Tampere University

Henna Lehto

FEEDBACK AND USER REVIEW ANALYZING TOOLS

On the use of app store reviews in software industry

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
April 2020

ABSTRACT

Henna Lehto: Feedback and user review analyzing tools: On the use of app store reviews in software industry.
M.Sc. Thesis
Tampere University
Master's Degree Programme in Software Development
April 2020

App stores offer a platform for users to download and browse, and for developers to launch applications. The use of app stores has risen along with the use of smartphones. App stores can be used for downloading, launching, updating, or browsing apps. Each app also offers a possibility to write a review of the app or give it a rating. Therefore, app stores offer a high number of feedback which can be informative for the developing teams of the applications. The thesis introduces six tools developed for managing and analyzing the feedback provided via app stores.

Research focuses on explaining the concept of feedback in software industry, characteristics of app stores and app store reviews, and introducing six different tools. The concepts of feedback, app store reviews, and four of the tools were based on literature, and two of the tools are introduced based on the use of free trials the providers offered.

The data proves that there is a need for app store review analyzers in software industry. Even though app store reviews can be generally associated with nonsense information or they may lack context, yet they offer a valuable repository of user feedback.

Consumption of applications is rising and therefore, the standards of quality will also do so. User involvement in software development is proven to lead to better understanding of requirements. Therefore, the use of app store reviews in software industry needs to be studied more to help integrating the use of them as a solid part a software development cycle.

Key words and terms: app store reviews, feedback, requirements engineering, user involvement.

Contents

1. Introduction

1.1. Research background

Technology has become more available for people to use, causing an increasing number of users. Nowadays it would be uncommon for an adult not to have a phone, and even toddlers learn how to use touch screen devices, such as smartphones and tablets, before they learn how to read. The larger number of users and their variety lead to larger number of products (such as mobile devices, applications, and interfaces) available. Therefore, users can be picky and select the exact product they want and need, and the one which satisfies their needs the most. Competition with different product providers is tense and the providers are doing their best to be the perfect fit for their customers – because if they are not, the customer will choose some other provider. But how to keep users satisfied? Providing an excellent product. That is done by listening to users and offering them to be a part of software evolution.

As the range of variety in technology users has risen, so has the differences between the skills of software developers and designers. Individual developers may even act as both developers and designers of their product, and they may lack formal education in software engineering and human computer interaction and build products that only satisfy their requirements and needs. [Iacob *et al.* 2013] Therefore, users should always be taken in concern, possibly already in the development process.

Pagano and Brügge [2013] conducted a study regarding user involvement in software evolution. User involvement is mostly done by collecting, reading and learning user feedback. According to Pagano and Brügge [2013], developers constantly need to assess the potential of user feedback to improve their software. Pagano and Brügge [2013] stated that software companies are interested in user feedback because of two reasons: the first one is that real-world data is needed from users, for example, which features are mostly used and which errors occur the most often. Secondly, according to companies which Pagano and Brügge [2013] interviewed, the user is the king. The latter would be explained with the high demand for excellent products, leading to the fact that users must be listened in order to retain them.

Moreover, software companies tend to reach better quality on their products if users are involved in the development process. According to Kujala *et al.* [2005] early user involvement is connected to a better understanding of requirements. The reason for this is that software is developed for users, which may not always be the development team. Therefore, there must be active communication between the development team and the users, so that the product would be built in the right way and it would be the one users needed. The earlier the communication process is started, the fewer misassumptions and mistakes would be made. In addition, Maalej *et al.* [2015] state that a major part of

requirements engineering is based on the users: they have to use the system, capture their needs and give feedback for the software to develop. However, more feedback is usually gathered after launching the product, not before, meaning that users must be taken in concern even after when the product is in use. The feedback available on launched products can supply software companies with a rich source of information that can be used to improve future releases [Galvis Carreño and Winbladh 2013].

One of the most popular devices today are smartphones. Since the use of smartphones has increased with another products, so does to use of mobile app stores. App stores are a significant part of a smartphone, and they could be even used as one of the defining elements of a smartphone. They offer a platform for applications, such as navigating, messaging and social media applications, to be launched, downloaded and updated.

Additionally, applications can be reviewed in app stores, however, according to Iacob *et al.* [2013] there has been only little interest in how online reviews could benefit developers. Submitting reviews on the app store is fast and easy for the users – an average app receives 22 reviews per day [Pagano and Maalej 2013], leading to a great amount of real-world user data available to developers. Moreover, usually, the number of reviews of one app exceeds a human's capacity to read them all to identify users' issues [Iacob and Harrison 2013]. Therefore, app store feedback provides a large source of information for both users and developers so it would be irresponsible not to take the feedback received from there into account when gathering user data. Moreover, having a positive rating in application distribution platforms and particular rating sites pushes applications into top lists, which can lead to more downloads and higher sales numbers [Pagano and Brügge 2013].

One solution for managing app store feedback is to use tools developed for especially managing and analyzing app store feedback from software industrial point of view. Since the issue of managing great amounts of app store reviews is rather new, the research for tools designed for review analyzing has also mostly begun during the last decade. Yet there are already promising tools in commercial use, and also more detailed research available for methods in building such tools. This thesis will cover four different approaches in building review analyzing tools and introduce two commercial app store review managing tools.

1.2. Research aims and approach

The thesis introduces and compares four tools which are designed for analyzing and managing app store reviews. Before introducing the tools, feedback collection in the software industry and common characteristics of app store reviews must be introduced. In addition, commercial services, which analyze reviews in app stores and provide marketing analyzes will be briefly covered.

The research is an integrative literature review. The purpose of this method is to overview the knowledge base and to review on the theoretical foundation of the topic, and the general aim of data analysis is to critically analyze and examine the literature and the main ideas and relationships of the issue [Snyder 2019]. As the thesis will overview and review the current problems and solutions of app store review managing, following the comparison of different tools, where relationships between different solutions are examined, the research will follow the concept of an integrative literature review. The information will be mostly literature, besides a few online sources. Mobile application development is a relevantly new topic, but there is a significant amount of source material available, so finding literature will not be an obstacle. Furthermore, mobile app reviews have also been studied during the last decade, providing many useful sources.

The research questions will be "What kind of tools can software companies use to manage app store feedback?" and "What are ideal features in app store review analyzing tools from software industry perspective?". Along with these questions, the research will also answer the questions "How can app store feedback be used in software evolution?" and "How software companies manage feedback?".

The process of making the thesis begun in October 2019 with browsing the literature and finding useful source material. The writing process begun two months later and was finished in April 2020. Even though most of the literature was collected before the writing process started, the information regarding the commercial app store review managing tools was collected with trials which took place in March 2020.

1.3. Thesis structure

The thesis will follow the following structure: The first section is the introduction. In the second section, feedback collection from a software industry perspective is introduced, covering different types of feedback, the style of written feedback and feedback management in software companies. The third section introduces app stores and characteristics of app store reviews. The fourth section is two-parted: the first part provides information of two commercially provided app store managing services, and the second part goes in more detail into the four tools selected for the review. The fifth section compares the tools selected for the review, covering their data collection, language processing, topic modelling, sentimental analysis tools and classifying methods. The sixth and the final section is the conclusion.

2. Feedback collection from software industrial perspective

Morales-Ramirez [2013] defines feedback as meaningful information provided by users of widely used software applications with the purpose of suggesting improvements to such applications, for example, new needs, modification or strategic behaviors.

Feedback can be given in multiple ways but the written form is the most popular and efficient. The reason for this is that spoken feedback may not always be heard by the right person or passed on to them. Additionally, spoken feedback can be easily forgotten, if there is no reminder or evidence of it, such as a written note. However, spoken feedback may come across as more genuine than for example, anonymous comments, but yet it is still harder to track and manage. Moreover, when it comes to ratings, such as star ratings, the reason why a user has given only two stars instead of five, can be left unclear without an explanation. This is why the written form is the most reliable: in the best cases, it can be full of useful and well-argued information, which can be observed by many people in different times and places.

Timely and constructive feedback from users is a significant help for developers to fix bugs, implement new features, and improve user experience [Chen *et al.* 2014] and according to Pagano and Brügge [2013] user feedback does play an important role in software companies. User feedback provides important information for developers, helps to improve software quality and to identify missing features. For example, the priority of a certain feature could change based on user feedback – the comments can tell if the users would accept the product without that feature. [Pagano and Brügge 2013] For instance, if an email app lacks a feature of adding attachments, users would most likely not accept and use the product. Therefore, the feature of adding attachment would have to be prioritized high.

However, different companies may be looking for different types of feedback. According to Pagano and Brügge [2013] a company was especially interested in only bug reports, whereas another stated that rating was the most critical. On the other hand, another company did take ratings as irrelevant, but the reason for the contrast of these two companies was the type and the number of end-users. The company, which cared for ratings, had a large consumer audience, unlike the latter one, whose audience was a small group of software professionals. [Pagano and Brügge 2013] Therefore, feedback has different meanings to them.

The company with a large consumer audience cares most about ratings mostly because ratings have a clear effect on sales, meaning that low ratings would cause less profit to the company. Additionally, since there is a large number of consumers, following ratings is a faster and easier way to keep up with the current state of users' opinions. On the other hand, the company with an audience of a small group of software professionals have fewer users to satisfy, but they may be more critical, since they are professionals. Having a correctly working program is more important to both users and the company, than having a more usable interface or more possibly unnecessary features.

2.1. The role of feedback in creating scenarios for requirements engineering

According to Sommerville and Sawyer [1997] "requirements engineering is the process of discovering, documenting and managing the requirements for a computer-based system. The goal of requirements engineering is to produce a set of system requirements which, as far as possible, is complete, consistent, relevant and reflects what the customer actually wants". A common tool for requirements engineering is creating scenarios, which describes issues such as who uses the product, how it is used, when it is used. This helps the developing team to get more insights into users' minds.

Li *et al.* [2018] state that scenarios could have various roles in software system development, from a rich and informal narrative description of real-world experience on using a system to formatted texts and more formal models of an event sequence or system behavior. Written end-user feedback usually tends to be more narrative and real-world based, because users write from their experiences, which normally come from using the product in real-world. For example, if a feature request is proposed, the team could start collecting the scenarios from the stakeholders who propose the change [Li *et al.* 2018]. For example, if there was a request for adding styling options into a mobile phone messaging application, the developing team would start finding scenarios from the users, who request it. A scenario could already be included in the feature request, such as: "I wish that there were more styling options. Like when I'm writing a long message, it would be great if I could somehow highlight some words. For example, if I could just bold the text by tapping the word and choosing the bold option". When the scenarios are collected, the developing team can be more able to specify the situational context of the feature with the reference of the situational context classification [Li *et al.* 2018]. The situational context is presented as context in the conceptual model of feedback presented later in Figure 1. The role of the user is a significant part of that, and user feedback can already offer the data needed for feature changes and scenarios, so there may not be even a need to conduct a study, for instance, to get the information.

2.2. Different types of feedback

As stated, feedback can be given in multiple ways. It can be written, spoken, rated or even in a form of a picture, such as a screenshot. In social media, likes and shares can also be considered as feedback, since they are often connected with the general user sentiment. Additionally, sometimes even facial expressions can also be counted as feedback. Since feedback can be given in various forms, it should be narrowed down so that the different types of feedback forms would be easier managed. The scope can be, for example, based on the platform the feedback was given or the type of it (spoken, written, etc.).

One option to categorize feedback is to split it into explicit and implicit feedback. Explicit feedback tells how the user is feeling about the product, and the user is

intentionally giving it. For example, emails, phone calls, comments on social media, structured feedback forms, surveys, and app store reviews are explicit feedback.

On the contrary, implicit feedback provides information that the user may not be intentionally giving. Usage data, for example, is implicit feedback. Usage data can include the number of clicks the user makes, mouse movement, performance info (such as disc memory and battery life), and feature usage sequences. Additionally, some providers offer beta versions of their products for users to test, and the usage data from beta version is collected for further development. Moreover, contexts issues such as time and location can also be classified as usage data. When and where the product is used is useful information, since it can tell for example, if the usage changes after an update, which can indicate if the update was successful or not. In addition to usage data, the data received from observing users or lead users [Thomke and Nimgade 1998] is implicit data. Explicit feedback summarizes the subjective opinions of users, where the implicit feedback can help with understanding the reason their opinion was based on [Maalej *et al.* 2015]. They support each other, since with usage data, developers would be able to understand the rationale for users' opinion [Maalej *et al.* 2015].

Together with the explicit feedback, it may help the developing team to solve a bug or user experience issue. However, implicit feedback can be helpful also alone. Sequences of feature usages, duration, and time and date can be studied to benefit developers and requirements analysts. [Maalej *et al.* 2015]

Another way to manage feedback is by dividing it into pulled and pushed feedback. Maalej *et al.* [2015] state that if software companies ask for feedback, it is pulled, but if they allow the user to trigger the feedback communication, it is pushed. Therefore, pulled feedback can, for example be given in workshops, questionnaires, interviews, or in other situations where the provider especially asks for user feedback. Moreover, usage data is also pulled feedback, because the provider is the initiative party of the communication – even though the user creates the data, the motion to collect it comes from the provider and the user may not realize that they are giving feedback. Additionally, when the provider conducts a survey, whether it is an online form or face-to-face interview, the users are asked to answer questions presented by the provider, meaning that the feedback is pulled from them.

