Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Identification of Criteria for Classifying the Explanation Needs in Apps

Masterarbeit

im Studiengang Informatik

von

Kliti Nikollau

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Klünder
Betreuer: Martin Obaidi

Hannover, 15.11.2024

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 15.11.2024

_____

Kliti Nikollau

Abstract

Software systems are an essential part of our everyday life. The implementation of artificial intelligence and machine learning algorithms has made apps increasingly more complex. This leads to users having to deal with unclear or unexpected outcomes when interacting with a software.

Explainability is an emerging quality aspect in software engineering. It can be defined as the ability of app to explain itself. Explainability needs can be often found in app reviews, where users frequently express the need for explanations about unexpected system behavior, unclear features, or specific interactions that do not align with their expectations. Addressing these needs can improve user experience and increase users trust in the software system. However, extracting and prioritizing explainability needs from large and diverse review pools remains challenging.

This thesis presents a structured approach for categorizing and prioritizing explainability needs in app reviews by developing a comprehensive set of criteria. This criteria list is designed to support better organization and prioritization of user feedback in app reviews. The criteria in this list come from app and review metadata, insights gathered through natural language processing methods from the reviews themselves, categorisation methods for the issue that are described in the review and other external factors. These criteria also went through stages of validation with explainability experts before reaching their final version.

To evaluate the relevance and practical application of these criteria, six requirement engineers were interviewed. They provided their perspectives on prioritizing criteria across different use cases. Furthermore, they also provided valuable insights for further expanding the criteria list and the use cases in which they can be used. Finally, they also confirmed, based on their experience, the potential that the implementation of this criteria list can have in a practical context.

Kurzfassung

Softwaresysteme sind ein wesentlicher Bestandteil unseres täglichen Lebens. Durch den Einsatz von Künstlicher Intelligenz und Algorithmen des maschinellen Lernens sind Apps immer komplexer geworden. Dies führt dazu, dass die Nutzer bei der Interaktion mit einer Software mit unklaren oder unerwarteten Ergebnissen konfrontiert werden.

Erklärbarkeit ist ein neuer Qualitätsaspekt in der Softwareentwicklung. Sie kann definiert werden als die Fähigkeit einer App, sich selbst zu erklären. Erklärungsbedarf findet sich häufig in App- Reviews, in denen Nutzer häufig Erklärungen zu unerwartetem Systemverhalten, unklaren Funktionen oder bestimmten Interaktionen, die nicht ihren Erwartungen entsprechen, fordern. Die Behebung dieses Bedarfs kann die Benutzererfahrung verbessern und das Vertrauen der Benutzer in das Softwaresystem stärken. Die Erkennung und Priorisierung von Erklärungsbedarf aus großen und vielfältigen Review-Pools bleibt jedoch eine Herausforderung.

In dieser Arbeit wird ein strukturierter Ansatz zur Kategorisierung und Priorisierung von Erklärungsbedarf in App-Reviews vorgestellt, indem eine umfassende Liste von Kriterien entwickelt wird. Diese Kriterienliste wurde entwickelt, um eine bessere Organisation und Priorisierung von Nutzerfeedback in App-Reviews zu ermöglichen. Die Kriterien in dieser Liste stammen aus App- und Bewertungs-Metadaten, aus Erkenntnissen, die durch Methoden der natürlichen Sprachverarbeitung aus den Bewertungen selbst gewonnen wurden, aus Kategorisierungsmethoden für das in der Bewertung beschriebene Problem und aus anderen externen Faktoren. Diese Kriterien wurden auch von Erklärungsexperten validiert, bevor sie ihre endgültige Form erreichten.

Um die Relevanz und praktische Anwendung dieser Kriterien zu bewerten, wurden sechs Anforderungsanalysten befragt. Sie gaben ihre Einschätzung zur Priorisierung der Kriterien für verschiedene Anwendungsfälle ab. Darüber hinaus lieferten sie wertvolle Erkenntnisse für die künftige Erweiterung der Kriterienliste. Schließlich bestätigten sie auf der Grundlage ihrer Erfahrungen auch das Potenzial, das die Umsetzung dieser Kriterienliste in der Praxis haben kann.

