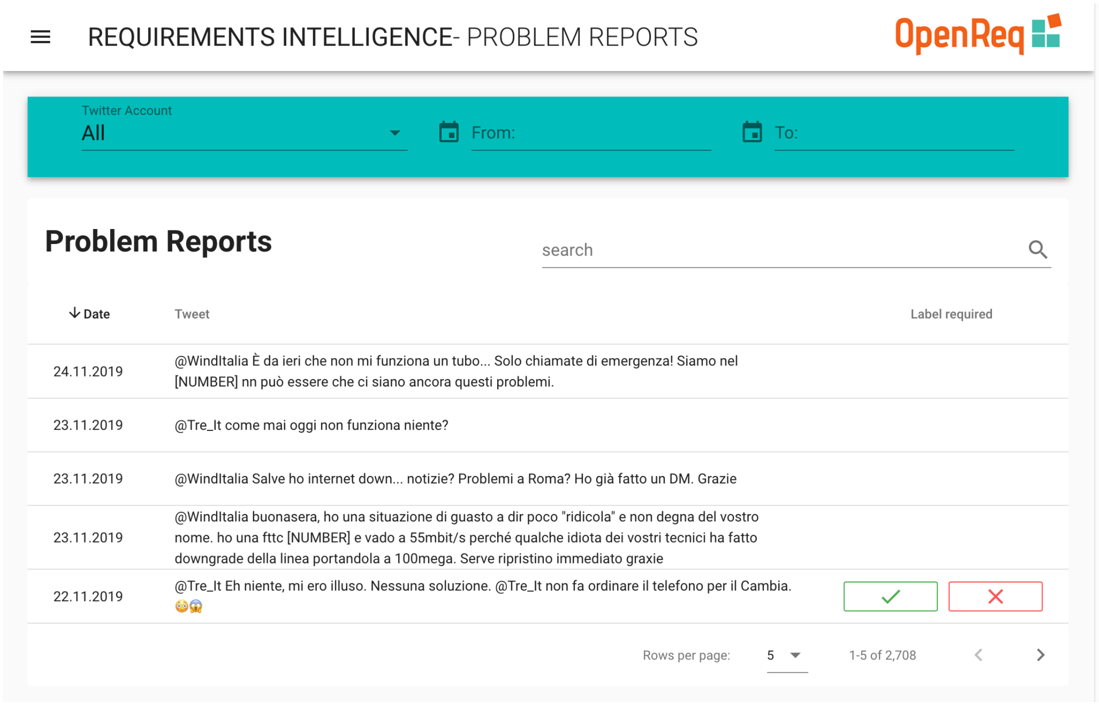
	00:00		02:00		04:00		06:00		08:00		10:00		12:00		14:00		16:00		18:00		20:00		22:00	
Monday	4	4	2		6	2	3	17	15	29	41	47	30	26	37	40	36	52	42	53	35	26	23	14
Tuesday	6	1	1	1		1	4	7	36	40	49	41	39	25	51	37	41	44	45	36	42	39	25	15
Wednesday	15	1	5		2	3	7	12	25	39	42	40	44	34	56	35	38	34	43	45	29	38	26	17
Thursday	19	5	1	2	1	2	8	22	27	21	37	44	30	34	43	37	29	38	45	38	40	31	29	7
Friday	4	5	1			3	5	10	28	29	50	45	44	25	38	29	24	38	36	25	28	30	22	13
Saturday	7	5	2	3		1	2	3	18	30	44	34	30	25	38	34	26	22	26	30	17	23	7	9
Sunday	5	1	1			5	4	1	21	28	29	22	20	17	26	16	24	26	25	25	23	21	17	10

Total Problem Reports		
2708		
YESTERDAY	LAST WEEK	LAST MONTH
2707	2671	2521
▲ 1	▲ 37	▲ 187

Total Inquiries		
2091		
YESTERDAY	LAST WEEK	LAST MONTH
2090	2063	1939
▲ 1	▲ 28	▲ 152

Average Sentiment		
0.13 - NEUTRAL		
YESTERDAY	LAST WEEK	LAST MONTH
0.13	0.13	0.13
— 0	— 0	— 0

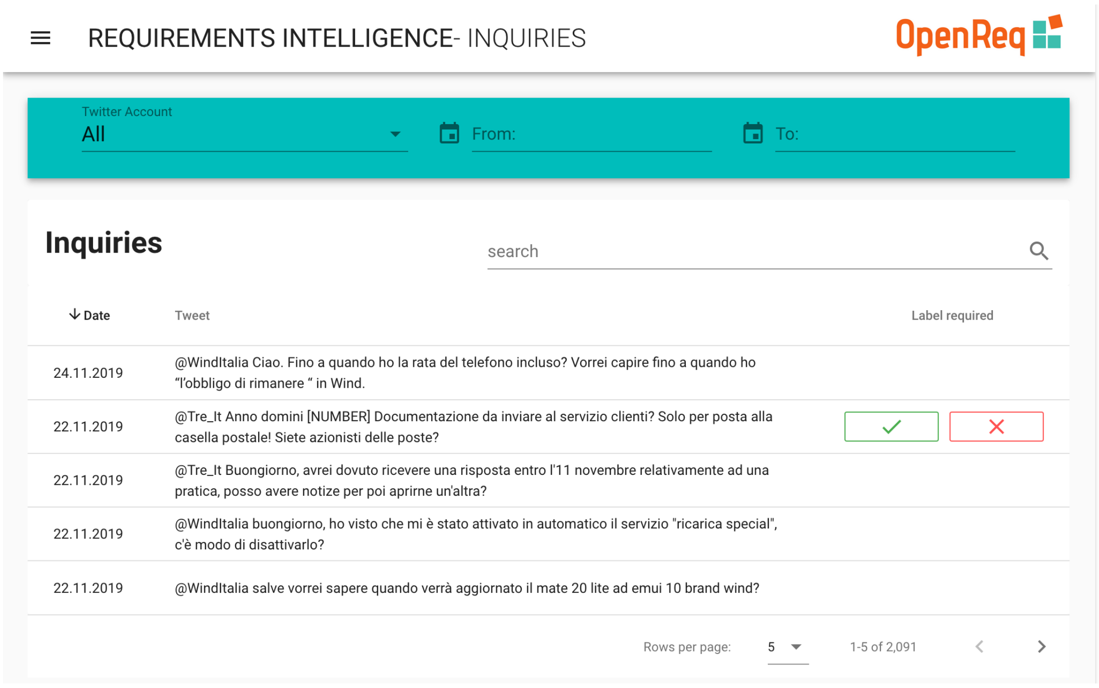
Figure B.6: Third iteration: dashboard view.



Problem Reports

Date	Tweet	Label required
24.11.2019	@WindItalia È da ieri che non mi funziona un tubo... Solo chiamate di emergenza! Siamo nel [NUMBER] nn può essere che ci siano ancora questi problemi.	
23.11.2019	@Tre_It come mai oggi non funziona niente?	
23.11.2019	@WindItalia Salve ho internet down... notizie? Problemi a Roma? Ho già fatto un DM. Grazie	
23.11.2019	@WindItalia buonasera, ho una situazione di guasto a dir poco "ridicola" e non degna del vostro nome. ho una fttc [NUMBER] e vado a 55mbit/s perché qualche idiota dei vostri tecnici ha fatto downgrade della linea portandola a 100mega. Serve ripristino immediato graxie	
22.11.2019	@Tre_It Eh niente, mi ero illuso. Nessuna soluzione. @Tre_It non fa ordinare il telefono per il Cambia. 😳😱	✓ ✕

Figure B.7: Third iteration: problem report focus view.



Inquiries

Date	Tweet	Label required
24.11.2019	@WindItalia Ciao. Fino a quando ho la rata del telefono incluso? Vorrei capire fino a quando ho "l'obbligo di rimanere " in Wind.	
22.11.2019	@Tre_It Anno domini [NUMBER] Documentazione da inviare al servizio clienti? Solo per posta alla casella postale! Siete azionisti delle poste?	✓ ✕
22.11.2019	@Tre_It Buongiorno, avrei dovuto ricevere una risposta entro l'11 novembre relativamente ad una pratica, posso avere notize per poi aprirne un'altra?	
22.11.2019	@WindItalia buongiorno, ho visto che mi è stato attivato in automatico il servizio "ricarica special", c'è modo di disattivarlo?	
22.11.2019	@WindItalia salve vorrei sapere quando verrà aggiornato il mate 20 lite ad emui 10 brand wind?	

Figure B.8: Third iteration: inquiry focus view.

B.4 Fourth Prototype Iteration

Screenshots

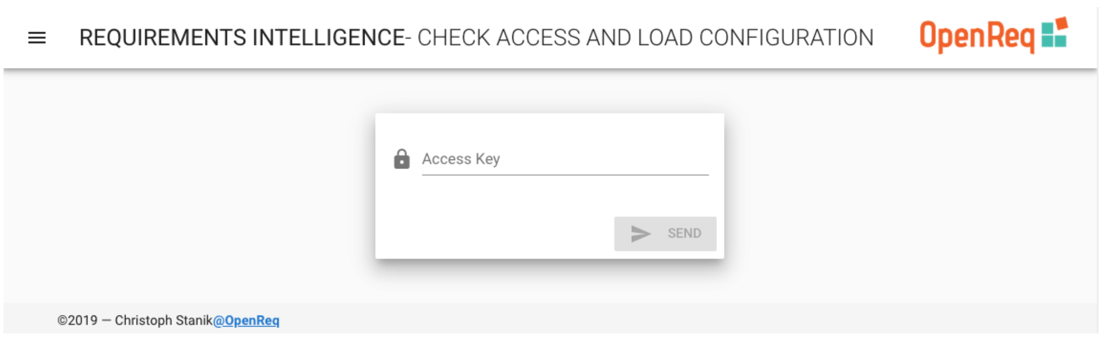
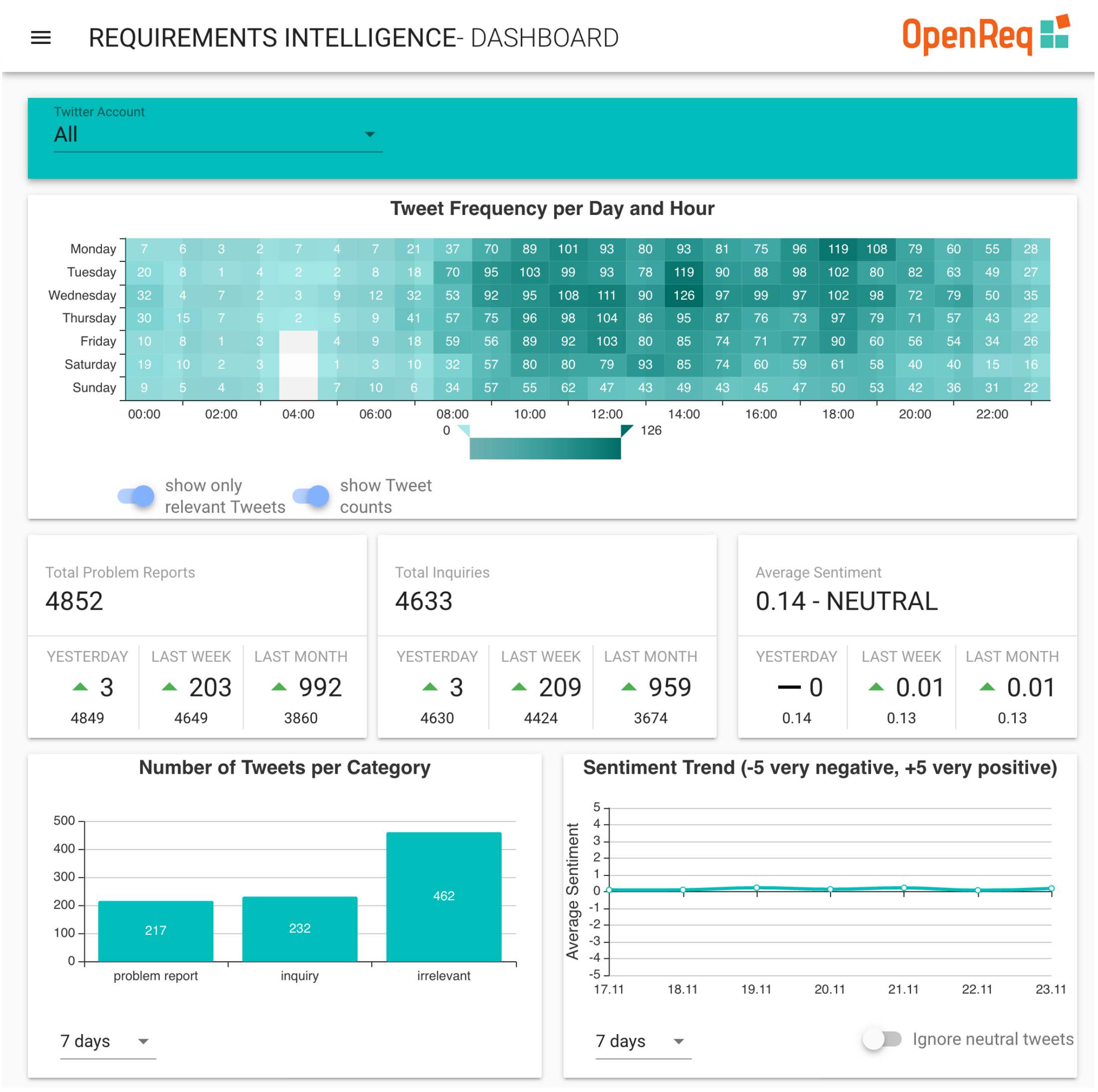