On the other hand, when the users themselves decide to start communicating the feedback is classified as pushed feedback. Writing comments on social media, rating services, liking, and sharing are all actions triggered by the user. Additionally, writing emails, calling, writing reviews and rating in app stores are also pushed feedback. Moreover, lead users also provide pushed feedback. They are users, who start the communication and implicitly deliver useful input. Typically lead users adjust the product by themselves to their needs which gives input to software engineers how to improve the

software system. [Maalej *et al.* 2015; Thomke and Nimgade 1998] Additionally, users can reveal useful input while being observed during their work [Maalej *et al.* 2015], which is also pushed feedback since the feedback is not especially asked but rather given by the intention of the user. For example, if the cashier is observed by the provider while using the cashing machine, the provider learns information from the use of the product in a natural environment. The information can be such issues as problems as a left-handed user or longer thinking pauses before doing some specific task.

Other ways to group feedback can be qualitative (descriptive, not measured) and quantitative (information about quantities), and structured or unstructured feedback. Structured feedback follows certain forms, such as ratings, multiple choices, or detailed forms. On the other hand, unstructured feedback is information provided by the user which does not follow any form, for example, email or a review written in natural language. Quantitative feedback tends to focus on usage data, where qualitative is more based on user opinions [Olsson and Bosch 2015]. There is a close connection between qualitative and quantitative, and structured and unstructured feedback. Qualitative feedback can be given in a structured or unstructured form, such as writings and ratings, but quantitative is given in a structured form.

Figure 1 presents a conceptual model of different feedback types, classifying the feedback channels and forms into pushed or pulled, and implicit or explicit feedback. Additionally, the types are marked either as quantitative or qualitative.

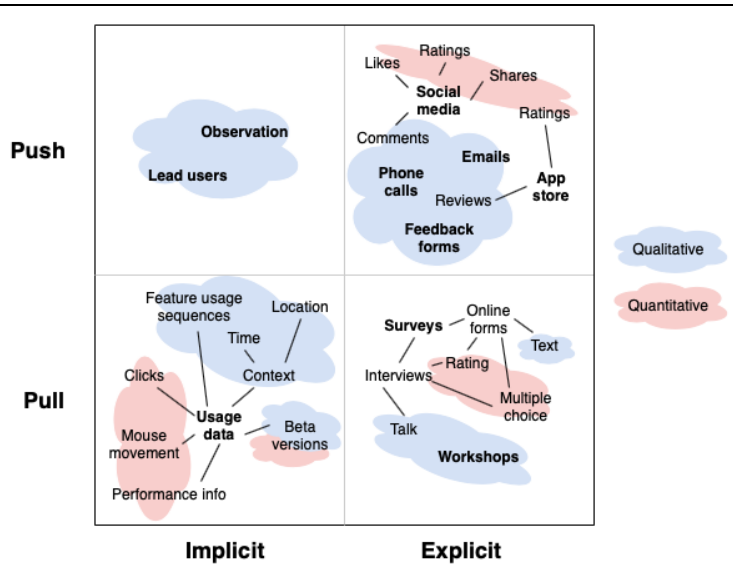

Figure 1. Conceptual model of various feedback types.

Observation and the information provided by lead users is implicit pushed feedback. As it is mainly descriptive data, it is qualitative. On the contrary, usage data is marked as pulled implicit data. Clicks, mouse movement, and performance info is data which can be measured, therefore, they are classified as quantitative data. However, feature usage

sequences and context cannot be measured, making them quantitative information. Yet they are both part of general usage data. Beta versions are attached to usage data, since the data collected from the use of them is mainly usage data. However, usage data collected from beta versions can be either quantitative or qualitative, since it can include both clicks, mouse movement, performance info, but also feature usage sequences and the context of use.

Surveys are pulled and explicit feedback. Surveys can be either on online or interviews, and have both qualitative and quantitative data. Text written in natural language or talking cannot be measured so it is qualitative information. However, multiple choice answers and ratings are quantitative. In addition to surveys, workshops arranged by the provider are pulled and explicit feedback, where the information cannot be measured, so it is quantitative.

Finally, pushed explicit feedback includes feedback given in social media, phone calls, emails, app store reviews and ratings, and feedback forms. Social media covers comments, likes, ratings, and shares, where app stores includers ratings and reviews. As social media comments, emails, phone calls, app store reviews, and feedback forms include natural language as user input, they are quantitative information. Social media and app store ratings, likes, and shares can be measured as numbers, so they are quantitative data.

2.3. Feedback channels

As there are different ways to provide feedback, there are multiple platforms to do that: more traditional ways such as calling or emailing, or a separate feedback form, or social media channels, such as Twitter and Facebook. Figure 2 presents a feedback form on Tampere University website.

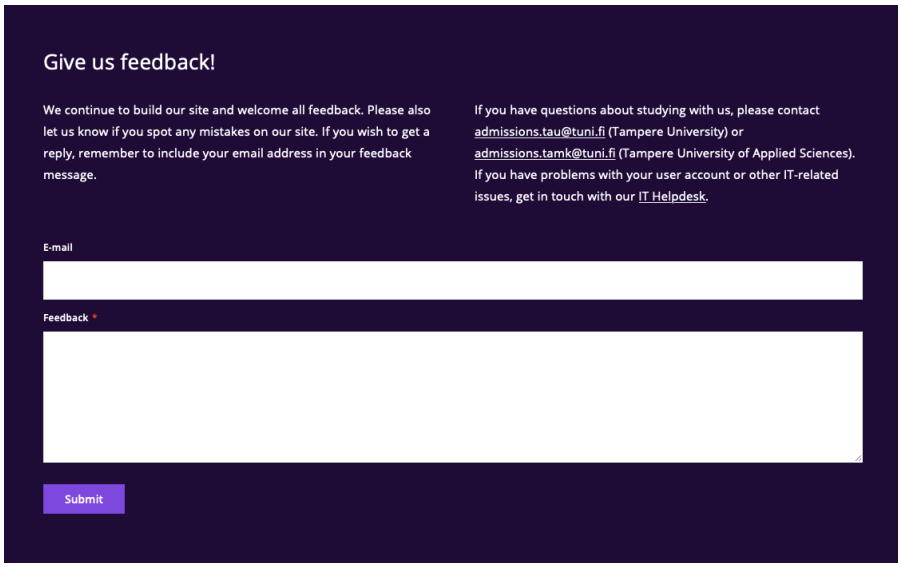

Figure 2. Screenshot of a feedback form on the Tampere University website.

The information written on feedback forms can be directed to an email and then be read by the person who is responsible from the email. Some forms may have an option of wanting an answer to the feedback, which can create trust on the user that the feedback will be read and replied by a human.

Figure 3 presents a post on the Facebook page of the messaging application WhatsApp. The post has been made by the administer of the page, but there are comments written by normal Facebook users. The significant difference between this feedback form is that other users can see what other people have already commented. This feature is common in also other social media channels. When users can see each other's comments, they may not feel the need to write a new one, if their issue has already been said. Instead, they can show that the issue presented in the other comment is relevant to them by liking the comment or/and replying the comment. This decreases the repetition and the amount of feedback to manage.

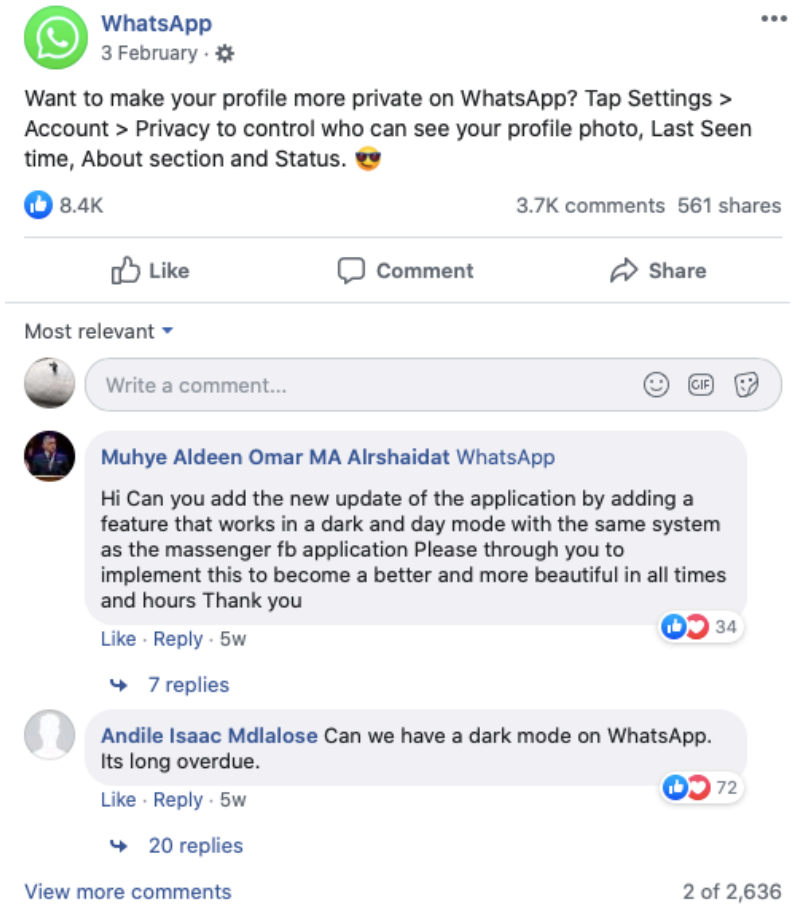

Figure 3. Screenshot of a post on the Facebook page of WhatsApp.

Additionally, the significance of an individual comment can be interpreted higher if the comment has many likes and replies supporting the original comment. Consequently, even though the feedback given on social media channels may not be as formal or even

thought as regular feedback, it should be taken in concern as least as much as the feedback provided on traditional channels. The reason is that as other users can see what everyone has commented, the company's actions towards feedback can increase or decrease the image they have on users' minds. For example, if a user writes a comment on social media asking for a dark mode and if the dark mode would be implemented, users could see the possible connection between the comment and the new feature. Especially if the comment has a high amount of likes and replies, and even if the administrator of the page would reply to the comment, telling that the feature is on its way. On the other hand, if the feature request would have been submitted via the regular feedback form, the connection between user feedback and the implementation would not be as clear. It would be useful for companies to show their users that they are listened, and this can be done easier on social media channels. On the contrary, social media can also be more crucial, if the number of unsatisfied users was high, and the company would not react to that in any way. In Figure 3, it can be seen that the comments the original post has are not related to the post. This makes comments regarding a certain issue more difficult to track and group.

As the focus of this thesis is the app store reviews, app stores generally and as a feedback platform will be introduced further in Section 3.

2.4. The style of written feedback

The style of written feedback varies significantly. A company that develops software on behalf of other companies receives feedback digests, pre-selected user feedback or feedback reported by their customers, whereas a company, that develops software for professional users, receives a single message covering all ideas [Pagano and Brügge 2013]. Since the variety of software users has changed from software professionals to practically any person [Grudin 1991], the people giving feedback are not most likely software professionals, like before.

Because users can now easily submit their feedback, review new releases, report bugs, and rate apps and its features, and request new features [Maalej *et al.* 2015], feedback does not usually follow any structured form [Stade *et al.* 2017]. Morales-Ramirez [2013] states that users would not feel fully stimulated if they had to give feedback following a certain structure, and this is a problem for developers. According to Pagano and Brügge [2013] user feedback written in natural language can especially cause problems, since those texts are typically written from a subjective perspective which requires developers to get into the user's mind to be able to reproduce their issue or request. Moreover, Maalej *et al.* [2015] state that developers and analysts can hardly use user feedback, in particular negative feedback, since it lacks context.

Additionally, Maalej *et al.* [2009] argue that users do not often know exactly what they want and they cannot always communicate their problems and needs in a clear and accurate way, because requests and reports often lack context information and cannot be

easily understood by the development team. Moreover, user value is a subjective measure and it is a combination of many factors and may be different for different stakeholders [Geer and Ruhe 2003]. A feature necessary for one user may not be useful for another, and there needs to be a balance between various users' requests. In addition to different opinions on different features, users might mention all their opinions in one review.

Furthermore, there may be individual ways to refer to the same feature, and users tend to use sarcasm often. [Maalej *et al.* 2015] These characteristics make natural language feedback difficult to automatically categorize, and sometimes for developers to understand. For example, if a message says "Well, this new feature is great", there is no clarification of which feature or not even certain if the user uses sarcasm or not.

2.5. Feedback management in software companies

Little is known about how software companies work with user feedback, especially when a large number of users are involved. There is a lack of knowledge on how the feedback is collected and what are the benefits and challenges when operating with feedback channels. [Stade *et al.* 2017; Pagano and Brügge 2013] Greer and Ruhe [2003] suggest that user involvement is one of the challenges in the requirements engineering process and according to Seyff *et al.* [2010] it has only recently begun to raise interest in the requirements engineering community. Receiving timely and accurate user feedback has been experienced as difficult, even though it would be crucial in order to do processes, such as requirements prioritization, successfully. It is argued that getting feedback from users is a slow process which also lacks mechanisms which would allow efficient user data collection and analysis. [Olsson and Bosch 2014]

After studying the different methods of using user feedback, Pagano and Brügge [2013] concluded that the companies manage feedback in the following six-step way: The first step is merging the information given over different feedback channels, like email, phone calls, and app store. In the second step, developers read messages and extract the included suggestions, and the third step is deciding if the feedback reports a problem or requests a feature. The fourth step includes developers assessing the individual priority of the feedback, and in the fifth step developers estimate the impact of the user feedback by investigating how frequently it occurs. Finally, in the last step developers connect with conventional development tools and workflows. Stade *et al.* [2017] conclude that software companies do not fully exploit the potential of feedback gathering for software development and evolution, even though research and industry provide solution ideas for feedback gathering and analysis.