Contents

Chapter 1

Introduction

Software systems have become an integral part of our daily lives, influencing a wide range of decisions that affect us in various ways. Some examples of these decisions range from navigation apps, like Google Maps, that choose the fastest routes to help us avoid traffic, to financial and healthcare applications with even more profound implications for our well-being. However, due to the increasing complexity of software systems [20], it has become difficult to understand the rationale behind certain outputs produced by these systems [19].

This can lead to the usage of explanations in applications, which can give an insight into the ways a system works and are seen as an option to mitigate the lack of transparency in a software system [14]. They may be used to make the system more transparent and to allow the user to build up more trust in it [26]. These explanations can answer different questions such as "Why did the system produce this output?" or "Why is this output different than usual?". This type of explanation is often implemented in recommender systems [11].

Explainability is generally defined as the ability of a software to be explained [6]. It is an emerging quality aspect evoking new research in the field of requirements engineering [6]. Many see explainability as a suitable means to foster stakeholder trust [22] [5]. However, when implementing explanations in applications, it should be taken in consideration that explainability needs vary depending on the software type [15]. Research has also shown that explainability has a clear "Double-Edged-Sword" effect [6]: while it can help in adding transparency and facilitating the understanding of a system, it may also result in an opposite effect, if incorrect design choices are taken during requirement analysis [6].

Applications Stores, such as Apple's App Store[1] and Google's Play Store[2], offer the possibility for users to provide feedback on apps they have downloaded. Research by Pagano and Maalej [28] highlighted that app reviews contain valuable requirements engineering related information, as they represent rich and readily available textual data that provides insights into thousands of user experiences. This feedback is valued by developers, since it provides them with important information to improve software quality and identify possible missing features [27]. Research has also shown that the involvement of users and their continuous feedback are major success factors for software projects [16]. A significant amount of the reviews include requirements-related information such as bugs or issues [28], summary of the user experience with certain features [17], requests for enhancements [18], and even ideas for new features [3], [28]. Finally, research has also shown that, although relatively sparse, explanation needs in app reviews are also valuable and contain rich information [34]. This, combined with the fact that app stores can serve as a communication channel between users and developers [28], makes app reviews an ideal source to capture and study users explanation needs.

1.1 Problem

Previous research, including work by Droste et al. [15] and Unterbusch et al. [34], have already made efforts to categorise explainability by creating taxonomies that classify these needs into different categories. These taxonomies provide a structured overview of the types of explanation needs that users might have. However, both these taxonomies, do not dive very deeply into which is the specific aspect that is being referenced by the review or the context that causes the explainability need. As a result, these taxonomies capture only a single dimension of analysis, focusing primarily on the high-level classification of explainability needs without accounting for other critical factors that might affect the categorisation of a review.

When considering the use case of classifying and prioritising explainability needs, relying solely on a single categorical classification might not be enough. This is because explainability needs can vary significantly in complexity, urgency, and relevance, depending on multiple contextual factors. Beyond simply categorizing the need for an explanation, it is essential to assess additional factors that can provide a more broad view of the user's issue. For example, sentiment analysis can reveal the emotional tone of a review, offering clues about whether a user is frustrated, confused, or simply requesting clarification. Similarly, the frequency with which a particular issue is reported may indicate a more systemic problem that affects

[1] https://www.apple.com/de/app-store/
[2] https://play.google.com/store

a broader user base, thereby needing a higher prioritization for addressing that need. The app version used by the reviewer is also a critical factor, as certain issues may be specific to particular updates, devices, or operating systems, helping developers identify and address version-specific concerns efficiently. By considering all these diverse criteria, the explanation needs can be categorised more accurately and the response to each need can be prioritised and delegated more efficiently.

1.2 Solution approach

This thesis tries to tackle the aforementioned problems by defining a list of different criteria that can be related to explainability needs. This criteria will then be validated for different use cases through interviews with requirements engineering experts. Furthermore in this thesis, several new criteria will be created through a pre-study in which explainability needs from an app-review dataset will be labeled and categorised. The objective here is to categorise explainability needs as detailed and precise as possible. These new criteria will be added to other criteria that will be gathered from previous research, review metadata and other factors which will be discussed later in this thesis. Lastly, after these criteria will be validated, a subset of the original review dataset will be prepared and labeled with the criteria that were deemed most valuable in the interviews with the experts.