REQUIREMENTS INTELLIGENCE- CHECK ACCESS AND LOAD CONFIGURATION
OpenReq
Access Key
SEND
©2019 — Christoph Stanik@OpenReq

Figure B.9: Fourth iteration: access key view.



REQUIREMENTS INTELLIGENCE- DASHBOARD

Twitter Account: All

Tweet Frequency per Day and Hour

	00:00		02:00		04:00		06:00		08:00		10:00		12:00		14:00		16:00		18:00		20:00		22:00	
Monday	7	6	3	2	7	4	7	21	37	70	89	101	93	80	93	81	75	96	119	108	79	60	55	28
Tuesday	20	8	1	4	2	2	8	18	70	95	103	99	93	78	119	90	88	98	102	80	82	63	49	27
Wednesday	32	4	7	2	3	9	12	32	53	92	95	108	111	90	126	97	99	97	102	98	72	79	50	35
Thursday	30	15	7	5	2	5	9	41	57	75	96	98	104	86	95	87	76	73	97	79	71	57	43	22
Friday	10	8	1	3		4	9	18	59	56	89	92	103	80	85	74	71	77	90	60	56	54	34	26
Saturday	19	10	2	3		1	3	10	32	57	80	80	79	93	85	74	60	59	61	58	40	40	15	16
Sunday	9	5	4	3		7	10	6	34	57	55	62	47	43	49	43	45	47	50	53	42	36	31	22

Total Problem Reports 4852		
YESTERDAY	LAST WEEK	LAST MONTH
▲ 3	▲ 203	▲ 992
4849	4649	3860

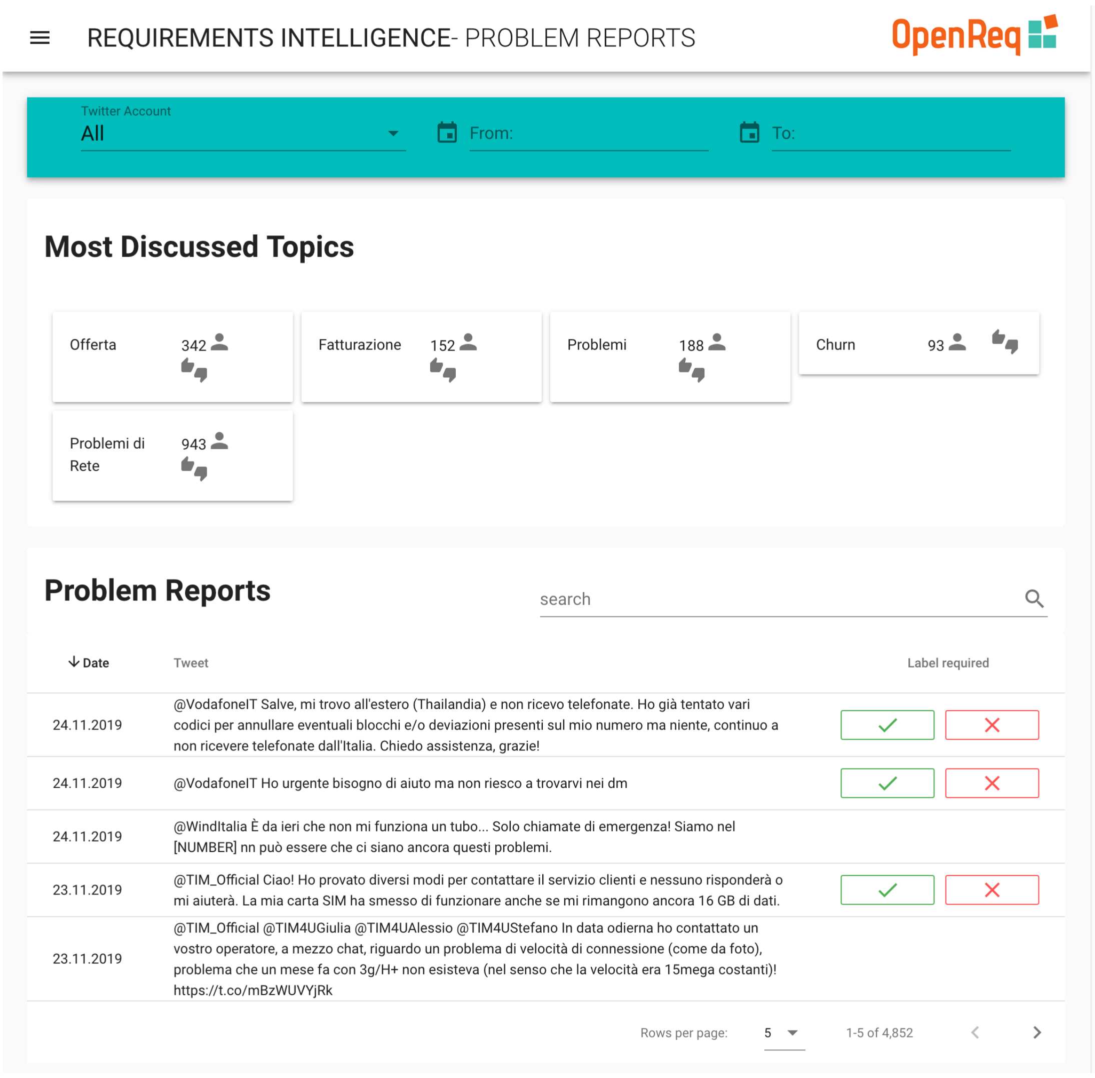
Total Inquiries 4633		
YESTERDAY	LAST WEEK	LAST MONTH
▲ 3	▲ 209	▲ 959
4630	4424	3674

Average Sentiment 0.14 - NEUTRAL		
YESTERDAY	LAST WEEK	LAST MONTH
— 0	▲ 0.01	▲ 0.01
0.14	0.13	0.13

Number of Tweets per Category

Sentiment Trend (-5 very negative, +5 very positive)

Figure B.10: Fourth iteration: dashboard view.



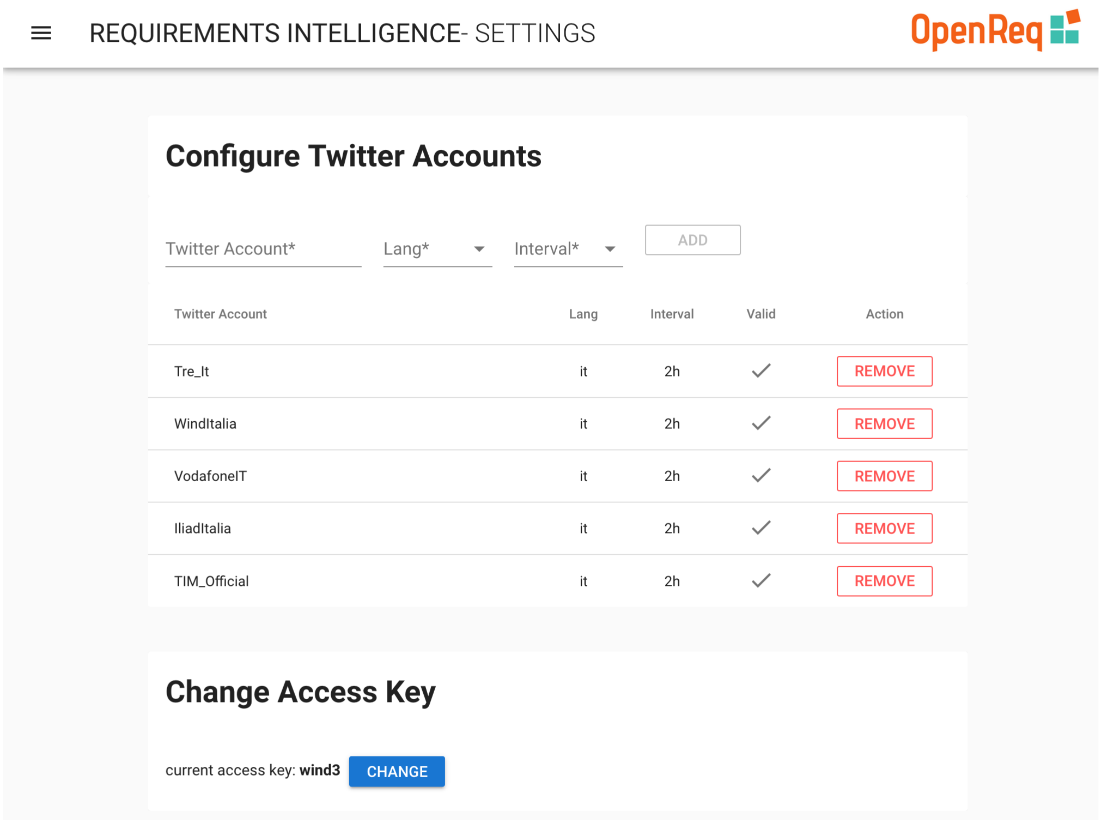
Most Discussed Topics

Offerta	342
Fatturazione	152
Problemi	188
Churn	93
Problemi di Rete	943

Problem Reports

Date	Tweet	Label required
24.11.2019	@VodafoneIT Salve, mi trovo all'estero (Thailandia) e non ricevo telefonate. Ho già tentato vari codici per annullare eventuali blocchi e/o deviazioni presenti sul mio numero ma niente, continuo a non ricevere telefonate dall'Italia. Chiedo assistenza, grazie!	✓ ✗
24.11.2019	@VodafoneIT Ho urgente bisogno di aiuto ma non riesco a trovarvi nei dm	✓ ✗
24.11.2019	@WindItalia È da ieri che non mi funziona un tubo... Solo chiamate di emergenza! Siamo nel [NUMBER] nn può essere che ci siano ancora questi problemi.	
23.11.2019	@TIM_Official Ciao! Ho provato diversi modi per contattare il servizio clienti e nessuno risponderà o mi aiuterà. La mia carta SIM ha smesso di funzionare anche se mi rimangono ancora 16 GB di dati.	✓ ✗
23.11.2019	@TIM_Official @TIM4UGiulia @TIM4UAlessio @TIM4UStefano In data odierna ho contattato un vostro operatore, a mezzo chat, riguardo un problema di velocità di connessione (come da foto), problema che un mese fa con 3g/H+ non esisteva (nel senso che la velocità era 15mega costanti)! https://t.co/mBzWUVYjRk	

Figure B.11: Fourth iteration: problem report focus view.