Maalej *et al*. [2015] suggest that in the future, previously known and used data sources such as business requirements, system and technical requirements, stakeholder preferences, and requirements interdependencies, will be combined with aggregated user data. Maalej *et al.* [2015], Stade *et al.* [2017], and Pagano and Brügge [2013] all state that

feedback analytics tools will help to manage with a large amount of user feedback by classifying, filtering and summarizing them. However, Pagano and Brügge [2013] note that analytics tool would have less impact on companies which only receive a small amount of feedback. But for companies which receive a large amount of feedback, Maalej *et al.* [2015] suggest that automatically collected user data, logs, and interaction traces could improve the feedback quality and assist developers to understand and react to it. Moreover, Maalej *et al.* [2015] state that there is no answer to the question of how could practitioners use the information provided by user feedback and integrate it into their processes and tools to decide about when the next release should be offered and what requirements and features should be added or eliminated. Evaluating user needs is a subtle process, and even the companies with special processes for gathering user input are not always successful [Maalej *et al.* 2009].

3. App store reviews

3.1. An overview of app stores

As mentioned in the introduction, app stores are a significant part of a smartphone ecosystem. The meaning of app stores is to provide different applications to download. From a developing perspective, app stores serve as a launching platform. Furthermore, Jansen and Bloemendal [2013] define app store as "an online curated marketplace that allows developers to sell and distribute their products to actors within one or more multisided software platform ecosystems". The impact of app stores in software business can be considered from at least three perspectives. First, the awareness of software business rises with the number of people exposed to the app business. Secondly, low prices of apps in the app stores courage business models to change radically to add value to the product as to generate as much revenue as they did before app stores. Finally, when building up healthy software ecosystems, app stores appear to be the method of choice. [Jansen and Bloemendal 2013; West and Mace 2010; Idu *et al.* 2004; Hyrynsalmi *et al.* 2012]

There have been multiple app stores, based on the provider of the phone: Apple's App Store [2020], Google Play [2020] of Android phones, BlackBerry's BlackBerry World [2020], and the Windows Phone Store [2020] for Windows Phones. However, the Windows Phone Store and BlackBerry World both closed by the end of 2019. Therefore, the most common ones are App Store and Google Play.

Even though App Store and Google Play are the most known and used app stores, they are not the only ones. There are many Chinese app stores for Android phones, such as MyApp (Tencent) [2020], 360 Mobile Assistant [2020] and Xiaomi App Store [2020]. Other alternative app stores are, for example, CodeNgo [2020] (for Android phones), AppBrain [2020] (for Android phones), Cydia [2020] (for iOS phones), and GetJar [2020]

(for iOS phones). The key difference between these alternative app stores and App Store and Google Play is the revenue share model, which tends to be better for developers. App Store and Google Play offer a standard 70/30 split.

Figure 4 has two screenshots of an application view on Apple App Store. In Figure 4, it can be seen that already the first review may have some useful information for developers, because it states that "I would love absolutely love to see a couple features added that I believe wouldn't be too difficult!".

Figure 4. Screenshots of an application view on App Store [2020]

Figure 5 presents screenshots from similar views taken from Finnish localization of Google Play. The first screenshot tells that the application has been updated on 2.3.2020, and the new version has a dark mode. It also suggests the user to review the application by having five empty starts and a highlighted text telling to write a review.

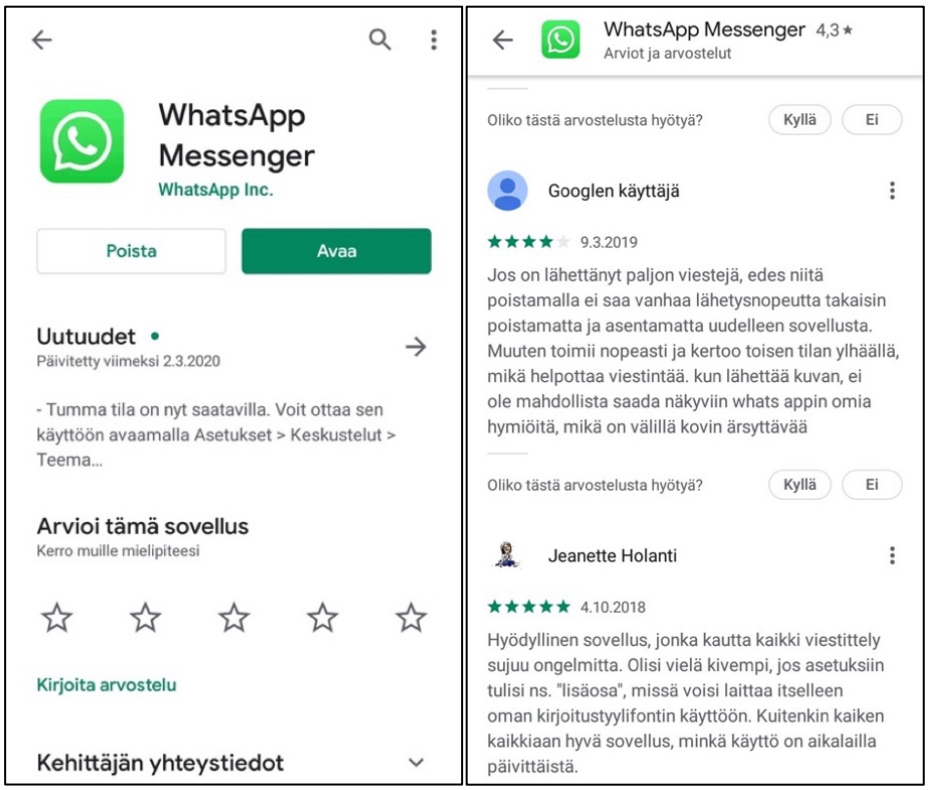

Figure 5. Screenshots of an application view on Google Play [2020]

In the second screenshot, there are two reviews visible. The one by *Googlen käyttäjä* can be translated as following: “If many messages have been sent, even deleting them won’t bring back the old sending speed, but removing and reinstalling the application has to be done. Otherwise, works fast and tells others’ state at the top, which makes communication easier. When sending a picture, it is not possible to see the own emojis of WhatsApp, which can sometimes be very annoying”. The second review, by *Jeanette Holanti*, is loosely translated into English as: “Useful application, which makes all the messaging work problem-free. It would be even nicer, if there was a so-called ‘supplement’ in the settings, which would allow using its own font. Otherwise, all in all a good application, which usage is quite a lot daily”. Each review also has an option to rate the review either as useful or not.

3.2. The value of the reviews from developers’ perspective

Since users can nowadays submit their feedback regarding software products in app stores, social media, or user groups [Maalej *et al.* 2015] the amount of feedback has increased. When it comes to app stores, Galvis Carreño and Winbladh [2013], Harman *et al.* [2012] and Pagano and Maalej [2013] showed that in addition to the high number of reviews, the feedback also contains key information to developers, such as user requirements, bug reports, feature requests and documentation of user experiences with specific app features. This can be seen already in the previous section, where three of the

three reviews shown in two randomly taken screenshots from app store reviews included a feature request. Therefore, app store feedback provides similar information as general feedback which makes the reviews as significant from developing points of view. Additionally, using app store reviews in application design processes can be less time consuming and more informing if used instead of classic software engineering methods, such as focus groups, interviews, or questionnaires. These require users to be present and they might be subject to questioning or analysis. On the contrary, app store reviews provide direct feedback from users without the need to interact with the developing team. Writing a review does not require the user to be in a laboratory setting and this makes it less biased, even though it can be less structured. [Iacob *et al.* 2013]

An average free mobile application on Apple's AppStore [2020] receives 36.87 daily reviews, whereas paid apps only 7.18. The difference may result from the larger user communities of free application. In addition, larger user communities have also effect on categories, which receive most reviews: popular categories like Games receive median 31.24 reviews per day, unlike less popular category, Books, which has a median of 0.53 reviews. However, the number of daily reviews can differ significantly, because large applications, like Facebook, receive even 4,275 reviews in one day. The median length of an average review is 61 characters and the mean is 106.09. Moreover, over 80% of the reviews contain fewer characters than a text message, which has a maximum of 160 characters. Therefore, the application feedback is more similar to short messages, such as a tweet, than to other communication artifacts such as an email. User feedback is also triggered by new releases, because most feedback is given the first few days after a release, leading to a long tail over time. Furthermore, users tend to write less the more they like and application, signing that there would be less to improve. [Pagano and Maalej 2013]

The quality of reviews varies strongly, and the content of a review tends to become insulting rapidly. For instance, Chen *et al.* [2014] state that the proportion of informative user reviews is relatively low, meaning that 35% of app reviews included in their study contained information that could be directly helpful for developers to improve their apps. In addition, Maalej *et al.* [2016] add that the majority of app store reviews are rather non-informative praising or repeating to the star rating words. Pagano and Brügge [2013] also state that user feedback, especially app store reviews, has poor quality, and a substantial part of the feedback is unqualified and does not prove any value to developers. However, the value of feedback can only be estimated after reading or browsing it, leading to the reader's frustration if the quality is poor, and therefore increasing the negative associations with user comments. [Pagano and Brügge 2013] Chen *et al.* [2014], Pagano and Brügge [2013], Gärtner and Schneider [2012] and Morales-Ramirez [2013] all state that the amount of app review data is too large and time-consuming to analyze manually.

However, app stores can still serve as a communication channel among users and with developers [Pagano and Maalej 2013]. For example, according to Iacob and Harrison [2013], feature requests are strongly present in reviews. In their study, they found that 23% of the reviews included a feature request. Even though the portion may not seem large, it should be taken in concern that that percent can mean over thousands of feature requests, if the total amount of received reviews is high, as it is in some popular applications. In addition to feature requests, a review can include a bug report such as a problem description, crash, erroneous behavior or a performance issue. A review can also have significant information regarding user experience, such as documentation of a particular experience with the app of its feature. [Maalej *et al.* 2015] Additionally, because reviews are usually written in a natural and not structured form, one review can express many issues, meaning that one review may not only include one feature request, but many requests, bug report and user experience. Therefore, mobile app reviews are valuable repositories of ideas coming directly form app users [Iacob and Harrison 2013].

The contrast between the value of app store reviews is relatively high, since others believe that it is not useful for developers but others underline its meaningfulness. However, there is a common problem between both of the ideologies: the amount is too high to manually analyze all of them. Analysts and developers would profit from tools an methods which would systematically filter and aggregate feedback [Pagano and Maalej 2013]. Based on the fact that almost a quarter of the reviews included a feature request in their study, Iacob and Harrison [2013] also state that there is a need for support in automatically retrieving feature requests from online reviews. Additionally, Iacob *et al.* [2013] argue that app stores should now have a permanent place in the mobile application development cycle, clarifying that the data they offer should be taken in concern.

4. App store review analyzing tools

The following app store review analyzing tools are introduced based on the information available on each tool and the tools were chosen objectively for the review. The goal of covering the following tools is to reflect research questions “What kind of tools can software companies use to manage app store feedback?” and “What are ideal features in app store review analyzing tools in software industry perspective?”.

4.1. Commercial app store review analyzers

There are already multiple app store review analyzing tools on the market. However, they do not provide detailed descriptions of their functionality and they may not only focus on managing the reviews, so they will be introduced briefly. The two tools covered both offer tree trials, which were exploited in this thesis.

4.1.1. Appbot

Appbot [2020], released in 2011, monitors and analyzes app reviews from iTunes, Google Play, Windows and Amazon. Moreover, Appbot [2020] can also be utilized for monitoring and analyzing product reviews on Amazon.com. Appbot provides automated easily understood sentiment analysis for both app and product reviews. Trends such as user sentiment, review volume and star rating can be measured with Appbot to find out how users think of changes made. [Appbot 2020] Appbot [2020] provides four different types of subscriptions, from small to medium, with price range 39$ to 349$ per month. Appbot [2020] offers integrations, such as Slack and Trello, which helps its users to be updated.

The main service offered by the Appbot is its sentiment analysis, which is artificial intelligence (AI) based. Appbot's [2020] sentiment analysis AI has been specifically trained on large customer feedback sets and tested by teams from popular brands of the world. However, it is not specified, which brands were included in the testing. The sentiment analysis AI is designed to answer questions such as "How did our recent app update affect sentiment" and "What do customer think of our new feature?" [Appbot 2020]. According to Appbot's [2020] website, sentiment analysis is important for brands to understand how users feel about their product or service. Appbot [2020] is promised to make customers' sentiments clearer by providing information on why the customers feel the way they do, with advanced text analysis. Appbot [2020] reads all app reviews, support tickets, survey results and any other types of feedback connected to the account, and given feedback is grouped together based on sentiment, topics, and keywords.