1.3 Objective of thesis

The objective of this thesis is to identify and validate criteria that can help in categorising and prioritise explanation needs in app reviews. To achieve this, the thesis will integrate existing criteria from prior research, app review metadata and a set of new categories which will be created by labeling an existing dataset of app reviews. In order to validate the found criteria, interviews with industry experts will take place. They will be asked to prioritise the found criteria for different use cases. These use cases include the manual and automatic categorisation and prioritisation of explanation needs and also using these criteria in eliciting requirements for explainability in apps. Furthermore, the experts will be asked to expand this list with other criteria if possible and also if they think that these criteria can be used in even more use cases than the ones already mentioned. Lastly, following research questions will be answered:

RQ1 What criteria can support the categorization and prioritization of explanation needs in app reviews?

RQ2 Which of these criteria are prioritized highest by requirements engineers?

1.4 Structure of thesis

This thesis is structured as follows: In Chapter 2, the fundamentals of app reviews and explainability will be discussed. In Chapter 3, works related to categorising explainability and app reviews will be considered. In Chapter 4, the steps taken in the development of the criteria list related to explainability needs in app reviews will be presented. The evaluation part of this thesis will be covered in Chapter 5 where the interview in which the criteria have been evaluated will be discussed and the results will be presented. Chapter 6 covers the discussion part of the thesis where the results will be interpreted and the research questions will be answered. The conclusion of this thesis will be discussed in Chapter 7.

Chapter 2

Foundations

2.1 Explainability and explanation needs

This section will provide an overview of key concepts related to explainability and explanation needs, establishing a clear understanding of these terms in the context of software systems. It will first define explainability, exploring its role as a quality attribute that enhances system transparency and user trust. This is followed by the definition of the concept of explanation needs, together with an example of categorisation of it.

2.1.1 Explainability

Shortly, explainability can be defined as the ability of a software system to be explained [6]. The ability to provide explanations, a natural ability of humans, is therefore considered an important capability of software systems. As such, explainability is now accepted as a critical quality attribute [6] and represents an emerging topic in the field of Requirements Engineering [2]. Considering this, Chazette et al. [5] define explainable systems more formally as follows:

> **Definition:** (Explainable systems) A system S is explainable with respect to an aspect X of S relative to an addressee A in context C if and only if there is an entity E (the explainer) who, by giving a corpus of information I (the explanation of X), enables A to understand X of S in C.

The following example can make this definition more practical: A smartphone app (**system S**) provides health-tracking features. The **aspect X** of **S** that needs to be explained is "how the app calculates daily calorie intake".

The **addressee A** would be the app user. The **context C** is the the user wanting to understand how these calculations are made in order to make informed dietary choices. Furthermore, the **entity E** (the explainer) is the app documentation or a customer support representative. The **corpus of information I** (explanation) are the details regarding the algorithm.

According to the definition, the app (**S**) is explainable with respect to calorie calculation (**X**) to the user (**A**) in the context of making dietary decisions (**C**) if and only if the documentation or support team (**E**) provides sufficient information (**I**) for the user to understand how calorie intake is calculated within this context.

Currently, research on explainability is often focusing on the field of AI [6], which why the term *Explainable Artificial Intelligence* (XAI) has gained prominence. XAI methods focus on explaining the underlying model, more precisely, the internal operations to justify decisions made [10]. These models are made explainable to enable the user to build more trust in the system [12]. For example, the explainability technique LIME tries to explain the predictions of any classifier in an interpretable and faithful manner [30].

As mentioned in Chapter 1, explainability has a "Double-Edged-Sword" effect [6]. While they may have a positive impact on some non-functional requirements that affect transparency, they must be carefully designed so that they do not have the opposite effect on software quality [6].

Kohl et al.[19] and Chazette et al. [6] have emphasized the significance of identifying users' specific needs for explanations and providing customized explanations correspondingly on their researches. Indeed, in cases where users do not require explanations, ensuring explainability may not be necessary [6], since implementing explanation can increase development costs [5].

2.1.2 Explanation needs

Unterbusch et al. [34] define an *Explanation Need* as a knowledge gap that a user intends to close through an app review. To consider a review as an *Explanation Need*, the user must explicitly raise a question or express a need for an explanation. Following the formatting of Chazette et al.'s definition of explainability [5], they formally define Explanation Needs as:

> **Definition:** (Explanation Needs) An addressee A has incomplete knowledge about an aspect X of system S in context C and requests a corpus of information I provided by an entity E that allows A to understand X of S in C.