REQUIREMENTS INTELLIGENCE- SETTINGS

Configure Twitter Accounts

Twitter Account* | Lang* | Interval* | ADD

Twitter Account	Lang	Interval	Valid	Action
Tre_It	it	2h	✓	REMOVE
WindItalia	it	2h	✓	REMOVE
VodafoneIT	it	2h	✓	REMOVE
IliadItalia	it	2h	✓	REMOVE
TIM_Official	it	2h	✓	REMOVE

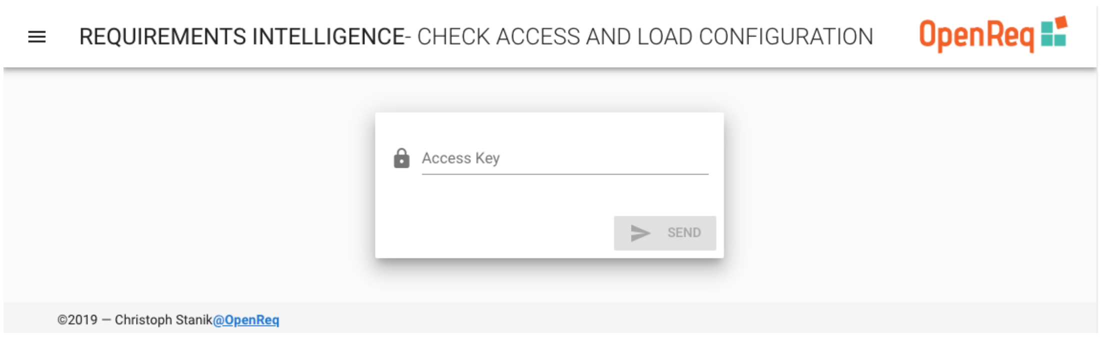
Change Access Key

current access key: **wind3** CHANGE

Figure B.12: Fourth iteration: feedback source configuration view.

B.5 Fifth Prototype Iteration

Screenshots



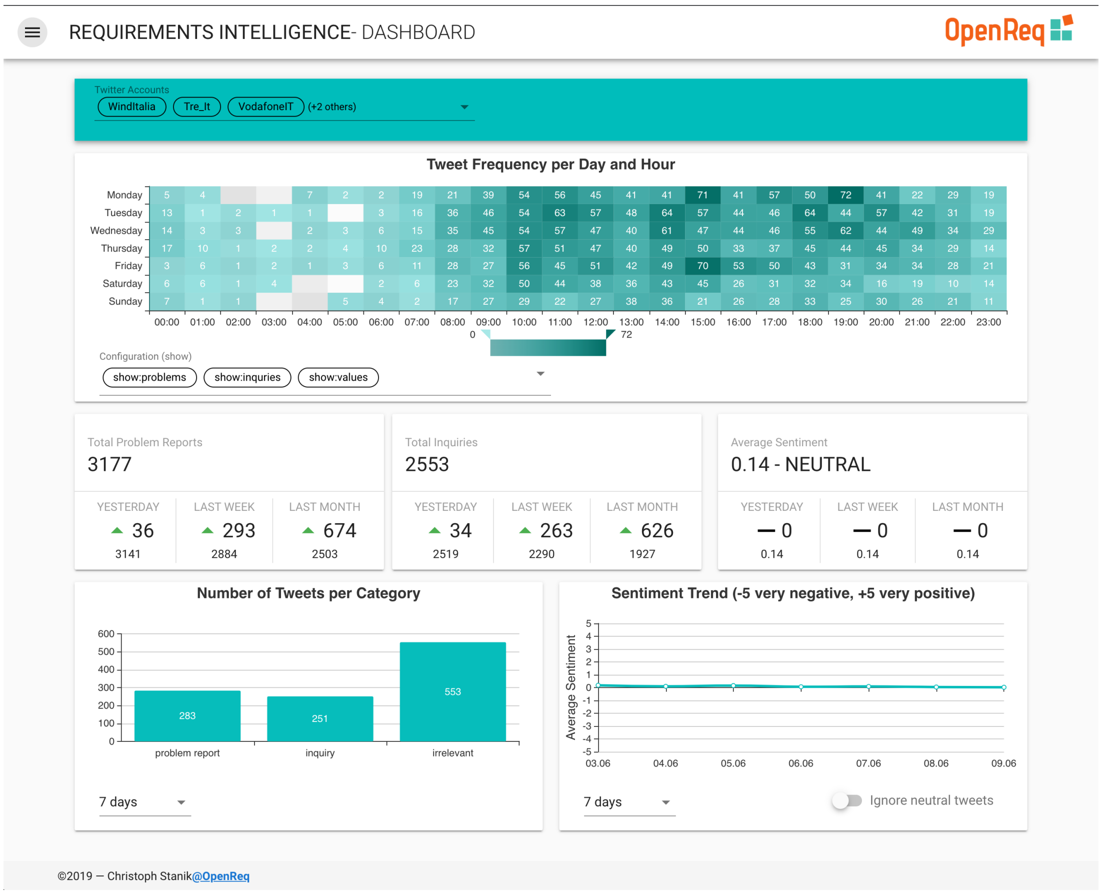
REQUIREMENTS INTELLIGENCE- CHECK ACCESS AND LOAD CONFIGURATION

Access Key

SEND

©2019 — Christoph Stanik@OpenReq

Figure B.13: Fifth iteration: access key view.



REQUIREMENTS INTELLIGENCE- DASHBOARD

OpenReq

Twitter Accounts: WindItalia, Tre_It, VodafoneIT (+2 others)

Tweet Frequency per Day and Hour

	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00
Monday	5	4			7	2	2	19	21	39	54	56	45	41	41	71	41	57	50	72	41	22	29	19
Tuesday	13	1	2	1	1		3	16	36	46	54	63	57	48	64	57	44	46	64	44	57	42	31	19
Wednesday	14	3	3		2	3	6	15	35	45	54	57	47	40	61	47	44	46	55	62	44	49	34	29
Thursday	17	10	1	2	2	4	10	23	28	32	57	51	47	40	49	50	33	37	45	44	45	34	29	14
Friday	3	6	1	2	1	3	6	11	28	27	56	45	51	42	49	70	53	50	43	31	34	34	28	21
Saturday	6	6	1	4			2	6	23	32	50	44	38	36	43	45	26	31	32	34	16	19	10	14
Sunday	7	1	1			5	4	2	17	27	29	22	27	38	36	21	26	28	33	25	30	26	21	11

Configuration (show): show:problems, show:inquries, show:values

Total Problem Reports 3177		
YESTERDAY	LAST WEEK	LAST MONTH
▲ 36	▲ 293	▲ 674
3141	2884	2503

Total Inquiries 2553		
YESTERDAY	LAST WEEK	LAST MONTH
▲ 34	▲ 263	▲ 626
2519	2290	1927

Average Sentiment 0.14 - NEUTRAL		
YESTERDAY	LAST WEEK	LAST MONTH
— 0	— 0	— 0
0.14	0.14	0.14

Number of Tweets per Category

7 days

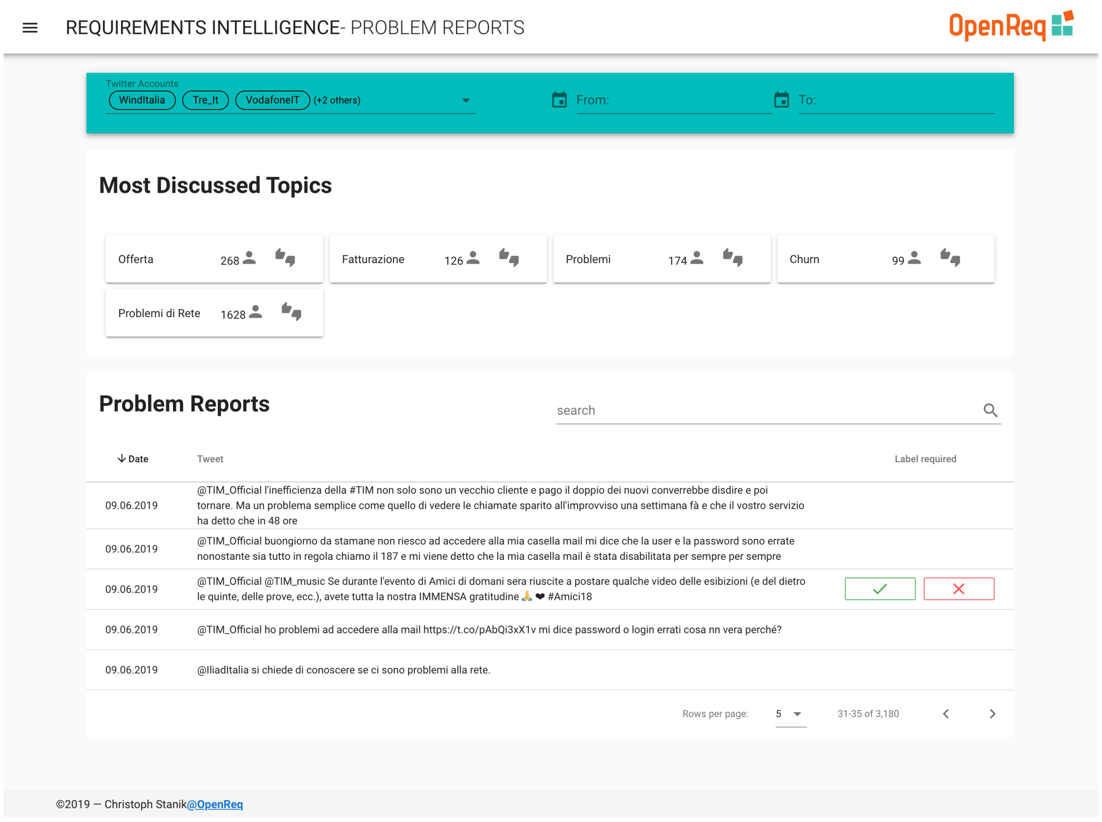
Sentiment Trend (-5 very negative, +5 very positive)

7 days

Ignore neutral tweets

©2019 — Christoph Stanik@OpenReq

Figure B.14: Fifth iteration: dashboard view.