The free trial offers a possibility to track applications and see some of their data. The data is separated into Analysis, Groupings and Audience. Analysis and Groupings are available for public applications, and some of the subsections of the audience, so they will be covered. The Analysis includes five different subsections: reviews, ratings, sentiment, dashboard, and comparison. The Grouping has four different subsections: words, topics, customer topics, and tags. The Audience has also four subsections: versions, countries, languages, and emotions. Figure 6 presents the opening page of the logged user. First, the Analysis section will be covered, followed by the Grouping section. Finally, the available features of the Audience will be introduced.

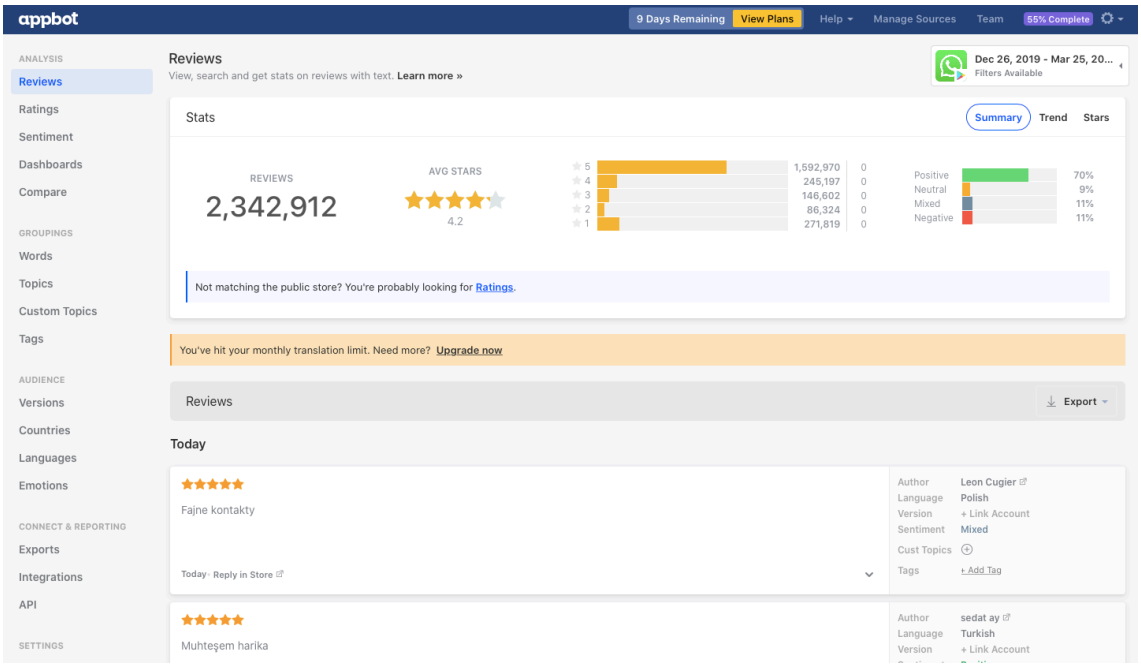

Figure 6. The opening page of the logged user of Appbot [2020]

The first feature of Analysis is reviews. Reviews can be filtered by the app, date, keyword, rating, sentiment, language and topic. Reviews can also be translated so they can be understood even if they were written in a foreign language. Figure 7 presents reviews after filtering.

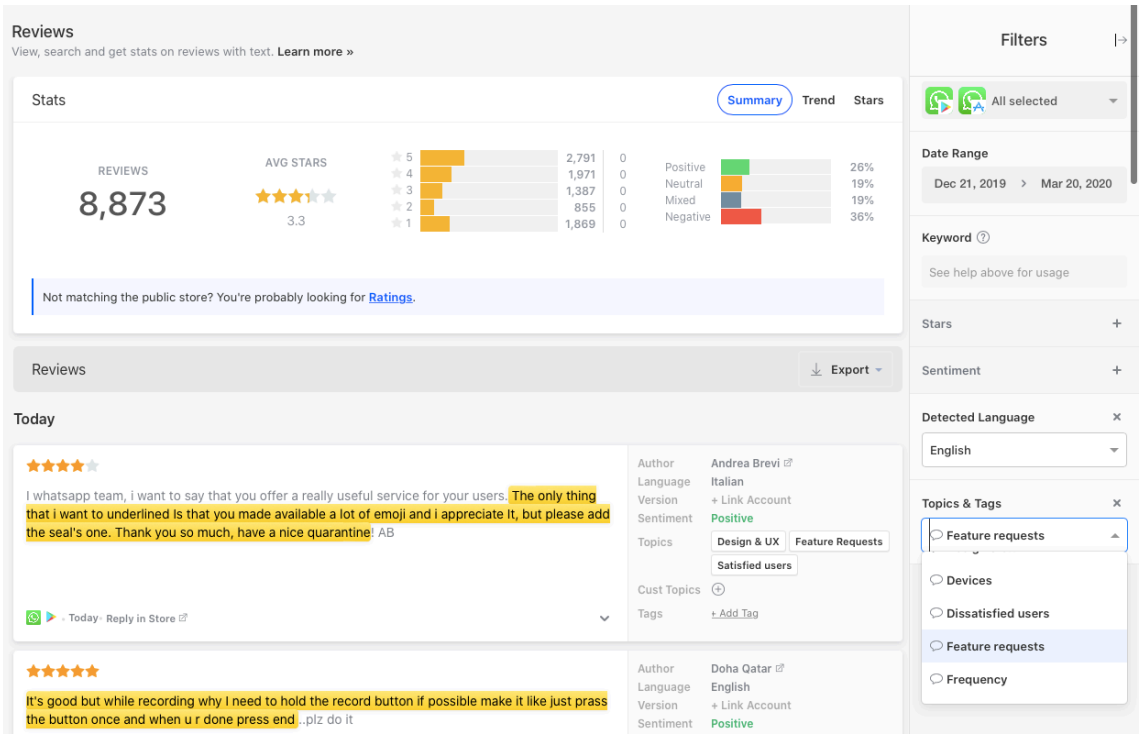

Figure 7. Filtered reviews of WhatsApp [Appbot 2020]

Rating page shows a graph presenting the change of the ratings during a certain period. Ratings can be viewed either based on public store ratings or private developer console ratings. The sentiment view offers information such as a bar chart of sentiment timeline, overall sentiment score and sentiment breakdown. Sentiments can also be filtered by date, application, keyword, star rating, language, topics, and if the review was replied or not. The final subsection of the Analysis is the comparison, where different applications can be compared. Review volume and ratings are compared in a graph and statistics are presented as a table. The table has columns for the name of the app, sentiment, score, reviews, average star rating, rating breakdown, and trend.

The first subsection of the Groupings section is words. Words tool offers a breakdown of the words used in the reviews, and they can be viewed from six different categories: interesting, popular, critical, trending up, trending down, and new. Additionally, the tool presents a word cloud. The second subsection of the Groupings section is topics. Topics can be, for example, satisfied users, design and user experience, and bugs. Topics are presented in a table, which shows the sentiment of the topic, its mentions, overall presence in reviews and a trend curve. Topics can also be custom made, which is the next subsection. After topics, the final subsection is tags. Tags are not created automatically, and reviews must be manually tagged.

The audience offers information regarding versions, but they are not available for public use. Countries can be viewed only for App Store applications, and that page presents information of reviews by country. Language tool is similar to countries, but instead of countries, it presents the reviews by the language they were written. Emotions tool shows the user emotions of individual reviews based on their negativity, positivity, and if they are more passive or assertive.

Overall, Appbot [2020] offers a lot of information from different perspectives. The design of it is clear and using it can be learned quickly. Based on the free trial, it can certainly offer useful information for developers, such as review filtering, sentiment analysis, and grouping. Those features can easily be utilized when analyzing and managing app store reviews, which would lead to exploiting more user data in development process, leading to better products.

4.1.2. AppFollow

AppFollow [2020] is similar to Appbot, and it's development begun on a hackathon in 2014. What makes a significant difference between Appbot [2020] and AppFollow [2020] is AppFollow's feature in which users can reply to app reviews. With AppFollow [2020], its user can both receive and reply reviews from platforms such as Slack, Zendesk and Helpshift. To help product providers to understand and reply to the reviews, AppFollow [2020] provides an auto-translation tool which translates the content to and from 31 languages. In addition, if a user updates or deletes their comments, it is showed to the

user of AppFollow [2020]. Reviews can be tracked based on the affect they have user's app, for example, user problems can be quickly fixed to raise the rating. Users can additionally choose the reviews they want to receive (positive, negative, all) the moment they were written, such as daily, weekly or monthly reports in Slack or email. Moreover, reviews can be analyzed by app, country, language, rating, user, status or device. [AppFollow 2020] AppFollow [2020] also has a semantic analysis tool to help providers to get a deeper understanding of how their users feel about the app, to monitor the most used words and change of users' attitude over time.

AppFollow [2020] offers multiple features for app store management. The opening page of the logged user of AppFollow [2020] is presented in Figure 8.

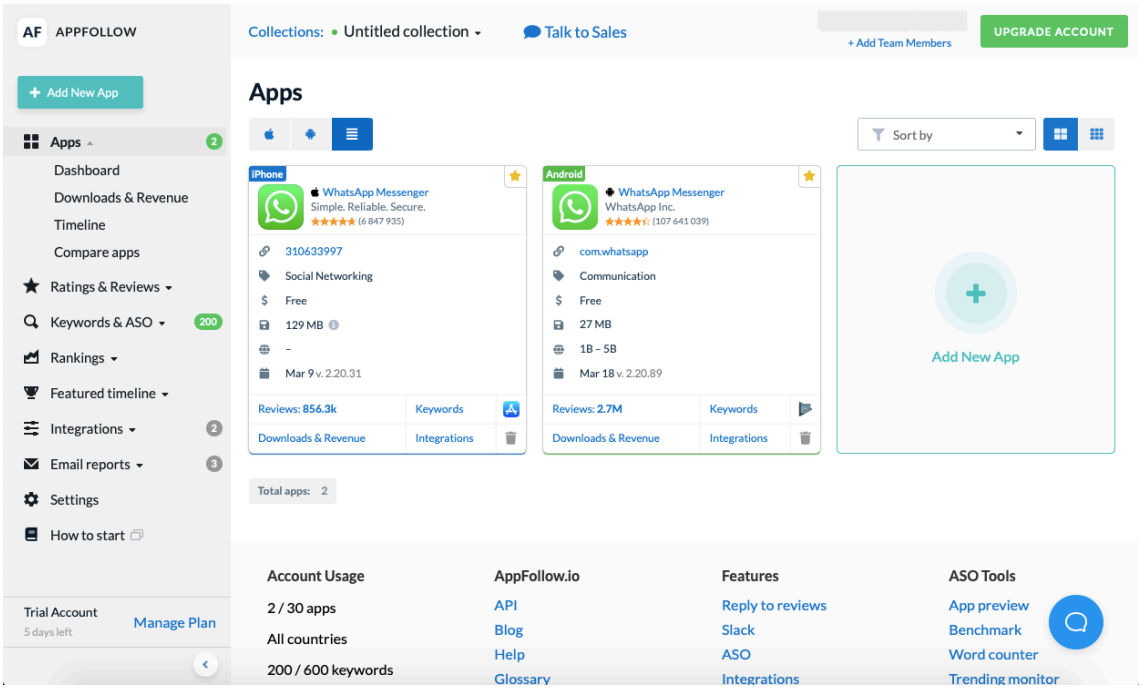

Figure 8. The opening page of the logged user of AppFollow [2020]

The menu is divided into 8 sections: Apps, Ratings & Reviews, Keywords & ASO, Rankings, Featured timeline, Integrations, Email reports, Setting, and How to start.

The AppFollow has four subsections: Dashboard, Downloads & Revenue, Timeline, and Compare Apps. The dashboard gives overall information regarding rankings, keywords, star rating, reviews, downloads, and revenue. Downloads and revenue are not public information. The timeline presents the information on the latest releases. App comparison compares the applications based mostly by their releases and the amounts of versions.

Ratings and reviews page is presented in Figure 9. Many of the subsections of that section are in the beta phase and therefore not available for use. Those features are Reviews Dashboard, Semantic Analysis, Auto-tags, Auto-replies and Reviews import.

Rating chart presents ratings in different charts, such as in bar and pie charts. They can be viewed based on country or version. Reviews chart presents a graph of star ratings, based on the amount and the date of the review. Additionally, reviews can be viewed in a pie chart based on the country, tags and the application. Tags can be manually made.