Unterbusch et al. [34] provided an example of categorisation of explanation needs in app reviews. To do this, they created a taxonomy and divided explanation needs firstly into two main categories: *Primary Concerns* and *Secondary Concerns*. Each of these categories was than divided into subcategories. *Primary concerns* included the subcategories: *Training*, *Interaction* and *Business*. *Secondary concerns* included *Dissatisfaction* and *Errata*. The full representation of this taxonomy is displayed in Figure 2.2.

2.2 App reviews

This section will discuss app reviews and their importance in software engineering. Furthermore, a gold-standard dataset that has been used in this thesis which includes 4500 reviews labeled with their respective explanation needs, will be presented.

2.2.1 User feedback through app reviews

Application distribution platforms, also known as just "app stores", such as *Google Play* and*App Store* from Apple allow users to search and download their favorite apps in their devices. As of August 2024, Android users were able to choose between 2.3 million apps, making Google Play the app store with the biggest number of available apps [4]. The Apple App Store was the second-largest app store with roughly two million available apps for iOS. Whereas the exact number of apps may fluctuate as Apple and Google regularly remove low-quality content from their app stores, the number of apps has been increasing over the years [4]. Apps can be downloaded for free or they might require a payment before downloading.

These app stores also offer the users the possibility to give feedback for the apps they have downloaded. This can be done by writing a review or giving a star rating for the apps. This feedback is public and is also available to the developers. This kind of feedback is interesting from the software and requirements engineering perspective, because it allows for a user-driven quality assessment [28].

Unlike structured interviews or online surveys, where participants respond to specific prompts or questions, app reviews are spontaneously written by users in an open-ended format. This unstructured nature allows users to express their thoughts, opinions, and experiences without constraints, making their feedback more authentic and reflective of their genuine concerns. Since app reviews are written voluntarily, users are typically driven by personal motivations, such as sharing their experiences, highlighting problems they encountered, or suggesting ways to improve the app's features and functionality. Maalej and Nabil [24] classified app reviews into 4 main

categories in their research: bug reports, feature requests, user experiences and ratings.

2.2.2 Gold-Standard dataset

Kupczyk created in his master thesis [13] a dataset of reviews [25] that contain explainability needs. To do this, he gathered 90,000 reviews which served as filter basis. Afterwards, filters were used to filter the dataset for explicit and implicit explanation needs. Of the original 90,000 reviews, 1,500 reviews containing potential implicit and 1500 reviews with potential explicit explanatory needs were filtered out [13]. In addition, 1500 reviews for which none of the filters applied were initially considered as reviews with no need for explanation. Finally, the remaining 4500 reviews were manually labeled again to assure their correctness of the filters [13].

This dataset served as the foundation for evaluating explanation needs in apps throughout this thesis. It was utilized to label the identified explanation needs using newly defined categories, which then served to create new criteria with which explainability needs can be identified and prioritised.

2.3 Criteria and Taxonomy

This section will define what a criterion in the context of app reviews is and also present the importance of using taxonomies when categorising elements in a software engineering context.

2.3.1 Criteria

In the context of app reviews, a criterion in this thesis is defined as a specific element or attribute that can be extracted or inferred from the review that returns a piece of information about the review or the explanation need expressed in it. These criteria can include direct information, such as the app name, genre, or username, as well as meta-information, like the sentiment of the user writing the app review, affected app aspects, or the discussed system behavior. The latter type of criteria may be identified through human labeling or automatically via machine learning algorithms. The purpose of these criteria is to systematically categorize and analyze the content of the reviews in order to detect explainability needs.

Each of these criteria can have a different contribution when classifying and prioritising an app review. Some criteria can provide information regarding the app itself that the user has used, like the name or version of the app. Other criteria like the sentiment of the review can be used to prioritise a review. For example, if the sentiment of the review is very negative, than maybe this review should be prioritised higher than another in which the

sentiment is rather neutral, since the problem discussed here has deeply affected the user. Lastly, other criteria can offer an insight into which aspect of the app is being discussed in this review. These criteria can allow a better categorisation and prioritisation of the tasks that arise during the evaluation of app reviews and consequently a more efficient delegation of them.