REQUIREMENTS INTELLIGENCE- PROBLEM REPORTS

Twitter Accounts: WindItalia, Tre_It, VodafoneIT (+2 others) — From: — To:

Most Discussed Topics

Offerta	268	
Fatturazione	126	
Problemi	174	
Churn	99	
Problemi di Rete	1628	

Problem Reports

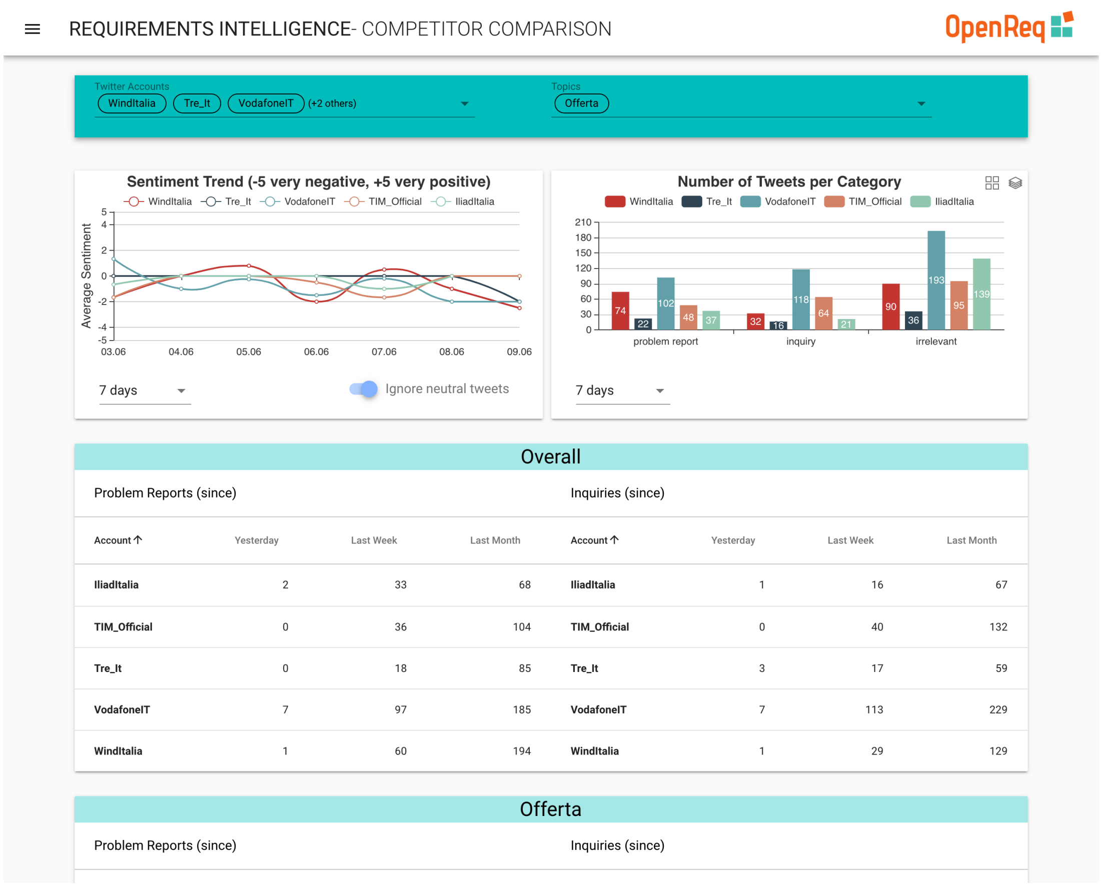
search

↓ Date	Tweet	Label required
09.06.2019	@TIM_Official l'inefficienza della #TIM non solo sono un vecchio cliente e pago il doppio dei nuovi converrebbe disdire e poi tornare. Ma un problema semplice come quello di vedere le chiamate sparito all'improvviso una settimana fà e che il vostro servizio ha detto che in 48 ore	
09.06.2019	@TIM_Official buongiorno da stamane non riesco ad accedere alla mia casella mail mi dice che la user e la password sono errate nonostante sia tutto in regola chiamo il 187 e mi viene detto che la mia casella mail è stata disabilitata per sempre per sempre	
09.06.2019	@TIM_Official @TIM_music Se durante l'evento di Amici di domani sera riuscite a postare qualche video delle esibizioni (e del dietro le quinte, delle prove, ecc.), avete tutta la nostra IMMENSA gratitudine 🙏 ❤ #Amici18	✓ ✗
09.06.2019	@TIM_Official ho problemi ad accedere alla mail https://t.co/pAbQi3xX1v mi dice password o login errati cosa nn vera perché?	
09.06.2019	@IliadItalia si chiede di conoscere se ci sono problemi alla rete.	

Rows per page: 5 — 31-35 of 3,180

©2019 — Christoph Stanik@OpenReq

Figure B.15: Fifth iteration: problem report focus view.


REQUIREMENTS INTELLIGENCE- COMPETITOR COMPARISON

Twitter Accounts: WindItalia, Tre_It, VodafoneIT (+2 others)

Topics: Offerta

Overall

Problem Reports (since)

Account ↑	Yesterday	Last Week	Last Month
IliadItalia	2	33	68
TIM_Official	0	36	104
Tre_It	0	18	85
VodafoneIT	7	97	185
WindItalia	1	60	194

Inquiries (since)

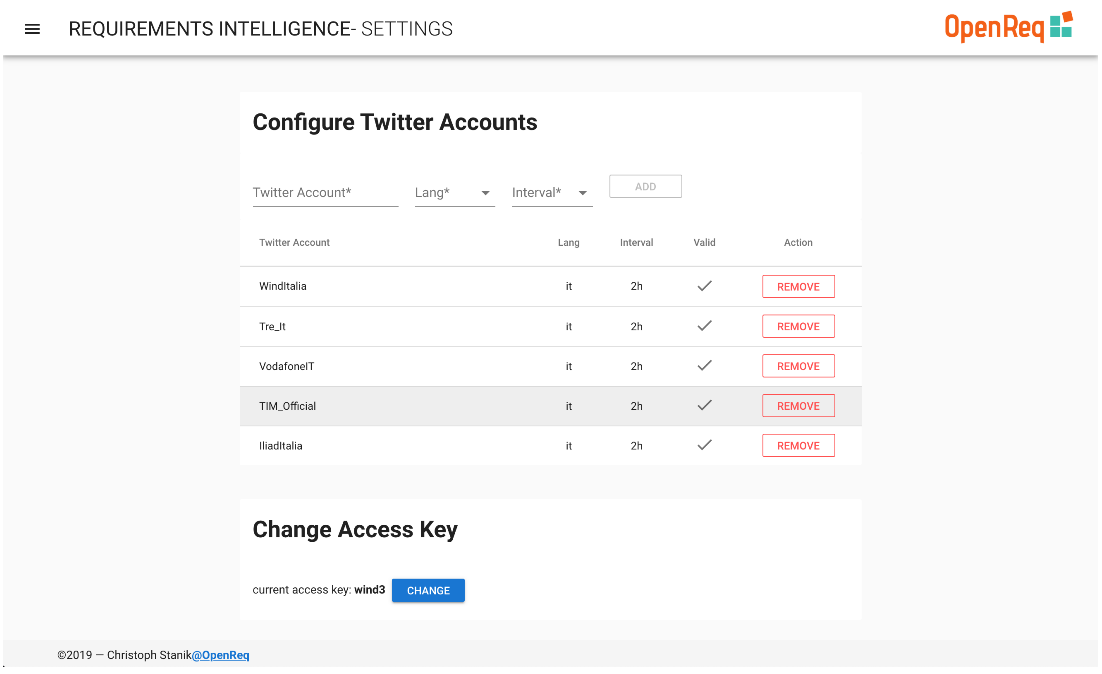
Account ↑	Yesterday	Last Week	Last Month
IliadItalia	1	16	67
TIM_Official	0	40	132
Tre_It	3	17	59
VodafoneIT	7	113	229
WindItalia	1	29	129

Offerta

Problem Reports (since)

Inquiries (since)

Figure B.16: Fifth iteration: competitor comparison view.



Configure Twitter Accounts

Twitter Account* | Lang* | Interval* | ADD

Twitter Account	Lang	Interval	Valid	Action
WindItalia	it	2h	✓	REMOVE
Tre_It	it	2h	✓	REMOVE
VodafoneIT	it	2h	✓	REMOVE
TIM_Official	it	2h	✓	REMOVE
IliadItalia	it	2h	✓	REMOVE

Change Access Key

current access key: **wind3** CHANGE

©2019 — Christoph Stanik@OpenReq

Figure B.17: Fifth iteration: feedback source configuration view.

List of Figures

List of Tables

List of Publications

Here, we list the co-authored publications of the author of this thesis, Christoph Stanik. We sorted the list by the year of publication.

Publications, partly verbatim, included in this work:

- Maalej, W., Kurtanović, Z., Nabil, H., and Stanik, C. “On the automatic classification of app reviews”. In: *Requirements Engineering* 21.3 (2016)
- Johann, T., Stanik, C., Alizadeh B., A. M., and Maalej, W. “Safe: A simple approach for feature extraction from app descriptions and app reviews”. In: *25th IEEE International Requirements Engineering Conference*. IEEE. 2017
- Stanik, C. and Maalej, W. “Requirements Intelligence with OpenReq Analytics”. In: *27th IEEE International Requirements Engineering Conference*. IEEE. 2019
- Stanik, C., Haering, M., and Maalej, W. “Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning”. In: *27th IEEE International Requirements Engineering Conference Workshops*. IEEE. 2019

Additional, not included publications:

- Stanik, C., Montgomery, L., Martens, D., Fucci, D., and Maalej, W. “A Simple NLP-based Approach to Support Onboarding and Retention in Open Source Communities”. In: *34th IEEE International Conference on Software Maintenance and Evolution*. IEEE. 2018
- Fucci, D., Stanik, C., Montgomery, L., Kurtanovic, Z., Johann, T., and Maalej, W. “Research on NLP for RE at the University of Hamburg: A Re-

port." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality Workshops*. 2018

- Felfernig, A., Stettinger, M., Wundara, M., and Stanik, C. "Künstliche Intelligenz in der Öffentlichen Verwaltung". In: *Handbuch E-Government*. Springer, 2019

Bibliography

[1] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. "Sentiment Analysis of Twitter Data". In: *Workshop on Language in Social Media*. Association for Computational Linguistics, 2011.

[2] Aksoy, S. and Haralick, R. M. "Feature normalization and likelihood-based similarity measures for image retrieval". In: *Pattern Recognition Letters* 22.5 (2001).

[3] Al-Subaihin, A. A., Sarro, F., Black, S., Capra, L., Harman, M., Jia, Y., and Zhang, Y. "Clustering Mobile Apps Based on Mined Textual Features". In: *10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Ciudad Real, Spain: ACM, 2016.

[4] Al-Subaihin, A. A. "Software Engineering in the Age of App Stores: Feature-Based Analyses to Guide Mobile Software Engineers". PhD thesis. UCL (University College London), 2019.

[5] Ali, S., Qureshi, M. N., and Abbasi, A. G. "Analysis of BYOD security frameworks". In: *Conference on Information Assurance and Cyber Security*. 2015.

[6] Amazon. *Mechanical Turk*. Accessed December 22, 2019. URL: `https://www.mturk.com/`.

[7] Amazon. *What are Microservices?* Accessed December 9, 2019. URL: `https://aws.amazon.com/microservices/`.