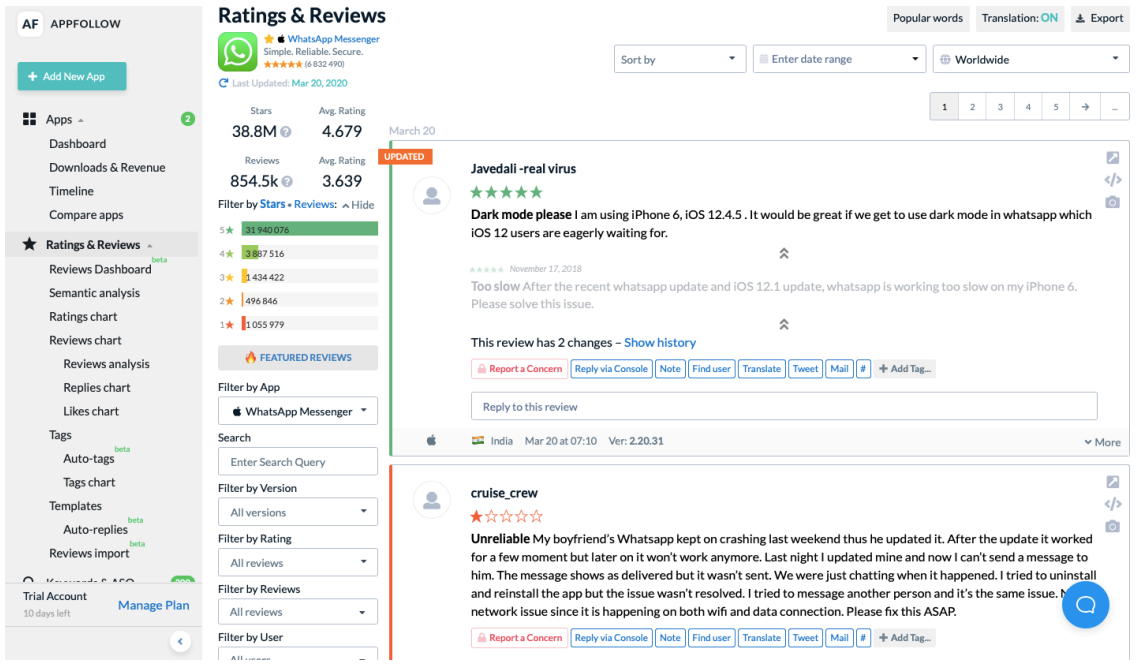

Figure 9. Ratings and Reviews on AppFollow [2020]

Keywords and ASO provide competitors overview, keyword research, keyword spy, ASO analytics, traffic score, word counter, app preview, and conversion benchmark tools. Keyword research shows the most suitable keywords for the applications, so they could be the most easily found in app stores. Keyword spy shows other keywords, which are used to find the application. Traffic score can only be used on App Store products, and it shows recommended keywords for application search. Word counter counts the words of submitted text and app preview shows how the application looks in the app store.

Rankings show the order of different types of applications: paid, free, or based on their category. It also presents a diagram of specific application rankings in different categories and how the position has changed.

The featured timeline tells if the application has been present on daily lists that app stores provide on applications. Integrations can be created so that users can be notified of the changes easier. Email reports provide tools for connecting email and receiving reports. Reports provided can be sent to the right team, for example, bug reports can be sent to developers, complaints to customer supports team and feedback to the owners of the product.

As a conclusion, AppFollow [2020] provides multiple useful tools for app review analyzing. Review filtering, semantic analysis, and keyword analysis can help companies

to develop their product and revenue. Additionally, replying to reviews can add more positive sentiments towards the provider from the user, since receiving a reply makes users feel like they have been heard.

4.2. Tools chosen for review

The four app store review analyzing tools which will be introduced next have multiple differences but also some similarities. They all have different approaches to the problem: The aim of the Mobile App Review Analyzer by Iacob and Harrison [2013] is to find only feature requests provided by app store reviewers, whereas the sentimental feature analyzer by Guzman and Maalej [2014] attempts to inform application provider about all sentiments which their users have regarding the features of the application. The Framework for App Review Mining by Chen *et al.* [2014] approaches the problem of managing a large amount of user feedback in app stores by filtering the reviews based on their informativity, and Maalej and Nabil [2015] and Maalej *et al.* [2016] did not focus on building one possible solution, but on comparing various tools and methods, which could be used in building such a tool. However, as mentioned, they all have some similar characteristics, which was the reason why they were chosen for this review.

4.2.1. Mobile App Review Analyzer (MARA)

Iacob and Harrison [2013] designed Mobile App Review Analyzer (MARA) as a prototype for automatic retrieval of mobile app feature requests from online reviews. MARA is designed to exploit users sentiments regarding the feature request, identifying pre-defined linguistic rules and evaluating on a large sample (161 apps with a total of 3279 reviews) of online reviews. Iacob and Harrison [2013] state that in addition to the number of reviews associated with one app, another challenge is the individual style of each review. Usually, the reviews are short, but they tend to lack structure and not obey grammar and punctuation rules. Additionally, users may use such humor forms as sarcasm while writing reviews, which can in some cases cause even human reader trouble with understanding, let alone computer.

Iacob and Harrison [2013] begun the process by considering a sample of reviews for analysis and identifying feature requests by counting how many of the reviews included a feature request and how the users expressed their issue. The findings informed the design of a prototype system, which Iacob and Harrison [2013] evaluated using a large sample of reviews and LDA [Blei *et al.* 2003] to identify common topics across the feature requests obtained as a result of this evaluation. For each app selected for evaluation, Iacob and Harrison [2013] automatically extracted and stored the reviews provided for it by users. In addition to the review collected, metadata was also stored: the posting date, the rating user gave, the device the review was posted with, the version of the app and the title given for the review.

Figure 10 presents the design of the structure of MARA.

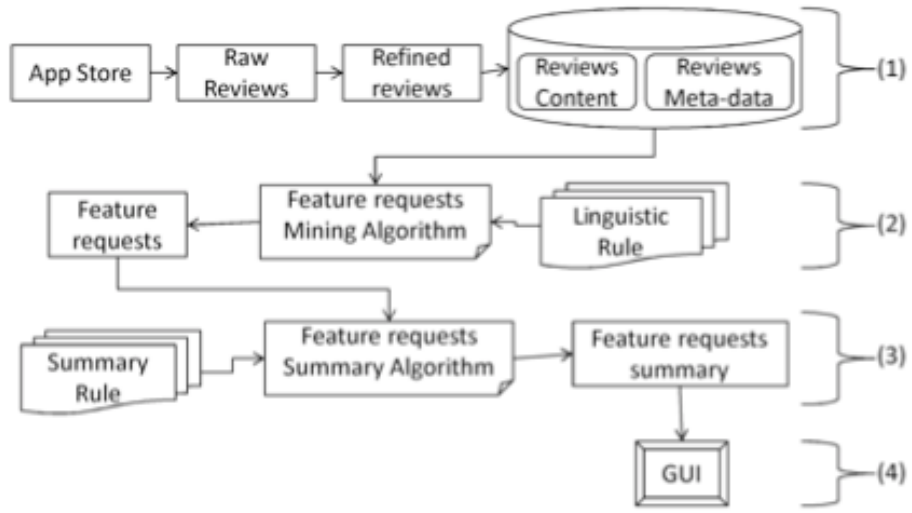

Figure 10. The architecture of the prototype of MARA [Iacob and Harrison 2013]

First, the system retrieves all the reviews. Secondly, the content of the reviews is mined for identifying sentences or fragments of sentences expressing feature requests. Third, content is summarized and lastly it is presented. [Iacob and Harrison 2013]

To filter feature requests out of other reviews, Iacob and Harrison [2013] defined a language for expressing requests. The language was created from the sentences labeled as a feature request by identifying a keyword from each sentence, which denotes the sentence as a request. To avoid accidental associations, the keywords chosen to use were the ones that were associated with more than 3 sentences. Finally, there were 24 keywords: add, allow, complaint, could, hope, if only, improvement, instead of, lacks, look forward to, maybe, missing, must, needs, please, prefer, request, should, suggest, waiting for, want, will, wish, would. 80% of the sentences included the listed keywords. When the content of a review and the metadata of the review were collected, the content was split into sentenced using LangPipe [LingPipe 2019], and normalized to reduce the noise in the final results. The split review's content is given as input to the feature request mining algorithm which uses a set of linguistic rules defined for supporting the identification of sentences which refer to such requests [Iacob and Harrison 2013]. "For example, feature requests are more prone to be expressed in sentences such as 'Adding an exit button would be great', which translated to a linguistic rule of the form 'Adding <request> would be <POSITIVE-ADJECTIVE>'", Iacob and Harrison [2013] state.

In the summarization phase, a set of predefined rules are used to summarize the extracted feature requests. More frequent and lengthier requests are would be shown up first in the summary to help the reader to see the most requested issues first without having to search for them from the large number of reviews. [Iacob and Harrison 2013]

Iacob and Harrison [2013] evaluated the design of MARA by considering precision, recall, and Matthews correlation coefficient (MCC). The metrics are defined as the following:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$MCC = \frac{TPXTN - FPXFN}{\sqrt{(TP + FP)(TP + FN)(TN + FRP)(TN + FN)}}$$

where TP presents true positives (results which are actual feature requests), TN true negatives (non feature requests which were not returned as results), FP false positives (results which are not feature requests) and FN false negatives (feature requests which did not appear in the results). For evaluation purposes, Iacob and Harrison [2013] downloaded 136,998 reviews from Google App Store, gave them as input to for the feature mining algorithm and evaluated the results using the metrics presented above. Out of 3000 feature requests resulted by the algorithm, a human coder identified false positives. The precision of the sample was P = 0.85. To evaluate MCC and recall, a random app was selected and its reviews were used for measuring the two metrics. 480 reviews were split into 778 sentences, and the results are shown in Figure 11.

Inputs	**TP**	**FP**	**TN**	**FN**	**R**	**MCC**
778	65	3	701	9	0.87	0.9

Figure 11. Recall and MCC metrics [Iacob and Harrison 2013]

4.2.2. Sentimental feature analyzer

Guzman and Maalej [2014] produced an analyzer which tells its user opinions of the users of an application on various features. First, it produces a fine-grained list of features mentioned in the reviews. Secondly, it extracts the user sentiments of the identified features and gives them a general score across all reviews. Finally, the analyzer groups the fine-grained features into more high-level features that tend to be mentioned in the same reviews and shows the sentiments of users about these high-level features. Guzman and Maalej [2014] chose to use collocation finding by Manning and Schütze [1999] for extracting the fine-grained features, sentiment analysis by Thelwall *et al.* [2010] for extracting the sentiments and opinions associated to the features, and topic modeling by Blei *et al.* [2003] for the grouping of related features.

The first step is collecting and preprocessing user reviews. The reviews were collected from both Apple App Store and Google Play, however, by using different tools. For Apple App Store reviews, a modified version of an open source scraping tool was used. On the other hand, for Google Play, a tool using Google Play Application Programming Interface was developed. The data was collected from 32210 reviews for seven iOS and Android applications. After collecting the data, the title and comment were extracted from each review. [Guzman and Maalej 2014]

The feature extraction includes three steps: Noun, verb and adjective extraction, stopword removal, and lemmatization. For the first step, Guzman and Maalej [2014] used the part of speech tagging functionality of the Natural Language Toolkit, NLTK [2019]. For stopword removal, Guzman and Maalej [2014] chose to use the standard list of stopwords provided by Lucene [2019] and in addition to common English language stopwords (e.g., “and”, “this”, and “is”) Guzman and Maalej [2014] expanded the set with words which are common is user reviews, but not used to describe features. For example, words like “app”, “please”, and “fix” were added to the set. For the last section of feature extraction, lemmatization, Guzman and Maalej [2014] used the Wordnet [Miller 1995] lemmatizer from NLTK [2019] for grouping the different inflected forms of words with the same part of speech tag which are syntactically different but semantically equal. This process concluded that the same word in different phrasing was grouped into the same lemma. For example, the terms describing the verbs “sees” and “saw” are grouped into the term “see”. [Guzman and Maalej 2014]

To find features from the user reviews, Guzman and Maalej [2014] used the collocation finding algorithm by the NLTK [2019] toolkit. A collocation is a collection of words that are often occurred together [Bird *et al.* 2009], for example, <strong tea> is a collocation, because the two words are commonly used together in the English language, unlike <powerful tea>. Features can generally be described as collocations, as they are normally a pair of words. Examples of collocations that are application features are <pdf viewer>, <user interface> and <view picture>, which all are concluded from two words. After finding the collocations they were filtered by taking into consideration only those that appear in at least three reviews and that have less than three words distance between them. [Guzman and Maalej 2014] To find synonyms Guzman and Maalej [2014] used Wordnet [Miller 1999] as a synonym dictionary. Wordnet was also used to group collocations with misspelling together. When grouping features together Guzman and Maalej [2014] considered the word collection with the highest frequency to be the name of the feature. For example, if there are the following word collections: <picture view>, <view photograph> and <see photo> with a frequency of 30, 10, and 4, the approach would be grouping these features together as synonyms. Afterwards, the one with the highest frequency would be chosen as the name of the feature, in this case <picture view>.