2.3.2 Taxonomies

The Cambridge dictionary [1] defines taxonomy as "a system for naming and organizing things, especially plants and animals, into groups that share similar qualities". Nevertheless, taxonomies are also used in software engineering [35]. Their usage offers different advantages. Since a taxonomy is mainly defined as a classification method [35], it can also help in the following areas in which a classification is useful. For example:

- It can provide a better understanding of the relationships between the objects of a knowledge field [36]
- It can support the decision making process [36]
- It can help in identifying gaps in a knowledge field [36] .

Hierarchical taxonomies consist of one top class with several sub-classes which can also have further sub-classes. A true hierarchy ensures mutual exclusivity, which means that an entity belongs to exactly one class [15]. This type of taxonomy is the most common in the software engineering sector [35]. An example of a hierarchical taxonomy taken from Droste et al. [15] is shown in Figure 2.1. Another example is the taxonomy from Unterbusch et al. [34], mentioned also in Subsection 2.1.2, which is displayed in Figure 2.2. Taxonomies in software engineering are most frequently developed for the knowledge areas of software construction, software design, software requirements and software maintenance [35].

Studying app reviews for explanation need identification is a relatively under-researched area. Consequently, a taxonomy of Explanation Needs can aid in advancing knowledge and eliciting requirements for developing explainable systems [34]. Another benefits of a taxonomy, is that it enables researchers and engineers to extract explainability requirements in a systematic and rigorous manner [34].

A taxonomy that contains the different types of explanation needs can serve as a checklist, giving the requirements engineers guidance to discuss with the customer or users which explanations are desired [23]. In addition, a taxonomy offers a clearly defined terminology which helps to express explanation requirements in the further requirements engineering process

[1] https://dictionary.cambridge.org/

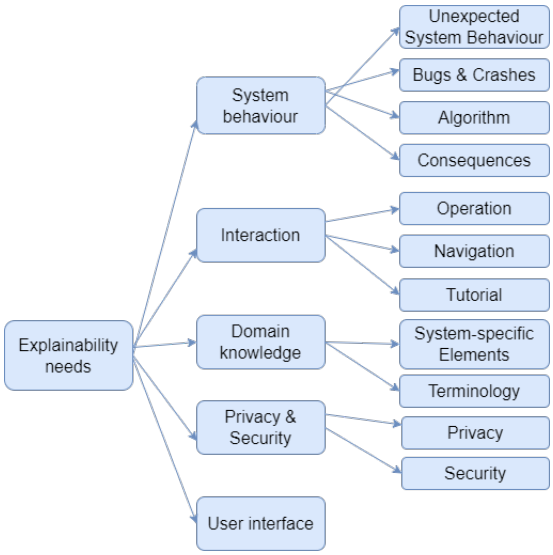

Figure 2.1: Taxonomy of explainability needs in everyday software systems [15]

[15]. In order to create such a taxonomy, however, it is necessary to find out which types of explanation requirements actually exist in different software systems [15].

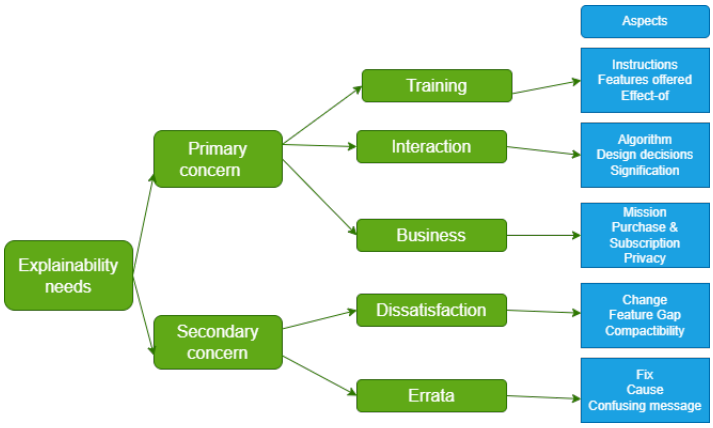

Figure 2.2: Taxonomy of explanation needs in app reviews [34]

2.4 Metrics

2.4.1 Interrater agreement

The Interrater Agreement is a measure of consistency among evaluators or raters who assess or categorize a set of items independently. It gauges the extent to which multiple raters provide the same ratings, thus assessing the reliability of subjective judgments across different evaluators. A high level of interrater agreement indicates that raters have similar perceptions or interpretations, enhancing the credibility of the assessment results.