[8] Appbot. *Website*. Accessed December 22, 2019. URL: `https://appbot.co/`.

[9] Apple. *Framework UIKit*. Accessed December 22, 2019. URL: `https://developer.apple.com/documentation/uikit`.

[10] Apple. *Apple Services Performance Partners*. Accessed January 5, 2017. URL: `https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api`.

[11] Arnott, D. and Pervan, G. "A critical analysis of decision support systems research". In: *Journal of Information Technology* 20.2 (2005).

[12] Bailey, K., Nagappan, M., and Dig, D. "Examining User-Developer Feedback Loops in the iOS App Store". In: *52nd Hawaii International Conference on System Sciences*. 2019.

[13] Baker, L. "Observation: A complex research method". In: *Library Trends* 55.1 (2006).

[14] Baltrunas, L., Church, K., Karatzoglou, A., and Oliver, N. "Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild". In: *arXiv preprint arXiv:1505.03014* (2015).

[15] Bano, M. and Zowghi, D. "User involvement in software development and system success: a systematic literature review". In: *17th International Conference on Evaluation and Assessment in Software Engineering*. ACM. 2013.

[16] Banovic, N., Brant, C., Mankoff, J., and Dey, A. "ProactiveTasks: the short of mobile device use sessions". In: *16th International Conference on Human Computer Interaction with Mobile Devices & Services*. ACM. 2014.

[17] Begel, A. and Zimmermann, T. "Analyze this! 145 questions for data scientists in software engineering". In: *36th International Conference on Software Engineering*. ACM. 2014.

[18] Ben-Hur, A. and Weston, J. "Data Mining Techniques for the Life Sciences". In: ed. by O. Carugo and F. Eisenhaber. Humana Press, 2010. Chap. A User's Guide to Support Vector Machines.

[19] Bergstra, J. and Bengio, Y. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012).

[20] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. 2011.

[21] Berkel, N. van, Luo, C., Anagnostopoulos, T., Ferreira, D., Goncalves, J., Hosio, S., and Kostakos, V. “A Systematic Assessment of Smartphone Usage Gaps”. In: *Conference on Human Factors in Computing Systems.* ACM, 2016.

[22] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. “What makes a good bug report?” In: *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2008.

[23] Bishop, C. M. *Pattern recognition and machine learning.* springer, 2006.

[24] Blythe, J. M., Coventry, L., and Little, L. “Unpacking Security Policy Compliance: The Motivators and Barriers of Employees’ Security Behaviors”. In: *11th Symposium On Usable Privacy and Security.* USENIX Association, 2015.

[25] Boehm, B. and Basili, V. R. “Software defect reduction top 10 list”. In: *Foundations of empirical software engineering: the legacy of Victor R. Basili* 426.37 (2005).

[26] Boehm, B., Bose, P., Horowitz, E., and Lee, M.-J. “Software requirements as negotiated win conditions”. In: *IEEE International Conference on Requirements Engineering.* IEEE. 1994.

[27] Böhmer, M., Hecht, B., Schöning, J., Krüger, A., and Bauer, G. “Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage”. In: *13th International Conference on Human Computer Interaction With Mobile Devices And Services.* ACM. 2011.

[28] Breu, S., Premraj, R., Sillito, J., and Zimmermann, T. “Information needs in bug reports: improving cooperation between developers and users”. In: *ACM Conference on Computer Supported Cooperative Work.* ACM. 2010.

[29] Brill, O. and Knauss, E. “Structured and unobtrusive observation of anonymous users and their context for requirements elicitation”. In: *19th IEEE International Requirements Engineering Conference.* IEEE. 2011.

[30] Brownlee", J. *How to Normalize and Standardize Your Machine Learning Data in Weka.* Accessed January 2, 2020. URL: `https://tinyurl.com/vcrm8gq`.

[31] Bruegge, B. and Dutoit, A. H. "Object–Oriented Software Engineering. Using UML, Patterns, and Java". In: *Learning* 5.6 ().

[32] Burkov, A. *The hundred-page machine learning book.* Quebec City, Canada: Andriy Burkov, 2019.

[33] Burman, P. "A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods". In: *Biometrika* 76.3 (1989).

[34] Buse, R. P. and Zimmermann, T. "Information needs for software development analytics". In: *34th international conference on software engineering.* IEEE Press. 2012.

[35] "BYOD: Security and Privacy Considerations". In: *IT Professional* 14.5 (2012).

[36] Carlshamre, P. "Release Planning in Market-Driven Software Product Development: Provoking an Understanding". In: *Requirements Engineering* 7.3 (2002).

[37] Chang, I., Tai, H., Hsieh, D., Yeh, F., and Chang, S. "Design and Implementation of the Travelling Time- and Energy-Efficient Android GPS Navigation App with the VANET-Based A* Route Planning Algorithm". In: *International Symposium on Biometrics and Security Technologies*. 2013.

[38] Chen, H., Chiang, R. H., and Storey, V. C. "Business intelligence and analytics: From big data to big impact." In: *MIS Quarterly* 36.4 (2012).

[39] Chen, N., Lin, J., Hoi, S. C. H., Xiao, X., and Zhang, B. "AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace". In: *36th International Conference on Software Engineering.* New York, NY, USA: ACM, 2014.

[40] Choi, J. "Context-driven requirements analysis". In: *International Conference on Computational Science and Its Applications*. Springer. 2007.

[41] Chollet, F. et al. *Keras.* `https://keras.io`. 2015.

[42] Chowdhury, G. G. *Introduction to modern information retrieval.* Facet publishing, 2010.

[43] Chung, L. et al. “Capturing and reusing functional and non-functional requirements knowledge: A goal-object pattern approach”. In: *IEEE International Conference on Information Reuse & Integration*. IEEE. 2006.

[44] Coble, J. M., Karat, J., and Kahn, M. G. “Maintaining a focus on user requirements throughout the development of clinical workstation software”. In: *ACM SIGCHI Conference on Human Factors in Computing Systems*. ACM. 1997.

[45] Cohen, D., Feather, M. S., Narayanaswamy, K., and Fickas, S. S. “Automatic monitoring of software requirements”. In: *International Conference on Software Engineering*. Vol. 97. Citeseer. 1997.

[46] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. “Natural language processing (almost) from scratch”. In: *Journal of Machine Learning Research* 12.Aug (2011).

[47] Coughlan, J. and Macredie, R. D. “Effective communication in requirements elicitation: a comparison of methodologies”. In: *Requirements Engineering* 7.2 (2002).

[48] Dąbrowski, J., Letier, E., Perini, A., and Susi, A. “Finding and analyzing app reviews related to specific features: A research preview”. In: *25th International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2019.

[49] Davenport, T. H., Harris, J. G., and Morison, R. *Analytics at work: Smarter decisions, better results*. Harvard Business Press, 2010.

[50] Davenport, T. H. et al. “Competing on analytics”. In: *Harvard Business Review* 84.1 (2006).

[51] Davis, J. and Goadrich, M. “The Relationship Between Precision-Recall and ROC Curves”. In: *23rd International Conference on Machine Learning*. ACM, 2006.

[52] Dedić, N. and Stanier, C. “Measuring the success of changes to existing business intelligence solutions to improve business intelligence reporting”. In: *International Conference on Research and Practical Issues of Enterprise Information Systems*. Springer. 2016.

[53] Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., and Kumar, R. "Rico: A Mobile App Dataset for Building Data-Driven Design Applications". In: *30th Annual Symposium on User Interface Software and Technology.* 2017.

[54] Deka, B., Huang, Z., Franzen, C., Nichols, J., LI, Y., and Kumar, R. "ZIPT: Zero-Integration Performance Testing of Mobile App Designs". In: *30th Annual Symposium on User Interface Software and Technology.* 2017.

[55] Deka, B., Huang, Z., and Kumar, R. "ERICA: Interaction mining mobile apps". In: *29th Annual Symposium on User Interface Software and Technology.* ACM. 2016.

[56] "Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences". In: *Conference on Human Factors in Computing Systems.* The Hague, The Netherlands: ACM, 2000.

[57] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[58] Di Sorbo, A., Panichella, S., Alexandru, C. V., Shimagaki, J., Visaggio, C. A., Canfora, G., and Gall, H. C. "What would users change in my app? summarizing app reviews for recommending software changes". In: *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM. 2016.

[59] Di Sorbo, A., Panichella, S., Alexandru, C. V., Visaggio, C. A., and Canfora, G. "SURF: summarizer of user reviews feedback". In: *39th IEEE/ACM International Conference on Software Engineering Companion.* IEEE. 2017.

[60] Dick, J., Hull, E., and Jackson, K. *Requirements engineering.* Springer, 2017.

[61] Do, T. M. T., Blom, J., and Gatica-Perez, D. "Smartphone Usage in the Wild: A Large-scale Analysis of Applications and Context". In: *13th International Conference on Multimodal Interfaces.* ACM, 2011.

[62] Dougherty, J., Kohavi, R., Sahami, M., et al. "Supervised and unsupervised discretization of continuous features". In: *Machine learning: twelfth international conference.* Vol. 12. 1995.

[63] Earthy, J., Jones, B. S., and Bevan, N. “The improvement of human-centred processes—facing the challenge and reaping the benefit of ISO 13407”. In: *International Journal of Human-Computer Studies* 55.4 (2001).

[64] ENG and HITeC. *D2.3 Requirements Intelligence Engine version 2*. English. Version 2.3. European Commission. URL: `https://cordis.europa.eu/project/id/732463/results/`. to appear.

[65] Eslahi, M., Naseri, M., Hashim, H., Tahir, N., and Saad, E. “BYOD: Current state and security challenges”. In: *IEEE Symposium on Computer Applications and Industrial Electronics*. 2014.

[66] "Facebook". *Google Play App Page*. Accessed January 2, 2020. URL: `https://play.google.com/store/apps/details?id=com.facebook.katana`.

[67] Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F., and Antoniol, G. “Keep it simple: Is deep learning good for linguistic smell detection?” In: *25th IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE. 2018.

[68] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., and Estrin, D. “Diversity in smartphone usage”. In: *8th International Conference on Mobile Systems Applications and Services*. ACM. 2010.

[69] Falkner, A., Palomares, C., Franch, X., Schenner, G., Aznar, P., and Schoerghuber, A. “Identifying Requirements in Requests for Proposal: A Research Preview”. In: *25th International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2019.

[70] Felfernig, A., Stettinger, M., Wundara, M., and Stanik, C. “Künstliche Intelligenz in der Öffentlichen Verwaltung”. In: *Handbuch E-Government*. Springer, 2019.

[71] Fickas, S. and Feather, M. S. “Requirements monitoring in dynamic environments”. In: *IEEE International Symposium on Requirements Engineering*. 1995.

[72] Figure Eight. *Website*. Accessed December 22, 2019. URL: `https://www.figure-eight.com/`.

[73] Finkelstein, A., Harman, M., Jia, Y., Martin, W., Sarro, F., and Zhang, Y. “App store analysis: Mining app stores for relationships between customer, business and technical characteristics”. In: *Research Note* 14.10 (2014).

[74] Finkelstein, A., Harman, M., Jia, Y., Martin, W., Sarro, F., and Zhang, Y. “Investigating the relationship between price, rating, and popularity in the Blackberry World App Store”. In: *Information and Software Technology* 87 (2017).