Sentiment analysis is the process of assigning a quantitative value (positive or negative) to a piece of text expressing an affect or mood of it [Onur Kucuktunc *et al.* 2012]. Guzman and Maalej [2014] chose to use SentiStrength [Thelwall *et al.* 2010] for analyzing sentiments in user reviews. SentiStrength is a lexical sentiment extraction tool specialized in dealing with short, low quality text. According to Thelwall *et al.* [2012] SentiStrength has good accuracy for short texts in social media, such as Twitter. SentiStrength divides the review text into sentences and then assigns a positive and negative value to each sentence, because humans can express both positive and negative sentiments in the same sentence. The review is scored either in a positive score, between 1 and 5, where 5 is extremely positive and 1 absence of sentiment, or negative, where -1 is an absence of negative sentiment and -5 denotes an extremely negative sentiment. The sentiment score of the whole sentence is computed by taking both the maximum and the minimum score among all the words in a sentence and creating a pair of numbers, like {1, -3}. [Guzman and Maalej 2014] The sentiment score of the sentences is exploited in computing the sentiment score for the features. The sentiment score of a feature is equal to the score with the sentence in which it is present. The feature score was chosen to be the score with the maximum absolute value. In the case that the positive and negative values are the same, the negative value is assigned to the feature. After the sentiment analysis step, there was a list of all extracted features, their frequencies, and their sentiment scores. [Guzman and Maalej 2014]

Finally, in the grouping phase, Guzman and Maalej [2014] used the Latent Dirichlet Allocation [Blei *et al.* 2003], also known as LDA, as the topic modelling algorithm. In LDA a topic is a probabilistic distribution over words and each document is modeled as a mixture of topics. Therefore, each review can be associated with different topics and topics are associated with different words with a certain possibility. An example of a topic can be the set of words which describe users' experiences when updating a faulty app, such as {crash, update, frustrated, newest, version, help, bug}. [Guzman and Maalej 2014] Normally LDA algorithm is given the words forming the vocabulary of analyzed reviews as an input, but Guzman and Maalej [2014] gave the list of extracted features and model each feature as a single word. An example of a topic given could then be the set of features {picture_view, camera_picture, upload_picture, delete_picture} which describes features related to manipulating pictures in an application. Guzman and Maalej [2014] calculated topic sentiments as follows: Let $R = \{r_1, r_2, \cdots, r_n\}$ be the set of analyzed reviews and $T = \{t_1, t_2, \cdots, t_m\}$ the set of extracted topics. The final output of the LDA computation is the matrix $W_{n\times m}$, where $w_{i,j}$ contains the number of times a feature mentioned in a review r_i is associated with topic t_j. Guzman and Maalej [2014] then used a weighted average to calculate the sentiment score of each topic. For every topic t_j they calculated the topic sentiment score ts_j as:

$$ts_j = \frac{\sum_{i=1}^{n} w_{i,j} \cdot s_i}{\sum_{i=1}^{n} w_{i,j}}$$

where $S = \{s_1, s_2, \cdots, s_l\}$ denoted the sentiment score of each feature associated with the topic t_j.

As evaluation criteria, Guzman and Maalej [2014] used precision, recall and F-measure. Precision was computed by dividing the number of true positives by the sum of true positives and false positives, and recall was computed by dividing the number of true positives by the sum of true positives and false negatives. To count the F-measure, Guzman and Maalej [2014] used the general form of the measure, which combines the precision and recall results, for its computation. Additionally, there were 9 human coders involved in creating the truth set. They all coded 2800 randomly sampled user reviews to extract feature requests or feedback about an existing feature, to identify the app feature mentioned in the review, and to assess the sentiment associated to each feature. However, after running the feature extraction step of the approach Guzman and Maalej [2014] obtained a list of word sets designating app features. Many of the word sets extracted as features contained words that do not describe features but rather the general opinion or sentiments of users (e.g., great, bad, good, like, hate…). To filter these words it was decided to slightly modify the approach and include all words that are assigned a sentiment by lexical sentiment analysis tool into the stopword list. The first approach is presented as F_S and the latter as F_{NS} in Figure 12.

App	Precision	Recall	F-measure
	F_S		
AngryBirds	0.335	0.332	0.334
Dropbox	0.608	0.475	0.533
Evernote	0.474	0.416	0.443
TripAdvisor	0.421	0.399	0.410
PicsArt	0.750	0.669	0.707
Pinterest	0.644	0.623	0.634
Whatsapp	0.843	0.728	0.781
F_S Average	**0.582**	**0.520**	**0.549**
	F_{NS}		
AngryBirds	0.368	0.321	0.343
Dropbox	0.603	0.473	0.531
Evernote	0.451	0.389	0.418
TripAdvisor	0.403	0.370	0.386
PicsArt	0.815	0.661	0.730
Pinterest	0.658	0.592	0.623
Whatsapp	0.910	0.734	0.813
F_{NS} Average	**0.601**	**0.506**	**0.549**

Figure 12. Metrics of F_S and F_{NS} [Guzman and Maalej 2014]

4.2.3. Framework for App Review Mining (AR-Miner)

Chen *et al.* [2014] created a novel computational framework for App Review Mining, a tool called AR-Miner. Overview of AR-Miner is the following: First, the raw user review data is preprocessed into a well-structured format to facilitate subsequent tasks. Secondly, a pre-trained classifier filters non-informative reviews out. Third, the reviews which are semantically similar are grouped into own groups. Fourth, groups and the reviews belonging to them are sorted by the level of importance by using a novel ranking model. Finally, the ranking results are visualized and an intuitive summary is presented. The overview is pictured in Figure 13. The experiments and case studies were done using Android apps.

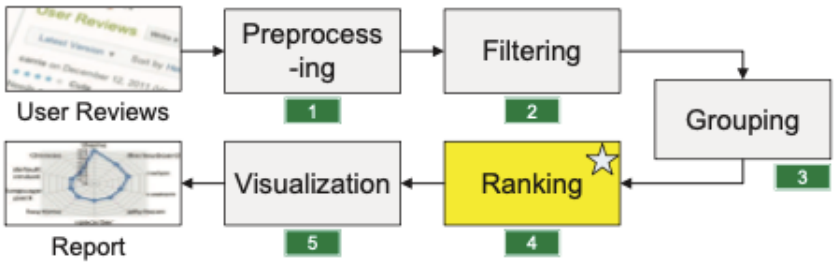

Figure 13. Overview of AR-Miner [Chen *et al.* 2014]

Since the non-informative reviews are filtered out, the difference between non-informative and informative feedback must be defined. The definition of an informative review is classified based on if the review contains information that app developers are looking to identify and is potentially useful for improving the quality or user experience. However, even for developers, it would be possible that no two people would have the same understanding of the definition of informative. To overcome this thread, online forums were studied to identify what kinds of information do real app developers consider as constructive. [Chen *et al.* 2014]

The first step of creating AR-Miner is preprocessing which converts the raw user reviews into sentence-level review instances. The format of a raw user review included the text written in the review, the rating given, and the time the review was posted. The preprocessing step also preprocesses the text out of the data of a raw user review. [Chen *et al.* 2014] For splitting sentences, Chen *et al*. [2014] used a standard sentence splitter provided by LingPipe [2019]. Each split sentence was generated values of rating and timestamp, based on the values of the attributes in the review the sentence was a part of. The preprocessing of the text includes tokenizing the text, removing all non-alpha-numeric symbols, converting words to lowercase and eliminating extra whitespace along with stopwords or rare words. The remaining words are stemmed to their root form and the sentences which became empty after the process was removed. [Chen *et al.* 2014]

The second phase is filtering non-informative reviews out of informative ones [Chen *et al.* 2014]. The rules are summarized in Figure 14.

Class	Type (Rule)	Real Example
Informative	Functional flaw that produces incorrect or unexpected result	None of the pictures will load in my news feed.
	Performance flaw that degrades the performance of Apps	It lags and doesn't respond to my touch which almost always causes me to run into stuff.
	Requests to add/modify features	Amazing app, although I wish there were more themes to choose from.
		Please make it a little easy to get bananas please and make more power ups that would be awesome.
	Requests to remove advertisements/notifications	So many ads its unplayable!
	Requests to remove permissions	This game is adding for too much unexplained permissions.
Non-informative	Pure user emotional expression	Great fun can't put it down!
		This is a crap app.
	Descriptions of (apps, features, actions, etc.)	I have changed my review from 2 star to 1 star.
	Too general/unclear expression of failures and requests	Bad game this is not working on my phone.
	Questions and inquiries	How can I get more points?

Figure 14. Types of informative and non-informative reviews [Chen *et al.* 2014]

Chen *et al*. decided to use a well-known and representative semi-supervised machine learning algorithm, i.e., Expectation Maximization for Naive Bayes (EMNB) [Nigam *et al.* 2000] to build some classifier on the historical training data. After the classifier is built, it could be applied to filter future unlabeled user reviews. Figure 15 presents a possible good results after the filtering. R represents the rating of the review, TS the timestamp and P indicates the probability of the review belongs to the informative class. [Chen *et al.* 2014]

ID	Text	R	TS	P
r_1	Nice application, but lacks some important features like support to move on SD card.	4	Dec 09	0.8
r_3	Can't change cover picture.	3	Jan 18	0.9
r_4	I can't view some cover pictures even mine.	2	Jan 10	0.9
r_5	Wish it'd go on my SD card.	5	Dec 15	0.9
...	...	...	...	
r_n	...	...	...	

Figure 15. Possible good results after filtering. [Chen *et al.* 2014]

After the filtering step, the grouping is done. The goal of the grouping phase is to separate the reviews into several groups such that the text of reviews in a group is more semantically similar to each other than the text of reviews in other groups. [Chen *et al.* 2014] To implement the grouping step, Chen *et al.* [2014] chose to adopt topic modelling which assigns multiple topics to each review, taking into account that one sentence can include feedback regarding multiple topics.

The focus of the documentation of AR-Miner is on the ranking step. The general form of the ranking model is shown in the algorithm in Figure 16. [Chen *et al.* 2014]

Input: A set of groups $\mathcal{G}$, feature function sets $\mathbf{f}^G = \{f_1^G, \ldots, f_m^G\}$ and $\mathbf{f}^I = \{f_1^I, \ldots, f_n^I\}$, weight vectors $\mathbf{w}^G = (w_1^G, \ldots, w_m^G)$ and $\mathbf{w}^I = (w_1^I, \ldots, w_n^I)$

for *each group* $g \in \mathcal{G}$ **do**
  Compute $f_1^G(g), \ldots, f_m^G(g)$
  Set $GroupScore(g) = \sum_{i=1}^{m}(w_i^G \times f_i^G(g))$
  for *each review instance* $r \in g$ **do**
    Compute $f_1^I(r), \ldots, f_n^I(r)$
    Set $InstanceScore(r) = \sum_{j=1}^{n}(w_j^I \times f_j^I(r))$
  end
end

Output: Groups in decreasing order of $GroupScore$; review instances in each group in decreasing order of $InstanceScore$

Figure 16. The general form of the ranking model algorithm [Chen *et al.* 2014]

The inputs are a set of groups (topics) G generated by the grouping step. The sets of functions $\mathbf{f}^G$ and $\mathbf{f}^I$ measure the importance of various features of groups and reviews, like volume and rating. Two weight vectors $\mathbf{w}^G$ and $\mathbf{w}^I$ correspond to $\mathbf{f}^G$ and $\mathbf{f}^I$. The algorithm computes the $Groupscore(g) \in [0,1]$ for each group $g \in G$, and the $InstanceScore(r) \in [0,1]$ for each review $r \in$ g. The larger the $Groupscore(g)$ and $InstanceScore(r)$ are, the higher importance will be marked. The final output will be the ranking results. [Chen *et al.* 2014] Since the detailed description of the algorithm is not necessary for this thesis, the detailed information regarding to functions $\mathbf{f}^G$ and $\mathbf{f}^I$ will not be covered.

The last step of AR-Miner is to visualize the results generated by the ranking model. The visualization includes the results of 10 highest ranked results, the $Groupscore$ value of each group, and the information of review instances of the results. [Chen *et al.* 2014]

The evaluation included 4 Android apps, 1000 reviews as labeled training pool and 2000 reviews as a test set. The first set of metrics include precision, recall, and F-measure. In addition, Normalized Discounted Cumulative Gain [Croft *et al.* 2010] was adopted as a measure for evaluating the quality of the highest ranked results. Moreover, two different topic models, LDA [Blei *et al.* 2003] and ASUM [Jo and Oh 2001], were used and compared in the results. When LDA was used, the average EMNB value was 0.54, whereas when using ASUM it was 0.355, indicating that using LDA concludes better results.

4.2.4. Research on automatically classifying app reviews

Maalej and Nabil [2015] compared different methods of app review classification and Maalej *et al.* [2016] continued their work. The methods were string matching as a basic classifier, bag of words as document classification, natural language processing for text preprocessing, rating and length analyzing in reviewing metadata, exploiting sentiment

scores in sentiment analysis, and comparison between binary and multiclass classifiers with supervised learning.

Maalej *et al.* [2016] describe string matching as the most trivial technique to automatically categorize user reviews. In sting matching, a review is checked if it contains a certain keyword, which was manually defined. Each keyword was also categorized into different categories, to indicate the type of a review. For example, words “bug”, “fix” and “problem” are typed as bug reports, whereas “add”, “please” and “improve” were signs of feature requests. In addition to bug reports and feature requests, ratings and user experiences were also categorized. [Maalej *et al.* 2016]

When using document classification in app store reviews, the reviews, including the title and the text, are the documents. The main difference between document classification and the string matching is that the first is dynamic and the latter static approach. The reason for that is that in document classification, the keywords are automatically identified whereas in string matching they are manually created. Bag of words (BOW) is a document classification technique which is commonly used. BOW creates a dictionary of all terms in the corpus of all reviews and calculates whether the term is present in the review of a certain type and how often. The types of reviews are identified based on the terms existence and frequency with the help of supervised machine learning algorithms. [Maalej and Nabil 2015]

Maalej and Nabil [2015] chose to exploit natural language processing, NLP, [Bird *et al.* 2009] in preprocessing the review text. Preprocessing is fundamental because stopword removal, stemming, lemmatization, tense detection, and bigrams can help to increase the classification accuracy. Removing stopwords increases the influence of informative terms like “bug” or “add”. [Maalej and Nabil 2015] However, some keywords Maalej and Nabil [2015] defined can be identified as stopwords [Bird *et al.* 2009] can be relevant for the review classification. For example, the terms “should” and “must” may indicate a feature request but they could be left out after removing stopwords because they are common English words which do not influence the semantic of the review. Bigrams [Bird *et al.* 2009] were also used by Maalej *et al.* [2016] to preprocess the review text. Bigrams are all combinations of two contiguous words in a sentence, for example, the sentence “The app crashes often.” has the bigrams “The, app”, “app, crashes” and “crashes, often”. Compared with using single terms, bigrams capture more context of the word in the review [Harman *et al.* 2012]. For example, single terms such as “crashes”, “never” and “always” might identify the review as a bug report, but bigram “never crashes” indicates to rating.