2.4.2 Cohen's Kappa

Cohen's Kappa (κ) is a statistical measure used to evaluate interrater reliability between two raters who independently classify items into mutually exclusive categories [7]. It accounts for the degree of agreement beyond chance, making it more robust than simple percent agreement. A Kappa value of 1 indicates perfect agreement, 0 indicates no agreement beyond what would be expected by chance, and negative values suggest agreement less than chance [7]. Cohen's Kappa is widely used in research contexts where consistent and reliable subjective judgments are required, especially in categorical data analysis [7]. Cohen's Kappa can be calculated using the following formula:

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

where

- P_o is the proportion of units the raters agreed with one another
- P_e is the proportion of units for which the agreement is expected to be by chance.

Kappa values are often interpreted with the scale by Landis and Koch, which is shown in Table 2.1 [21]. Although these divisions are arbitrary, they do provide useful "benchmarks" for discussion [21].

Kappa Statistic	Strength of agreement
< 0.00	Poor
0.00-0.20	Slight
0.21-0.40	Fair
0.41-0.60	Moderate
0.61-0.80	Substantial
0.81-1.00	Almost perfect

Table 2.1: Strength of agreement for Kappa Statistic

2.5 Terminology

In this section, some often used terms during this thesis will be explained in order to avoid later confusion between them.

Criterion

A single element of the final criteria list that is developed through this master thesis.

Bottom-Up Approach

This is an approach that is used throughout this thesis to create new criteria to be added to the final list. Through this approach, explainability needs are labeled and than these labels are grouped together into categories. These categories are further grouped together if they have similarities with one another until no more grouping is possible. These categories are refined through different validation steps.

Code system

This term is used to refer to the new criteria that are created in this thesis. These newly created criteria will than be added to the rest of the gathered criteria in the final list.

Chapter 3

Related works

3.1 Explainability

Complex software systems, especially those that use artificial intelligence, can be used to make important decisions in our lives, like for example in medicine application. This is why explainability in a major topic in the field of AI in medicine [8]. Ribeiro et al. try to tackle this problem by introducing a method called LIME (Local Interpretable Model-Agnostic Explanations) for explaining the predictions of machine learning models [30].

LIME is a novel method designed to explain the predictions of any classifier by learning an interpretable model locally around each prediction. It can be applied to both text and image classification models (by using random forests, neural networks etc.) and provides explanations that are understandable and faithful to the model's behavior [30]. LIME is designed to be model-agnostic, meaning it can work with any machine learning model. This flexibility allows it to be used in a variety of applications where models are often viewed as "black boxes". It provides qualitative insights by identifying the features that contribute most to a specific prediction [30].

While the research from Ribeiro et al. [30] focuses mostly on AI models and providing explanations for their outputs [30], this thesis focuses on gathering criteria that can categorise explanation needs. Both these works though, have as end-goal building user trust in complex systems through explainability.

Deters et al. introduce a technique to elicit explainability requirements [11]. This approach allows users to request explanations as they interact with a system, putting them in a realistic usage context. This also helps in determining which explanations are truly needed, addressing the challenge of eliciting meaningful explainability requirements during the early stages of

software development.

To achieve this, a user study with 21 participants was conducted using a high-fidelity prototype to evaluate the effectiveness of the technique. The study examined how well users could categorize their needs into the three predefined categories and provided insights into which types of explanations were requested at different stages of interaction [11]. The proposed technique also addresses the "Why-Not" effect, where users tend to request explanations just because they are available. By offering explanations on demand and requiring users to actively seek them, the technique ensures that only genuinely needed explanations are requested, resulting in a more accurate elicitation of explainability requirements [11].

The goal of the paper from Deters et al.[11] is to elicit requirements for explainability during the requirements engineering phase, while this thesis tries to develop a set of criteria to categorise explainability needs for an already existing application. Both works are concerned with understanding where and why users need explanations. Furthermore, both studies emphasize the importance of involving users in the process, either through feedback analysis, which is the approach of this thesis, or direct interaction and feedback elicitation, which is the approach used in the paper from Deters et al. [11]).