[75] Fotrousi, F., Fricker, S. A., and Fiedler, M. “Quality requirements elicitation based on inquiry of quality-impact relationships”. In: *22nd IEEE International Requirements Engineering Conference*. IEEE. 2014.

[76] Fu, W. and Menzies, T. “Easy over hard: A case study on deep learning”. In: *11th Joint Meeting on Foundations of Software Engineering*. ACM. 2017.

[77] Fucci, D., Stanik, C., Montgomery, L., Kurtanovic, Z., Johann, T., and Maalej, W. “Research on NLP for RE at the University of Hamburg: A Report.” In: *International Working Conference on Requirements Engineering: Foundation for Software Quality Workshops*. 2018.

[78] Galvis Carreño, L. V. and Winbladh, K. “Analysis of user comments: an approach for software requirements evolution”. In: *35th International Conference on Software Engineering*. IEEE Press, 2013.

[79] Galvis Carreño, L. V. and Winbladh, K. “Analysis of User Comments: An Approach for Software Requirements Evolution”. In: *35th International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013.

[80] Gao, C., Wang, B., He, P., Zhu, J., Zhou, Y., and Lyu, M. R. “PAID: Prioritizing app issues for developers by tracking user reviews over versions”. In: *26th IEEE International Symposium on Software Reliability Engineering*. 2015.

[81] Gao, C., Xu, H., Hu, J., and Zhou, Y. “AR-Tracker: Track the Dynamics of Mobile Apps via User Review Mining”. In: *IEEE Symposium on Service-Oriented System Engineering*. 2015.

[82] Gao, C., Zheng, W., Deng, Y., Lo, D., Zeng, J., Lyu, M. R., and King, I. "Emerging App Issue Identification from User Feedback: Experience on WeChat". In: *41st International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '10. Piscataway, NJ, USA: IEEE Press, 2019.

[83] Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[84] Glinz, M. "On Non-Functional Requirements". In: *15th IEEE International Requirements Engineering Conference*. 2007.

[85] Glinz, M. and Wieringa, R. J. "Guest editors' introduction: Stakeholders in requirements engineering". In: *IEEE software* 24.2 (2007).

[86] Gómez, M., Rouvoy, R., Adams, B., and Seinturier, L. "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring". In: *IEEE/ACM International Conference on Mobile Software Engineering and Systems*. IEEE. 2016.

[87] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[88] Google. *Google Play Store*. Accessed August 11, 2019. URL: `https://play.google.com/store/apps`.

[89] Google. *Firebase Crashlytics*. Accessed December 22, 2019. URL: `https://firebase.google.com/products/crashlytics`.

[90] Google. *Android for work*. Accessed January 15, 2017. URL: `https://www.android.com/work/`.

[91] Google. *Google Analytics for Mobile Apps*. Accessed July 23, 2019. URL: `https://developers.google.com/analytics/solutions/mobile`.

[92] Google. *Build more accessible apps*. Accessed September 8, 2018. URL: `https://developer.android.com/guide/topics/ui/accessibility/index.html`.

[93] Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. "Checking App Behavior Against App Descriptions". In: *36th International Conference on Software Engineering*. Hyderabad, India: ACM, 2014.

[94] Gorry, G. A. and Scott Morton, M. S. “A framework for management information systems”. In: (1971).

[95] Gu, X. and Kim, S. “" What Parts of Your Apps are Loved by Users?"(T)”. In: *30th IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2015.

[96] Guo", Y. *The 7 Steps of Machine Learning*. Accessed January 2, 2020. URL: `https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e`.

[97] Guzman, E., El-Haliby, M., and Bruegge, B. “Ensemble Methods for App Review Classification: An Approach for Software Evolution (N)”. In: *30th IEEE/ACM International Conference on Automated Software Engineering*. 2015.

[98] Guzman, E., Ibrahim, M., and Glinz, M. “Prioritizing User Feedback from Twitter: A Survey Report”. In: *4th IEEE/ACM International Workshop on CrowdSourcing in Software Engineering*. 2017.

[99] Guzman, E., Alkadhi, R., and Seyff, N. “A needle in a haystack: What do twitter users say about software?” In: *24th IEEE International Requirements Engineering Conference*. IEEE. 2016.

[100] Guzman, E., Ibrahim, M., and Glinz, M. “A little bird told me: Mining tweets for requirements and software evolution”. In: *25th IEEE International Requirements Engineering Conference*. IEEE. 2017.

[101] Guzman, E. and Maalej, W. “How do users like this feature? a fine grained sentiment analysis of app reviews”. In: *22nd IEEE International Requirements Engineering Conference*. IEEE. 2014.

[102] Hall, M. A. “Correlation-based Feature Subset Selection for Machine Learning”. PhD thesis. Hamilton, New Zealand: University of Waikato, 1998.

[103] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. “The WEKA data mining software: an update”. In: *ACM SIGKDD explorations newsletter* 11.1 (2009).

[104] Häring, M., Loosen, W., and Maalej, W. “Who is addressed in this comment?: Automatically classifying meta-comments in news comments”. In: *Proceedings of the ACM on Human-Computer Interaction* 2.CSCW (2018).

[105] Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., and Mobasher, B. “Supporting domain analysis through mining and recommending features from online product listings”. In: *IEEE Transactions on Software Engineering* 39.12 (2013).

[106] Harman, M., Jia, Y., and Zhang, Y. “App store mining and analysis: MSR for app stores”. In: *9th IEEE Working Conference on Mining Software Repositories*. 2012.

[107] Hassan, S., Tantithamthavorn, C., Bezemer, C.-P., and Hassan, A. E. “Studying the dialogue between users and developers of free apps in the google play store”. In: *Empirical Software Engineering* 23.3 (2018).

[108] Hong, D. K., Nikravesh, A., Mao, Z. M., Ketkar, M., and Kishinevsky, M. “PerfProbe: A Systematic, Cross-layer Performance Diagnosis Framework for Mobile Platforms”. In: *6th International Conference on Mobile Software Engineering and Systems*. MOBILESoft ’19. Piscataway, NJ, USA: IEEE Press, 2019.

[109] Hoon, L., Vasa, R., Schneider, J.-G., and Grundy, J. *An analysis of the mobile app review landscape: trends and implications*. Tech. rep. Swinburne University of Technology, 2013.

[110] Humm, B. and Wietek, F. “Architektur von Data Warehouses und Business Intelligence Systemen”. In: *Informatik-Spektrum* 28.1 (2005).

[111] Hutchings, A. F. and Knox, S. T. “Creating products customers demand”. In: *Communications of the ACM* 38.5 (1995).

[112] Iacob, C. and Harrison, R. “Retrieving and analyzing mobile apps feature requests from online reviews”. In: *10th Working Conference on Mining Software Repositories*. IEEE Press, 2013.

[113] "IDC". *Smartphone OS Market Share, 2017 Q1*. Accessed Feburary 15, 2017. URL: `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

[114] “IEEE Standard Glossary of Software Engineering Terminology”. In: *IEEE Std 610.12-1990* (1990).

[115] Iqbal, T., Seyff, N., and Mendez, D. “Generating Requirements Out of Thin Air: Towards Automated Feature Identification for New Apps”. In: (2019).

[116] Jansen, B. J. and Spink, A. “An analysis of web searching by European AlltheWeb. com users”. In: *Information Processing & Management* 41.2 (2005).

[117] Jesdabodi, C. and Maalej, W. “Understanding Usage States on Mobile Devices”. In: *ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015.

[118] Johann, T., Stanik, C., Alizadeh B., A. M., and Maalej, W. “Safe: A simple approach for feature extraction from app descriptions and app reviews”. In: *25th IEEE International Requirements Engineering Conference*. IEEE. 2017.

[119] Johanssen, J. O., Kleebaum, A., Bruegge, B., and Paech, B. “How do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering?” In: *27th IEEE International Requirements Engineering Conference*. 2019.

[120] Johnson, R. B. “Examining the validity structure of qualitative research”. In: *Education* 118.2 (1997).

[121] Jokela, T., Ojala, J., and Olsson, T. In: *33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015.

[122] Jones, S. L., Ferreira, D., Hosio, S., Goncalves, J., and Kostakos, V. “Revisitation analysis of smartphone app use”. In: *ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2015.

[123] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. “FastText.zip: Compressing text classification models”. In: *arXiv preprint arXiv:1612.03651* (2016).

[124] Kanchev, G. M., Murukannaiah, P. K., Chopra, A. K., and Sawyer, P. “Canary: Extracting Requirements-Related Information from Online Discussions”. In: *25th IEEE International Requirements Engineering Conference*. 2017.

[125] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. *Feature-oriented domain analysis (FODA) feasibility study.* Tech. rep. Carnegie-Mellon University Pittsburgh. Software Engineering Institute, 1990.

[126] Keertipati, S., Savarimuthu, B. T. R., and Licorish, S. A. "Approaches for prioritizing feature improvements extracted from app reviews". In: *20th international conference on evaluation and assessment in software engineering.* ACM. 2016.

[127] Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., and Ziegler, H. "Visual analytics: Scope and challenges". In: *Visual Data Mining.* Springer, 2008.

[128] Kim, Y. "Convolutional Neural Networks for Sentence Classification". In: (2014).

[129] Kodeswaran, P., Chakraborty, D., Sharma, P., Mukherjea, S., and Joshi, A. "Combining Smart Phone and Infrastructure Sensors to Improve Security in Enterprise Settings". In: *ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication.* ACM, 2013.

[130] Kohavi, R. et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *14th International Joint Conference on Artificial Intelligence.* Vol. 14. 2. 1995.

[131] Kotonya, G. and Sommerville, I. *Requirements engineering: processes and techniques.* Wiley Publishing, 1998.

[132] Kujala, S., Kauppinen, M., Lehtola, L., and Kojo, T. "The role of user involvement in requirements quality and project success". In: *13th IEEE International Conference on Requirements Engineering.* IEEE. 2005.

[133] Kurtanović, Z. and Maalej, W. "Automatically classifying functional and non-functional requirements using supervised machine learning". In: *25th IEEE International Requirements Engineering Conference.* IEEE. 2017.

[134] Kurtanović, Z. and Maalej, W. "Mining user rationale from software reviews". In: *25th IEEE International Requirements Engineering Conference.* IEEE. 2017.

[135] Larman, C. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2010.

[136] Laudon, J. P. and Laudon, K. C. *Management Information Systems: Managing the Digital Firm*. Pearson Education, 2017.

[137] Lee, B.-H., Kim, H.-N., Jung, J.-G., and Jo, G.-S. "Location-Based Service with Context Data for a Restaurant Recommendation". In: *Database and Expert Systems Applications*. Ed. by S. Bressan, J. Küng, and R. Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[138] Legg, S., Hutter, M., et al. "A collection of definitions of intelligence". In: *Frontiers in Artificial Intelligence and applications* 157 (2007).

[139] Li, H., Zhang, L., Zhang, L., and Shen, J. "A user satisfaction analysis approach for software evolution". In: *IEEE International Conference on Progress in Informatics and Computing*. Vol. 2. 2010.

[140] Li, H., Zhang, L., Zhang, L., and Shen, J. "A user satisfaction analysis approach for software evolution". In: *IEEE International Conference on Progress in Informatics and Computing*. Vol. 2. IEEE. 2010.

[141] Li, Y., McLean, D., Bandar, Z. A., O'Shea, J. D., and Crockett, K. "Sentence Similarity Based on Semantic Nets and Corpus Statistics". In: *IEEE Transactions on Knowledge and Data Engineering* 18.8 (2006).