Maalej *et al.* [2016] collected metadata of the reviews, which included the star rating, the length of the review, and the submission time. Metadata is significant for classifying app reviews because it can include helpful information. For example, the reviews with

negative rating are likely to contain a bug report, and according to Pagano and Maalej [2013] user experience is likely to be found in positive reviews. Pagano and Maalej [2013] also state that the length of the review is usually connected to its meaningfulness, meaning that lengthy reviews tend to be more informative indicating a report on an experience or a bug. Maalej *et al.* [2016] stated that the tense of the verb can also be used as an indication of the review type. Past tense is more commonly used for reporting and could reveal a description of a feature than future tense which is more likely to be used for a promise or a hypothetical scenario and might reveal an enhancement or a feature request. Since one review can include several tenses, Maalej *et al.* [2016] calculated the ratio of each tense as metadata, using NLP technique. For more fine-grained sentimental analysis Maalej *et al.* [2016] chose to use SentiStrength [Thelwall *et al.* 2012], which assigns for each review one negative and one positive sentiment score. The scale of negative ratings is from -5 to -1 and for positive from 1 to 5.

Binary classifier decided whether the review is of a certain type or not. In this study, each review can be binary-classified four times: as a bug report or not, a feature request or not, user experience or not, and a rating or not. Each of these four classification models needs to be created and trained with true positives and true negatives separately. [Maalej *et al.* 2016] On the other hand, a review can also belong to several classes at once. To find all the classes of a single review there needs to be multiclass classification. Naïve Bayes [Bird *et al.* 2009] algorithm and Decision Tree learning [Torgo 2011] were used for binary classifiers and the multinomial logistic regression [Torgo 2011] for multiclass classification.

As research data, Maalej *et al.* [2016] collected ~1.1 million reviews from 1100 Apple store apps and 146057 reviews from 80 apps from Google store. Half of the apps were paid and half free. For the truth set creation, there were 4400 reviews, where half were from Apple App Store and the other half from Google App Store, and the all were peer, manual and content analyzed. There was no clear trend for NLP techniques, meaning that more language processing may not lead to better results. Removing stopwords and adding lemmatization separately caused an increase in the precision to bug reports, feature requests and user experience, but decreased the precision for ratings. However, combining stopword removal and lemmatization did not have a significant effect on precision and recall. Additionally, there was no significant difference between using one or two sentiment scores. [Maalej *et al.* 2016] The highest precision was achieved for predicting user experience and ratings (92%) and the highest recall and F-measure was for predicting user experience. For bug reports, Maalej *et al.* [2016] found that the highest precision, 89%, was achieved with the bag of words, rating, and one sentiment, and the highest recall, 98%, with using bigrams, rating, and one score sentiment. To achieve a balance for precision and recall, the best result was received when combining a bag of

words, lemmatization, bigram, rating, and tense. Feature requests were found best using the bag of words, rating, and one sentiment, resulting in the highest precision 89%, whereas the best F-measure was achieved with a bag of words, lemmatization, bigram, rating, and tense (85%). The best option for predicting user experiences was a combination of the bag of words with bigrams, lemmatization, the rating, and the tense. To predict ratings, the best precision (92%) was reached when using the bigram, rating and one sentiment score. In the comparison between binary and multiclass classification, it turned out that binary classifiers are more accurate for predicting the review types. [Maalej *et al.* 2016]

5. Comparison between the tools

The four app store review analyzing tools introduced have multiple differences but also some similarities. The main reason for such differences is that each of the tools implements their solutions for analyzing and managing app store reviews from a different perspective. The aim of the Mobile App Review Analyzer by Iacob and Harrison [2013] is to find only feature requests provided by app store reviewers, whereas the sentimental feature analyzer by Guzman and Maalej [2014] attempts to inform application provider about all sentiments which their users have regarding the features of the application. The sentiments may cover also feature requests, but will also include information about, for example, malfunctioning features, which would be most likely classified as a bug report instead of a feature request. On the other hand, the Framework for App Review Mining by Chen *et al.* [2014] approaches the problem of managing a large amount of user feedback in app stores by filtering the reviews based on their informativity. This approach can cover more than users' sentiments regarding features, because an informative review can include general information about user experience, for example. Finally, Maalej and Nabil [2015] and Maalej *et al.* [2016] did not focus on building one possible solution, but on comparing various tools and methods, which could be used in building such a tool. However, yet this research was chosen to be covered in this thesis mainly because of the wide and detailed range of possible solutions.

5.1. Data retrieving and defining metadata

Even though the approaches of the tools vary, there are still some similarities in their structures. The first step of each tool is to retrieve data and process it. However, the data used is not the same as any of the tools. The amount of reviews exploited in research varies from 2 000 to over a million reviews, and some of them are collected from both Apple App Store and Google Play Store. On the other hand, Chen *et al.* [2014] only use reviews written in Google Play Store.

In addition to different amounts and sources of data, some also exploit metadata. However, metadata used also varies: Iacob and Harrison [2013] include the date the review was written, star rating given, the device used for writing the review, version of the application and the title of the reviews as metadata, where Chen *et al.* [2014] use the posting time, rating and the title. Maalej *et al.* [2016] have similar metadata as Chen *et al.* [2014] but also they include the information of the length of the review and the tense it was written as metadata.

The reasons for why especially posting date, star rating and the tile were included in metadata in all those researches are logical. Firstly, the posting date may indicate to the version of the application, and therefore may inform if an issue presented in the review is relevant. Secondly, the star rating may have a significant role in identifying the type of the review, because less rated reviews are more likely bug reports whereas top rated concern commonly user experience. Lastly, the title may have important information regarding the review, since the main issue can be mentioned there, so it should be concerned as a key part of written review.

However, there are differences in using metadata, for example, Iacob and Harrison [2013] also collect the version of the application, but the other two do not. The reason why the latter ones do not collect the information may be because the posting time may be connected with the version, as described before. Additionally, Maalej *et al.* [2016] differ from Iacob and Harrison [2013] and Chen *et al.* [2014] by using the length of the review and its tense as metadata. Using the length as a part of metadata is reasoned with the fact that Pagano and Maalej [2013] state that the length of the review is usually connected to its meaningfulness, meaning that lengthy reviews tend to be more informative indicating a report on an experience or a bug. Moreover, using past tense is more common for reporting and could reveal a description of a feature than future tense which is more likely to be used for a promise or a hypothetical scenario and might reveal an enhancement or a feature request. On the other hand, Guzman and Maalej [2014] combine different tenses under one word, meaning that they do not use the information given by the tense of the sentence in their metadata, or in research. Moreover, Guzman and Maalej [2014] do not mention using any metadata at all.

5.2. Sentence splitting and language processing

Iacob and Harrison [2013], Guzman and Maalej [2014] and Chen *et al.* [2014] all split the content of the reviews into individual sentences using LangPipe [LingPipe 2019] and then analyzing feedback on sentence-level, instead of covering reviews as a whole. This may be useful since one review can contain many sentences, which can therefore, cover many issues. However, even one sentence can contain information about multiple things, such as “I really like the colors but the exit button does not work”, which has both information regarding user experience and a bug report.

The tools have different approaches when it comes to language processing. Chen *et al.* [2014], Guzman and Maalej [2014] and Maalej *et al.* [2016] all remove stopwords from the text. However, the set of stopwords removed varies based on the approach. Guzman and Maalej [2014] used stopwords provided by Lucene [2019] and added words which are common in user reviews but do not describe the application itself, such as "app" and "please". In their second approach, they added all the words assigned a sentiment into their stopword set to increase their accuracy. Maalej *et al.* [2016] exploited NLP [Bird *et al.* 2009] to define stopwords. However, they modified the set by removing some common English words such as "should" from the set, because they could imply that the review contains a feature request. It is a major difference that where Guzman and Maalej [2014] removed words such as "please" from their set of stopwords, Maalej *et al.* [2016] removed words indicating a request or such. For example, the word "please" is commonly used when someone asks or needs something, which may be a sign of a feature request. However, the difference can be explained with a different kind of approach to the problem.

On the contrary, Chen *et al.* [2014] do not define the set or a tool used for stopwords, but they are still used. Therefore, it can be assumed that they used a basic set of English words without any removal or addition. On the other hand, Iacob and Harrison [2013] do not use stopwords at all, but rather define keywords to identify a feature request. However, the scope of their study is significantly smaller than other tools', which may clarify the reason why stopwords were not used. Moreover, using only manually collected keywords was also one technique Maalej *et al.* [2016] used when comparing the methods.

In addition to utilizing single words, Maalej *et al.* [2016] and Guzman and Maalej [2014] used bigrams and collocations. Maalej *et al.* [2016] decided to use bigrams, because single terms such as "crashes", "never" and "always" might identify the review as a bug report, but bigram "never crashes" indicates to rating. Therefore, using a bigram instead of a word, the meaning behind the sentence can be clarified. As mentioned, Guzman and Maalej [2014] also chose to exploit using words as a pair in addition to singular words. A collocation is a collection of words that are often occurred together [Bird *et al.* 2009], and in the case of analyzing users' sentiments regarding the features of an application, some features were presented as a collocation, for example, <pdf viewer>. This clarifies identifying features, since the words are not considered as individuals, but as a collocation. Therefore, it can be concluded that using a pair of words can sometimes be a significant help to identify the meaning behind the text. There are multiple ways of doing so, and in these examples bigrams and collocations were used successfully. The situation should define which approach could be used, such as in the latter example, where collocations were useful to help to identify a certain feature.

5.3. Topic modelling

A common tool used in many of these researches was the topic modelling tool Latent Dirichlet Allocation, LDA [Blei *et al.* 2003]. Topic modelling can generally be an essential feature in managing and analyzing app store reviews, because categorizing issues by their topic makes it more efficient for the user to browse information. Since Iacob and Harrison [2013] focused on identifying feature requests, they used LDA for identifying common topics across the feature requests. On the other hand, Guzman and Maalej [2014] used LDA to get a matrix including the information of the number of times a feature was mentioned and was associated with a certain topic. Chen *et al.* [2014] compared LDA with another topic modelling tool, ASUM [Jo and Oh 2011]. The topics, where the sentences were categorized, had a wide range, such as “theme” and “emoji”. The various kinds of usage of LDA suggest that generally topic modelling is useful with these kinds of tools and can be exploited in different helpful ways.

5.4. Sentimental analysis

Since user feedback contains usually information which expresses user’s feelings towards a certain issue, sentimental analysis can be concerned as a useful method to identify those feelings. Both Guzman and Maalej [2014] and Maalej *et al.* [2016] exploited SentiStrength [Thelwall *et al.* 2010]. SentiStrength [Thelwall *et al.* 2010] divides the review text into sentences and then assigns a positive and negative value to each sentence. The output is a pair of numbers, informing the negativity and the positivity of a sentence. Because the output is a pair of numbers, they can be considered as individual scores, or as Guzman and Maalej [2014] did, as a single number presenting the score (the number was chosen to be the one, which had higher absolute value). Having sentimental scores may help to identify the type of the review, for example, it is more likely that a review with high negative sentimental score expresses malfunctioning and therefore could be a bug report.

5.5. Classifying

Classifying user feedback can be done in multiple ways, for example, Chen *et al.* [2014] classified non-informative reviews out of informative ones using Expectation Maximization for Naive Bayes (EMNB) [Nigam *et al.* 2000]. Classifying can be either binary classification, where the given input is classified either to something or not, or multiclass classification, where the classifier returns different probability values regarding its class. In the latter one, the given input can be assigned to multiple classes, where in binary classification it cannot. [Maalej *et al.* 2016] Where Chen *et al.* [2014] used binary classification for identifying informative reviews, Maalej *et al.* [2016] used it for finding the types of the reviews: is it a bug report or not, a feature request or not, a user experience or not, or a rating or not. Maalej *et al.* [2016] compared Naive Bayes

algorithm [Bird *et al.* 2009] and Decision Tree learning [Torgo 2010] with binary classification, and the multinomial logistic regression, MaxEnt [Torgo 2010] for multiclass classification. Both require creating different classification models and training them with true positives and true negatives.

5.6. Summary

Figure 17 concludes the common characteristics and differences mentioned in the previous subsections.