3.2 Categorising explainability

Droste et al. [15] developed a taxonomy of explainability needs based on an online survey with 84 participants. The researchers asked the participants to state their questions and confusions concerning their three most recently used software systems and elicited both explicit and implicit explainability needs from their statements. In total, 315 explainability needs were identified and classified from the survey answers. They classified users' needs across five main categories and 11 subcategories. The five primary categories include: *System Behavior, Interaction, Domain Knowledge, Privacy & Security* and *User Interface*. The full categorisation is shown in Figure 2.1

The study shows that the need for explanations varies based on the type of software. For instance: Interaction explanations were most prevalent in complex systems like productivity, creativity and design tools. System behavior explanations were prevalent across all software types. Privacy and security explanations were particularly needed in software systems and in software for communication and information [15].

This thesis, similar to the research from Droste et al. [15], focuses on categorising explainability needs. The difference is that in this thesis, the objective is to create a list of different criteria that can help in this task,

while the aforementioned paper categorises explainability needs based on only one criterion. Furthermore, the researchers above gather explainability needs through an online survey, while in this thesis, explainability needs are gathered from a dataset of app reviews.

Unterbusch et al. [34] explore in their paper explanation needs in app reviews. They manually coded a set of 1,730 app reviews from 8 apps and also derived a taxonomy of Explanation Needs. They were firstly divided into two main categories: Primary Concerns and Secondary Concerns. Each of these categories was than divided into subcategories. Primary concerns included the subcategories: Training, Interaction and Business. Secondary concerns included Dissatisfaction and Errata. The taxonomy can help developers understand and address user needs for explanations in a more systematic way. This paper also demonstrated that app reviews can be useful when searching for explanation needs [34].

The paper evaluates multiple machine learning (ML) and deep learning (DL) approaches for automatically detecting explanation needs in app reviews [34]. The best-performing model is a fine-tuned BERT (Bidirectional Encoder Representations from Transformers) model, which achieved a weighted F-score of 86% on unseen app reviews.

Similar to this thesis, the research from Unterbusch et al. [34] explores the needs for explanation in app reviews. Similar to the research from Droste et al. [15] though, the explanation needs here are also categorised using only one criterion. This thesis on the other hand, aims to find a list of criteria than can help in identifying, categorising and prioritising explanation needs, since using only one criterion may not be enough.

Tsakalakis et al. [33] provide a different approach from the previous research when using classifications in a explainability context. Instead of classifying explanation needs, they classify the explanations that are provided in an application. In the paper they introduce a proactive, explainability-by-design approach. This approach integrates explanations into systems from the outset, rather than as an afterthought.

The authors develop a nine-dimensional taxonomy to classify explanations based on different criteria: *Source, Perspective, Autonomy, Trigger, Content, Scope, Explainability Goal, Intended Recipient* and *Priority* [33]. The taxonomy can be translated into a machine-readable format, allowing automated systems to generate compliant and meaningful explanations. The paper applies the taxonomy to real-world use cases (e.g., loan applications) and demonstrates how explanations can be generated systematically to meet legal and governance standards. This taxonomy and the explainability-by-design approach can help organizations create more transparent, trustworthy, and legally compliant automated systems.

This paper [33] differentiates itself from this thesis and the other previous papers mainly in it purpose. Its main objective is classifying the created explanations instead of the explanation needs of the user. Nevertheless, this paper provides insights on the importance of providing the correct explanations in different real-world use cases.

3.3 App-Reviews

Research from Pagano and Maalej [28] provides one of the earliest comprehensive empirical studies analyzing over 1 million reviews from the Apple AppStore. It investigates how and when users provide feedback, the content of the feedback, and its impact on app popularity and user communities. The authors identified and categorized user feedback into 17 different topics (e.g., praise, feature requests, bug reports, and shortcomings) and further grouped them into four themes: Rating, User Experience, Requirements, and Community. This categorization helps better understand the types of information users share in app reviews.

The authors explore the relationships between different feedback types and app ratings. They found that some feedback topics (e.g., feature requests, bug reports) correlate with lower ratings, while positive themes (e.g., praise and helpfulness) correlate with higher ratings [28]. Positive feedback, such as praise and recommendations, tended to improve app visibility and increase download numbers, while negative feedback often had the opposite effect. This suggests that user feedback not only impacts app quality perception but also its market success [28].

The paper provides insights on how app developers can leverage user reviews to extract requirements, identify improvement areas, and prioritize development tasks. It suggests using user feedback as an alternative to traditional requirements elicitation methods like interviews and surveys [28].