[142] Li, Y., Lu, J., Zhang, L., and Zhao, Y. "Taxi booking mobile app order demand prediction based on short-term traffic forecasting". In: *Transportation Research Record* 2634.1 (2017).

[143] Liang, T.-P., Li, X., Yang, C.-T., and Wang, M. "What in Consumer Reviews Affects the Sales of Mobile Apps: A Multifacet Sentiment Analysis Approach". In: *International Journal of Electronic Commerce* 20.2 (2015).

[144] Licorish, S. A., Tahir, A., Bosu, M. F., and MacDonell, S. G. "On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios". In: *24th Australasian Software Engineering Conference*. 2015.

[145] Lin, J., Liu, B., Sadeh, N., and Hong, J. I. “Modeling Users’ Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings”. In: *10th Symposium On Usable Privacy and Security*. USENIX Association, 2014.

[146] Liu, T. F., Craft, M., Situ, J., Yumer, E., Mech, R., and Kumar, R. “Learning Design Semantics for Mobile Apps”. In: *31st Annual ACM Symposium on User Interface Software and Technology*. ACM, 2018.

[147] Lopez, M. M. and Kalita, J. “Deep Learning applied to NLP”. In: *arXiv:1703.03091 [cs]* (2017). arXiv: 1703.03091.

[148] Maalej, W. and Pagano, D. “On the Socialness of Software”. In: *9th IEEE International Conference on Dependable, Autonomic and Secure Computing*. 2011.

[149] Maalej, W. “Intention-Based Integration of Software Engineering Tools”. PhD thesis. Technische Universität München, 2010.

[150] Maalej, W., Fritz, T., and Robbes, R. “Collecting and processing interaction data for recommendation systems”. In: *Recommendation Systems in Software Engineering*. Springer, 2014.

[151] Maalej, W., Kurtanović, Z., Nabil, H., and Stanik, C. “On the automatic classification of app reviews”. In: *Requirements Engineering* 21.3 (2016).

[152] Maalej, W. and Nabil, H. “Bug report, feature request, or simply praise? on automatically classifying app reviews”. In: *23rd IEEE International Requirements Engineering conference*. IEEE. 2015.

[153] Maalej, W., Nayebi, M., Johann, T., and Ruhe, G. “Toward Data-Driven Requirements Engineering”. In: *IEEE Software: Special Issue on the Future of Software Engineering* 33.1 (2016). published.

[154] Maalej, W. and Thurimella, A. K. *Managing requirements knowledge*. Springer, 2013.

[155] Manning, C. D. and Schütze, H. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[156] Martens, D. and Johann, T. “On the Emotion of Users in App Reviews”. In: *2nd IEEE/ACM International Workshop on Emotion Awareness in Software Engineering*. 2017.

[157] Martens, D. and Maalej, W. “Extracting and Analyzing Context Information in User-Support Conversations on Twitter”. In: (2019).

[158] Martens, D. and Maalej, W. “Release Early, Release Often, and Watch Your Users’ Emotions: Lessons From Emotional Patterns”. In: *IEEE Software* 36.5 (2019).

[159] Martin, L. and Pu, P. “Prediction of helpful reviews using emotions extraction”. In: *28th AAAI conference on artificial intelligence*. 2014.

[160] Martin, S., Aurum, A., Jeffery, R., and Paech, B. “Requirements engineering process models in practice”. In: *7th Australian Workshop on Requirements Engineering*. 2002.

[161] Martin, W., Harman, M., Jia, Y., Sarro, F., and Zhang, Y. “The app sampling problem for app store mining”. In: *12th Working Conference on Mining Software Repositories*. IEEE Press. 2015.

[162] Martin, W., Sarro, F., Jia, Y., Zhang, Y., and Harman, M. “A survey of app store analysis for software engineering”. In: *IEEE Transactions on Software Engineering* (2016).

[163] Martínez-Fernández, S., Vollmer, A. M., Jedlitschka, A., Franch, X., López, L., Ram, P., Rodríguez, P., Aaramaa, S., Bagnato, A., Choraś, M., et al. “Continuously assessing and improving software quality with software analytics tools: a case study”. In: *IEEE access* 7 (2019).

[164] McIlroy, S., Ali, N., and Hassan, A. E. “Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store”. In: *Empirical Software Engineering* 21.3 (2016).

[165] Menzies, T. and Zimmermann, T. “Software Analytics: So What?” In: *IEEE Software* 30.4 (2013).

[166] Mikolov, T., Chen, K., Corrado, G., and Dean, J. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[167] Minelli, R. and Lanza, M. “Software Analytics for Mobile Applications–Insights Lessons Learned”. In: *17th European Conference on Software Maintenance and Reengineering*. 2013.

[168] Mokbel, M. F. and Levandoski, J. J. “Toward Context and Preference-aware Location-based Services”. In: *8th ACM International Workshop on Data Engineering for Wireless and Mobile Access*. ACM, 2009.

[169] Moran, K., Li, B., Bernal-Cárdenas, C., Jelf, D., and Poshyvanyk, D. “Automated Reporting of GUI Design Violations for Mobile Apps”. In: *40th International Conference on Software Engineering*. ACM, 2018.

[170] Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., and Poshyvanyk, D. “Auto-completing bug reports for android applications”. In: *10th Joint Meeting on Foundations of Software Engineering*. ACM. 2015.

[171] Murphy-Hill, E., Zimmermann, T., and Nagappan, N. “Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?” In: *36th International Conference on Software Engineering*. ACM. 2014.

[172] "MUSES". *GitHub Repository for the Android Data Collection App*. Accessed September 13, 2017. URL: `https://github.com/MusesProject/MusesClient`.

[173] "MUSES". *Project GitHub Site*. Accessed September 13, 2017. URL: `https://github.com/MusesProject`.

[174] "MUSES". *Project Website*. Accessed September 13, 2017. URL: `https://www.musesproject.eu/`.

[175] Nayebi, M., Farahi, H., and Ruhe, G. “Which Version Should Be Released to App Store?” In: *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2017.

[176] Nayebi, M., Cho, H., and Ruhe, G. “App store mining is not enough for app improvement”. In: *Empirical Software Engineering* 23 (2018).

[177] Nayebi, M., Marbouti, M., Quapp, R., Maurer, F., and Ruhe, G. “Crowdsourced Exploration of Mobile App Features: A Case Study of the Fort Mcmurray Wildfire”. In: *39th International Conference on Software Engineering: Software Engineering in Society Track*. IEEE Press, 2017.

[178] Nayebi, M. and Ruhe, G. *Trade-off service portfolio planning–a case study on mining the Android app market*. Tech. rep. PeerJ PrePrints, 2015.

[179] Nuseibeh, B. and Easterbrook, S. “Requirements engineering: a roadmap”. In: *Conference on the Future of Software Engineering*. ACM. 2000.

[180] Oh, J., Kim, D., Lee, U., Lee, J.-G., and Song, J. “Facilitating Developer-user Interactions with Mobile App Review Digests”. In: *Extended Abstracts on Human Factors in Computing Systems*. ACM, 2013.

[181] OpenReq. *API Documentation*. Accessed December 22, 2019. URL: `https://api.openreq.eu/`.

[182] OpenReq. *Homepage*. Accessed December 22, 2019. URL: `https://openreq.eu`.

[183] OpenReq. *Project GitHub Site*. Accessed December 22, 2019. URL: `https://github.com/openreqeu`.

[184] Oriol, M., Stade, M., Fotrousi, F., Nadal, S., Varga, J., Seyff, N., Abello, A., Franch, X., Marco, J., and Schmidt, O. “FAME: supporting continuous requirements elicitation by combining user feedback and monitoring”. In: *26th IEEE International Requirements Engineering Conference*. IEEE. 2018.

[185] Pagano, D. “PORTNEUF-A Framework for Continuous User Involvement”. PhD thesis. Technische Universität München, 2013.

[186] Pagano, D. and Brügge, B. “User involvement in software evolution practice: a case study”. In: *35th International Conference on Software Engineering*. IEEE Press. 2013.

[187] Pagano, D. and Maalej, W. “User feedback in the appstore: An empirical study”. In: *21st IEEE International Requirements Engineering Conference*. IEEE. 2013.

[188] Pahikkala, T., Boberg, J., and Salakoski, T. “Fast n-fold cross-validation for regularized least-squares”. In: *9th Scandinavian conference on artificial intelligence (SCAI 2006)*. Vol. 83. Otamedia Oy Espoo, Finland. 2006.

[189] Pak, A. and Paroubek, P. “Twitter as a corpus for sentiment analysis and opinion mining.” In: *7th International Conference on Language Resources and Evaluation*. Vol. 10. 2010. 2010.

[190] Palomba, F., Linares-Vasquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. "User reviews matter! tracking crowd-sourced reviews to support evolution of successful apps". In: *31st IEEE international conference on software maintenance and evolution*. IEEE. 2015.

[191] Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., and Gall, H. C. "How can i improve my app? classifying user reviews for software maintenance and evolution". In: *31st IEEE international conference on software maintenance and evolution*. IEEE. 2015.

[192] Pathak, A., Hu, Y. C., and Zhang, M. "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof". In: *7th ACM European Conference On Computer Systems*. ACM. 2012.

[193] Pearson, K. "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50.302 (1900).

[194] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011).

[195] Penn Arts & Sciences, Department of Linguistics. *POS tags of the Penn Treebank Project*. Accessed January 5, 2017. URL: `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`.

[196] Peterson, M. *An Introduction to Decision Theory*. Cambridge Introductions to Philosophy. Cambridge University Press, 2009.

[197] Pogarcic, I., Gligora Markovic, M., and Davidovic, V. "BYOD: A challenge for the future digital generation". In: *36th International Convention on Information Communication Technology Electronics Microelectronics*. 2013.

[198] Pohl, K. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[199] Potts, C., Takahashi, K., and Anton, A. I. *Inquiry-based scenario analysis of system requirements.* Tech. rep. Georgia Institute of Technology, 1994.

[200] Qian, F., Wang, Z., Gerber, A., Mao, Z., Sen, S., and Spatscheck, O. "Profiling resource usage for mobile applications: a cross-layer approach". In: *9th International Conference On Mobile Systems Applications, and Services.* ACM. 2011.

[201] Queirós, A., Faria, D., and Almeida, F. "Strengths and limitations of qualitative and quantitative research methods". In: *European Journal of Education Studies* (2017).

[202] Ramesh, B., Cao, L., and Baskerville, R. "Agile requirements engineering practices and challenges: an empirical study". In: *Information Systems Journal* 20.5 (2010).

[203] Raschka, S. *Python machine learning.* Packt Publishing Ltd, 2015.

[204] Replication package. *Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning.* Accessed December 22, 2019. URL: `https://mast.informatik.uni-hamburg.de/replication-packages/`.

[205] Replication package. *Safe: A simple approach for feature extraction from app descriptions and app reviews.* Accessed September 12, 2018. URL: `https://mast.informatik.uni-hamburg.de/app-review-analysis`.

[206] Robertson, S. and Robertson, J. *Mastering the Requirements Process.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.

[207] Robinson, W. N. "Monitoring software requirements using instrumented code". In: *35th Annual Hawaii International Conference on System Sciences.* 2002.

[208] Rodríguez, P., Bautista, M. A., Gonzalez, J., and Escalera, S. "Beyond one-hot encoding: Lower dimensional target embedding". In: *Image and Vision Computing* 75 (2018).