	Mobile App Review Analyzer	Sentimental feature analyzer	Framework for App Review Mining	Automatically classifying app reviews
Data	161 Google Play applications, 3279 reviews	7 Google Play applications, 7 App Store applications, 32210 reviews	4 Google Play applications, 2000 reviews	40 Google Play applications, 1100 App Store applications, over 1.2 million reviews
Metadata	Date, rating, device, version, title	-	Date, rating, the title	Date, rating, title, length, tense
Sentence splitting	LangPipe	LangPipe	LangPipe	-
Set of stopwords	-	Provided by Lucene, additions such as “app”, “please”	Not defined but used	Provided by NLP, removals such as “should”
Topic modelling	LDA to identify feature requests	LDA to create a matrix of feature requests	Comparing between LDA and ASUM	-
Sentimental analysis	-	SentiStrength	-	SentiStrength
Classifying	-	-	EMNB (binary)	Naive Bayes and Decision Tree (binary), MaxEnt (multiclass)

Figure 17. Matrix presenting the key similarities and differences between the tools.

The data used varied significantly, but implied that the scopes of the researches were also different. Most similarities were found in defining metadata, sentence splitting, topic modelling, and sentimental analysis. After data, the most significant difference between the researches was the use and definition of the set of stopwords. The four tools chosen for the review are not in commercial use, but yet they offer useful information, such as various approaches and the use of resources, for developers in app store review analyzing perspective.

All the tools covered in this research were proven to have positive impact on managing and analyzing app store reviews. Commercial tools, Appbot [2020] and AppFollow [2020] are already in use by multiple software companies, and the features they have provide both concluded and detailed information of users' opinions, making user data more accessible to use in development process.

Since the structures of Appbot [2020] and AppFollow [2020] are not publicly available, it is not possible to compare their approaches with the four other tools. However, sentimental analysis was utilized in the researches by Guzman and Maalej [2013] and Maalej *et al.* [2016], as it was in both commercial tools. Additionally, dividing the reviews by topic or keyword was also used in both commercial tools and in the four tools.

6. Conclusion

This research introduced general characteristics of feedback and its management in the software industry. It was proven that feedback can be pushed and explicit, pulled and explicit, pushed and implicit, or pulled and implicit [Maalej *et al.* 2015]. Additionally, the outputs of user feedback can be classified as either qualitative or quantitative information. Different feedback categories were presented and concluded as a conceptual model of feedback types, which can help product providers to make more strategic planning of feedback collection for product development and maintenance. More strategic planning with user data usage in development leads to better understanding of requirements [Kujala *et al.* 2005] and therefore, to building products with higher quality.

In addition to feedback types, some feedback channels were presented. General feedback form on a web page and Facebook page were covered more detailed and compared with other platforms. Following feedback channels, the characteristics of written feedback were presented. It was stated that the style of written feedback varies significantly, it does not follow any structured form [Stade *et al.* 2017] and may lack context, which would make it more useful [Maalej *et al.* 2015]. The answer to the research question "How software companies manage feedback?" was deliberated with introducing the current challenges, the lack of knowledge on how the feedback is generally collected, and the benefits and challenges when operating with feedback channels. It was suggested

that feedback analytic tools would help to manage user feedback, which could improve user involvement and therefore the quality of the product.

In Section 3, app stores were generally introduced. In addition, the characteristics, usefulness, and opportunities of app store reviews were covered. It was stated that app store reviews can serve as a major platform for feedback gathering, as the reviews can include useful information for developers and it is easier to collect compared with classic software engineering methods [Iacob *et al.* 2013]. Even though some of the reviews may have poor quality or they do not prove any value to developers [Pagano and Brügge 2013], they can still include important information which would make them valuable repositories of ideas coming directly from users [Iacob and Harrison 2013].

Two commercial app store review analyzers, Appbot [2020] and AppFollow [2020] were introduced by concluding the information received after free trials. Both tools were proven to be helpful to find useful information from the reviews. After the commercial tools, four tools chosen for the review were covered. First, Mobile App Review Analyzer by Iacob and Harrison [2013] focused on finding feature requests from app store reviews. Secondly, the sentimental feature analyzer by Guzman and Maalej [2014] provides information on the sentiments of application users towards the product. Thirdly, Framework for App Review Mining by Chen *et al.* [2014] categorized informative reviews from non informative ones. Lastly, the research by Maalej and Nabil [2015] and Maalej *et al.* [2016] tested various tools and methods which could be exploited in building a solution for app review analyzing.

Finally, the four approaches for managing app store reviews were compared with each other. The data used varied significantly, but implied that the scopes of the researches were also different. The question “What kind of tools can software companies use to manage app store feedback?” was answered with introductions of two commercial tools and more detailed information of the structures and approaches of four other tools. The ideal features in app store review analyzing were sentimental analysis and topic modelling. It was proven that since app store reviews offer a large and varied source of user data, managing the feedback would require a separate tool to analyze and categorize the reviews.

It was proven that app store reviews are useful in software development as a source of user feedback. Additionally, the issues with the quality and the amount of the reviews can be managed with appropriate tools. Having users involved in the development can, therefore, be more efficient which can lead to better products.

As this research is based on the literature and study of the available tools, the limitation of the work was the lack of both empirical evaluation in the real work settings and discussions with the practitioners. In the future, this work could be continued by

doing research based on those perspectives. However, there is yet no intention of continuing the work.

References

360 Mobile Assistant. 2020. *https://www.360download.org/download-360-mobile-assistant-for-android*. Visited on 26.3.2020.

Appbot. 2020. *https://appbot.co*. Visited on 12.2.2020.

AppBrain. 2020. *https://www.appbrain.com/info/advertise?utm_source=directory &utm_campaign=Businessofapps*. Visited on 26.3.2020.

AppFollow. 2020. *http://appfollow.io/*. Visited on 20.2.2020.

AppStore. 2020. *https://www.apple.com/ios/app-store/*. Visited on 25.3.2020.

Steven Bird, Ewan Klein and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc.

BlackBerry World. 2020. https://appworld.blackberry.com/webstore/?countrycode=US&lang=en. Visited on 25.3.2020.

D. M. Blei, A. Y. Ng, M. I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, Jan, 3, 993–1022.

Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In: *Proceedings of the 36th International Conference on Software Engineering*, 767–788.

CodeNgo. 2020. *http://www.codengo.com* . Visited on 26.3.2020.

Bruce Croft, Donald Metzler and Trevor Strohman. 2010. *Search Engines: Information Retrieval in Practice.* Addison-Wesley Reading.

Cydia App Store. 2020. *https://cydia-app.com* . Visited on 26.3.2020.

Laura V. Galvis Carreño and Kristina Winbladh. 2013. Analysis of user comments: an approach for software requirements evolution. In: *35th International Conference on Software Engineering (ICSE)*, 582–591.

Emitza Guzman, Walid Maalej. 2014. How do users like this feature? A fine grained sentiment analysis of app reviews. In: *IEEE 22nd International Requirements Engineering Conference*, 153–162.

Des Greer, Guenther Ruhe. 2003. Software release planning: an evolutionary and iterative approach. *Information and software technology*, 46, 4, 243–253.

Jonathan Grudin. 1991. Interactive systems: bridging the gaps between developers and users. *IEEE Computer*, 24, 4, 59–69.

Google Play. *https://play.google.com/store?hl=en* . Visited on 25.3.2020.

Stefan Gärtner, Kurt Schneider. 2012. A method for prioritizing end-user feedback for requirements engineering. In: *5th International Workshop on Co-operative and Human Aspects of Software Engineering*, 47–49.

GetJar. 2020. *https://getjar.com*. Visited on 26.3.2020

Mark Harman, Ye Jia and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In: *Proceeding of the 9th IEEE Working Conference on Mining Software Repositories*, 108–111.

Sami Hyrynsalmi, Tuomas Mäkilä, Antero Järvi, Arho Suominen, Marko Seppänen and Timo Knuutila. 2012. App store, marketplace, play! An analysis of multi-homing in mobile software ecosystems. In: *Proceedings of the International Workshop on Software Ecosystems,* 59–72.

Claudia Iacob, Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In: *10th Working Conference on Mining Software Repositories*, 41–44.

Claudia Iacob, Rachel Harrison and Shamal Faily. 2013. Online reviews as a first class artifacts in mobile app development. In: *International Conference on Mobile Computing, Applications, and Services*, 47–53.

Andrei Idu, Tommy van de Zande and Slinger Jansen. 2011. Multi-homing in the apple ecosystem: why and how developers target multiple apple app stores. In: *Proceedings of the International Conference on Management of Emergent Digital Ecosystems*, 122–128.

Slinger Jansen and Ewoud Bloemendal. 2013. Defining app stores: the role of the curated marketplaces in software ecosystems. In: *International Conference os Software Business*, 195–206.

Yohan Jo and Alice H. Oh. 2011. Aspect and sentiment unification model for online review analysis. In: *Proceedings of the fourth ACM International Conference on Web Search and Data Mining*, 815–824.

Onur Kucuktunc, Barla Cambazoglu, Ingmar Weber and Hakan Ferhatosmanoglu. 2012. A large-scale sentiment analysis for Yahoo! answers. In: *Proceeding of the fifth ACM International Conference on Web Search and Data Mining*, 633–642.

Sari Kujala, Marjo Kauppinen, Laura Lehtola and Tero Kojo. 2005. The role of user involvement in requirements quality and project success. In: *13th IEEE International Conference on Requirements Engineering*, 75–84.

Xiaozhou Li, Zheying Zhang and Timo Poranen. 2018. Scenario-driven continuous mobility requirements analysis in mobile app maintenance. *CEUR-WS.org*, online CEUR-WS.org/Vol-2075/CRE18_paper1.pdf.

LingPipe. 2019. *http://alias-i.com/lingpipe/index.html.* Visited on 13.12.2019.

Lucene. 2019. *https://lucene.apache.org*. Visited on 13.12.2019.

Walid Maalej, Hans-Jörg Happel and Asarnusch Rashid. 2009. When users become collaborators: towards continuous and context-aware user input. In: *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, 981–990.

Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In: *IEEE 23rd International Requirements Engineering Conference*, 116–125

Walid Maalej, Laeknaz Nayebi, Timo Johann and Guenther Ruhe. 2015. Towards data-driven requirements engineering. In: *19th International Working Conference on Requirements Engineering: Foundations for Software Quality, Doctoral Symposium Programme*, 223–230.

Walid Maalej, Zijad Kurtanović, Hadeer Nabil and Christoph Stanik. 2016. *On the automatic classification of app reviews*. Requirements Engineering, 21, 3, 311–331.

Christopher D. Manning and Hinirich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press.

George A. Miller. 1995. Wordnet: a lexical database for English. *Communications of the ACM*, 38, 11, 39–41.

Itzel Morales-Ramirez. 2013. On exploiting end-user feedback in requirements engineering. In: *19th International Working Conference on Requirements Engineering: Foundations for Software Quality, Doctoral Symposium Programme*, 223–230.

MyApp. 2020. *https://android.myapp.com*. Visited on 26.3.2020

Natural Language Toolkit. 2019. *https://www.nltk.org* . Visited on 13.12.2019.

Helena Holmström Olsson and Jan Bosch. 2014. From opinions to data-driven software R&D: a multi-case study on how to close the 'open loop' problem. In: *40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 9–16.

Helena Holmström Olsson and Jan Bosch. 2015. Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation. In: *International Conference of Software Business,* 154–166.

Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun and Tom Mitchell. 2000. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39, 2–3, 103–134.

Dennis Pagano and Bernd Brügge. 2013. User involvement in software evolution practice: a case study. In: *Proceedings of the 2013 International Conference on Software Engineering*, 953–962.

Dennis Pagano and Walid Maalej. 2013. User feedback in Appstore: An empirical study. In: *21st IEEE International Requirements Engineering Conference*, 125–134.

Norbert Seyff, Florian Graf and Neil Maiden. 2010. Using mobile re tools to give end-users their own voice. In: *18th IEEE International Requirements Engineering Conference*, 37–46.

Hannah Snyder. 2019. Literature review as a research methodology: An overview and guidelines. *Journal of Business Research*, 104, 333–339.

Ian Sommerville and Pete Sawyer. 1997. *Requirements Engineering: a Good Practice Guide.* John Wiley & Sons, Inc.

Melanie Stade, Farnaz Fotrousi, Norbert Seyff and Oliver Albrecht. 2017. Feedback gathering from an industrial point of view. In: *IEEE 25th International Requirements Engineering Conference*, 71–79.

Mike Thelwall, Kevan Buckley, Georgios Paltoglou. 2012. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63, 1, 163–173.

Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai and Arvid Kappas. 2010. Sentiment Strength Detection in Short Informal Text. *Journal of the American Society for Information Science and Technology,* 61, 12, 2544–2558.

Stefan Thomke and Ashok Nimgade. 1998. Note on lead user research. *Harvard Business Online*.

Luis Torgo. 2011. *Data Mining with R: Learning with Case Studies*. Chapman et Hall.

Joel West and Michael Mace. 2010. Browsing as the killer app: Explaining the rapid success of apple's iphone. *Telecommunications Policy,* 34, 5–6, 270–286.

Windows Phone Apps. 2020. *https://www.microsoft.com/en-us/store/apps/windows-phone.* Visited on 25.3.2020.

Xiaomi App Store. 2020. *https://www.mi.com/in/appdownload/*. Visited on 26.3.2020.