[209] Roehm, T., Gurbanova, N., Bruegge, B., Joubert, C., and Maalej, W. "Monitoring user interactions for supporting failure reproduction". In: *21st International Conference on Program Comprehension.* IEEE. 2013.

[210] Roehm, T., Nosovic, S., and Bruegge, B. “Automated extraction of failure reproduction steps from user interaction traces”. In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE. 2015.

[211] Rubinov, K., Rosculete, L., Mitra, T., and Roychoudhury, A. “Automated Partitioning of Android Applications for Trusted Execution Environments”. In: *38th IEEE/ACM International Conference on Software Engineering*. 2016.

[212] Ruhe, G., Eberlein, A., and Pfahl, D. “Trade-off analysis for requirements selection”. In: *International Journal of Software Engineering and Knowledge Engineering* 13.04 (2003).

[213] Runeson, P. and Höst, M. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical Software Engineering* 14.2 (2009).

[214] Saif, H., He, Y., and Alani, H. “Semantic sentiment analysis of twitter”. In: *International Semantic Web Conference*. Springer. 2012.

[215] Saliu, O. and Ruhe, G. “Supporting Software Release Planning Decisions for Evolving Systems”. In: *29th Annual IEEE/NASA Software Engineering Workshop*. 2005.

[216] Salz, P. A. “Monitoring mobile app performance”. In: *Journal of Direct, Data and Digital Marketing Practice* 15.3 (2014).

[217] Sarro, F., Al-Subaihin, A. A., Harman, M., Jia, Y., Martin, W., and Zhang, Y. “Feature lifecycles as they spread, migrate, remain, and die in app stores”. In: *23rd IEEE International Requirements Engineering Conference*. IEEE. 2015.

[218] Sarro, F., Harman, M., Jia, Y., and Zhang, Y. “Customer rating reactions can be predicted purely using app features”. In: *26th IEEE International Requirements Engineering Conference*. IEEE. 2018.

[219] Sawyer, P., Sommerville, I., and Viller, S. “Improving the Requirements Process.” In: *4th International Workshop on Requirements Engineering: Foundation for Software Quality*. Citeseer. 1998.

[220] Shah", T. *About Train, Validation and Test Sets in Machine Learning.* Accessed January 2, 2020. URL: `https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7`.

[221] Sharma, S. and Pandey, S. K. "Requirements elicitation: Issues and challenges". In: *2014 International Conference on Computing for Sustainable Global Development*. IEEE. 2014.

[222] Shekhar", A. *What is Feature Engineering for Machine Learning? Feature Engineering is an art.* Accessed January 2, 2020. URL: `https://medium.com/mindorks/what-is-feature-engineering-for-machine-learning-d8ba3158d97a`.

[223] Sheth, S., Kaiser, G., and Maalej, W. "Us and Them: A Study of Privacy Requirements Across North America, Asia, and Europe". In: *36th International Conference on Software Engineering.* ACM, 2014.

[224] Shin, C., Hong, J.-H., and Dey, A. K. "Understanding and prediction of mobile application usage for smart phones". In: *ACM Conference on Ubiquitous Computing.* ACM. 2012.

[225] Shollo, A. and Galliers, R. D. "Towards an understanding of the role of business intelligence systems in organisational knowing". In: *Information Systems Journal* 26.4 (2016).

[226] Silverstein, C., Marais, H., Henzinger, M., and Moricz, M. "Analysis of a very large web search engine query log". In: *ACM SIGIR Forum.* Vol. 33. 1. ACM. 1999.

[227] Slavin, R., Wang, X., Hosseini, M. B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T. D., and Niu, J. "Toward a Framework for Detecting Privacy Policy Violations in Android Application Code". In: *38th IEEE/ACM International Conference on Software Engineering*. 2016.

[228] Snapchat. *Twitter Profile of Snpachat Support (snapchatsupport.* Accessed October 9, 2019. URL: `https://twitter.com/snapchatsupport`.

[229] Sommerville, I. *Software Engineering (7th Edition).* Pearson Addison Wesley, 2004.

[230] Song, W. and Cai, J. "End-to-end deep neural network for automatic speech recognition". In: *Standford CS224D Reports* (2015).

[231] Spacy. *Website.* Accessed December 22, 2019. URL: `https://spacy.io/`.

[232] Sparck Jones, K. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 28.1 (1972).

[233] Spink, A., Ozmutlu, S., Ozmutlu, H. C., and Jansen, B. J. “US versus European Web searching trends”. In: *ACM Sigir Forum.* Vol. 36. 2. ACM. 2002.

[234] Stade, M., Fotrousi, F., Seyff, N., and Albrecht, O. “Feedback gathering from an industrial point of view”. In: *25th IEEE International Requirements Engineering Conference.* IEEE. 2017.

[235] Stanik, C., Haering, M., and Maalej, W. “Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning”. In: *27th IEEE International Requirements Engineering Conference Workshops.* IEEE. 2019.

[236] Stanik, C. and Maalej, W. “Requirements Intelligence with OpenReq Analytics”. In: *27th IEEE International Requirements Engineering Conference.* IEEE. 2019.

[237] Stanik, C., Montgomery, L., Martens, D., Fucci, D., and Maalej, W. “A Simple NLP-based Approach to Support Onboarding and Retention in Open Source Communities”. In: *34th IEEE International Conference on Software Maintenance and Evolution.* IEEE. 2018.

[238] Statista. *Distribution of free and paid apps in the Apple App Store and Google Play as of December 2019.* Accessed July 10, 2019. URL: `https://www.statista.com/statistics/263797`.

[239] Statistia. *Number of apps available in leading app stores as of 3rd quarter 2019.* Accessed July 10, 2019. URL: `https://www.statista.com/statistics/276623`.

[240] Stumpf, S., Bao, X., Dragunov, A., Dietterich, T. G., Herlocker, J., Li, L., and Shen, J. “Predicting user tasks: i know what you’re doing”. In: *20th National Conference on Artificial Intelligence, Workshop on Human Comprehensible Machine Learning.* 2005.

[241] Sun, C., Zheng, J., Yao, H., Wang, Y., and Hsu, D. F. “AppRush: Using Dynamic Shortcuts to Facilitate Application Launching on Mobile Devices”. In: *Procedia Computer Science* 19 (2013). 4th International Conference on Ambient Systems, Networks and Technologies, 3rd International Conference on Sustainable Energy Information Technology.

[242] Sundara Rajan, V. S., Malini, A., and Sundarakantham, K. “Performance evaluation of online mobile application using Test My App”. In: *IEEE International Conference on Advanced Communications, Control and Computing Technologies.* 2014.

[243] Sype, Y. S. V. D. and Maalej, W. “On lawful disclosure of personal user data: What should app developers do?” In: *7th IEEE International Workshop on Requirements Engineering and Law.* 2014.

[244] Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. “Sentiment strength detection in short informal text”. In: *Journal of the Association for Information Science and Technology* 61.12 (2010).

[245] Thematic. *Website.* Accessed December 22, 2019. URL: `https://getthematic.com/`.

[246] Twitter. *Pricing Website.* Accessed December 22, 2019. URL: `https://developer.twitter.com/en/pricing`.

[247] Twitter. *About page.* Accessed October 9, 2019. URL: `https://about.twitter.com/en_gb.html`.

[248] Valverde-Albacete, F. J. and Peláez-Moreno, C. “100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox”. In: *PloS one* 9.1 (2014).

[249] Villarroel, L., Bavota, G., Russo, B., Oliveto, R., and Di Penta, M. “Release planning of mobile apps based on user reviews”. In: *38th IEEE/ACM International Conference on Software Engineering.* IEEE. 2016.

[250] Vogelsang, A. and Borg, M. “Requirements Engineering for Machine Learning: Perspectives from Data Scientists”. In: *27th IEEE International Requirements Engineering Conference Workshops.* IEEE. 2019.

[251] Wagner, D. T., Rice, A., and Beresford, A. R. “Device Analyzer: Large-scale Mobile Data Collection”. In: *ACM SIGMETRICS Performance Evaluation Review* 41.4 (2014).

[252] Wang, Y., Wei, J., and Vangury, K. “Bring your own device security issues and challenges”. In: *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*. 2014.

[253] Wellsandt, S., Hribernik, K. A., and Thoben, K.-D. “Qualitative comparison of requirements elicitation techniques that are used to collect feedback information about product use”. In: *Procedia CIRP* 21 (2014).

[254] "Whatsapp". *Google Play App Page*. Accessed January 2, 2020. URL: `https://play.google.com/store/apps/details?id=com.whatsapp`.

[255] Wiegers, K. and Beatty, J. *Software Requirements (Developer Best Practices)*. 3rd Edition. Microsoft Press, 2014.

[256] Williams, G. and Mahmoud, A. “Modeling User Concerns in the App Store: A Case Study on the Rise and Fall of Yik Yak”. In: *26th IEEE International Requirements Engineering Conference*. 2018.

[257] Williams, G. and Mahmoud, A. “Mining Twitter feeds for software user requirements”. In: *25th IEEE International Requirements Engineering Conference*. IEEE. 2017.

[258] Wunderlist. *Google Play App page*. Accessed Feburary 15, 2017. URL: `https://play.google.com/store/apps/details?id=com.wunderkinder.wunderlistandroid`.

[259] Yan, T., Chu, D., Ganesan, D., Kansal, A., and Liu, J. “Fast App Launching for Mobile Devices Using Predictive User Context”. In: *10th International Conference on Mobile Systems, Applications, and Services*. ACM, 2012.

[260] Yang, T., Vlas, R., Yang, A., and Vlas, C. “Risk Management in the Era of BYOD: The Quintet of Technology Adoption, Controls, Liabilities, User Perception, and User Behavior”. In: *International Conference on Social Computing*. 2013.

[261] Young, R. R. *The requirements engineering handbook*. Artech House, 2004.

[262] Young, T., Hazarika, D., Poria, S., and Cambria, E. "Recent trends in deep learning based natural language processing". In: *ieee Computational intelligenCe magazine* 13.3 (2018).

[263] Yousuf, M. and Asger, M. "Comparison of various requirements elicitation techniques". In: *International Journal of Computer Applications* 116.4 (2015).

[264] Zhang, D., Han, S., Dang, Y., Lou, J.-G., Zhang, H., and Xie, T. "Software analytics in practice". In: *IEEE software* 30.5 (2013).

[265] Zhang, Z. "Effective requirements development-a comparison of requirements elicitation techniques". In: *Software Quality Management XV: Software Quality in the Knowledge Society* (2007).

[266] Zhao, S., Luo, Z., Jiang, Z., Wang, H., Xu, F., Li, S., Yin, J., and Pan, G. "AppUsage2Vec: Modeling Smartphone App Usage for Prediction". In: *35th IEEE International Conference on Data Engineering*. 2019.

[267] Zhao, S., Ramos, J., Tao, J., Jiang, Z., Li, S., Wu, Z., Pan, G., and Dey, A. K. "Discovering Different Kinds of Smartphone Users Through Their Application Usage Behaviors". In: *ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2016.

[268] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., and Weiss, C. "What Makes a Good Bug Report?" In: *IEEE Transactions on Software Engineering* 36.5 (2010).

[269] Zowghi, D. and Coulin, C. "Requirements elicitation: A survey of techniques, approaches, and tools". In: *Engineering and Managing Software Requirements*. Springer, 2005.

Eidesstattliche Eklärung:

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den ....

Christoph Stanik