The referenced paper focuses on categorizing app reviews into distinct review types (e.g., feature requests, bug reports, user experiences) using predefined topics and themes to support software maintenance and evolution tasks. Their approach classifies the overall nature of the review, helping to understand broad categories of user feedback [28]. This thesis on the other hand, focuses on categorising explainability needs in app reviews, which is not a focus of the paper mentioned above. Nevertheless, the paper from Pagano and Maalej [28] serves as a basis when leveraging user feedback to extract useful information, such as in this thesis case, for further app improvement or explainability needs.

Maalej and Nabil [24] introduces a systematic approach to classify app reviews into four primary categories: bug reports, feature requests, user

experiences, and ratings. This kind of classification aims to filter and organize the large volume of user feedback in app stores that can be of interest for certain stakeholders such as developers, analysts, and other users.

The authors propose several probabilistic techniques, including text classification, natural language processing, and sentiment analysis, to predict the review type. They utilize metadata such as star ratings, text length, and sentiment scores, combined with different machine learning models like Naive Bayes, Decision Trees, and Logistic Regression [24].

The paper conducts an extensive evaluation to compare the performance of different classification approaches. They highlight that metadata alone is insufficient for accurate classification, while combining metadata with text-based classifiers yields much higher precision (70-95%) and recall (80-90%) [24].

Their findings can provide insights into building review analytics tools that can assist app developers and vendors in processing large amounts of user feedback more effectively. Their findings also include suggestions on how to combine text and metadata features to improve classification accuracy [24].

While the referenced paper [24] focuses on classifying app reviews into predefined categories such as bug reports, feature requests, user experiences, and ratings, this thesis takes a different approach by developing a set of criteria specifically tailored to categorize explainability needs within these reviews, which is not in focus of the aforementioned paper. Instead of categorizing the type of the review, this thesis focus is the understanding of what aspects of the app users seek clarification or explanations for. Nevertheless, in both works, app reviews offer the main exploration point.

Panichella et al. [29] introduced a taxonomy to systematically categorize app reviews into four key categories: Information Giving, Information Seeking, Feature Request, and Problem Discovery. These categories aim to highlight reviews that contain valuable information for software maintenance and evolution tasks. To build their taxonomy, the authors analyzed developer communication channels (such as mailing lists) to identify sentence-level categories relevant for software maintenance. This grounded approach helps ensure that the taxonomy aligns with the types of information developers actually need.

The paper proposed an approach that integrates Natural Language Processing (NLP), Text Analysis (TA), and Sentiment Analysis (SA) to classify app reviews according to the defined taxonomy. This multi-dimensional analysis allows the system to capture more nuanced information, including user intentions and emotions, that can be useful for understanding the context of each review [29]. The authors conduct a comprehensive empirical study to evaluate the performance of different machine learning classifiers

using combinations of NLP, TA, and SA features. They demonstrate that combining all three techniques outperforms using each individually, achieving higher precision and recall [29].

This paper demonstrates the value of leveraging multiple feature types to capture different aspects of user reviews [29]. This is a similar approach to the one used in this thesis, in which several criteria that can be extracted from app reviews are to be considered, which might help in a categorising and prioritising process. Furthermore, both works argue that using NLP, TA and SA methods can be useful when trying to classify app reviews. Although both works use app reviews to explore their objectives, the focuses of this thesis and the paper are different, since this thesis focuses entirely on explainability needs in app reviews, which is not a focus in the paper discussed.

Chapter 4

Study design

4.1 Research Goal and Research Questions

This section describes the research goal and the objective of the research questions. The following goal has been formulated using the Goal-Definition Template: [38]:

Research Goal: *Create* a list of criteria *for the purpose of* categorising and prioritising explainability needs *with respect to* improving the precision and efficiency of solving the explainability needs from app users *from the point of view of* software engineers and customer support teams *in the context of* app reviews.

Following research questions are addressed:

- **RQ1:** What criteria can support the categorization and prioritization of explanation needs in app reviews?
- **RQ2:** Which of these criteria are prioritized highest by requirements engineers?

Research question **RQ1** addresses the core objective of this study. By creating a comprehensive list of criteria, this research aims to equip app developers, project managers, and other stakeholders with a structured approach to categorizing and prioritizing explainability needs. Such a list will allow these professionals to better organize feedback from users. This categorization can also support developers in identifying recurring issues that users face, enhancing overall user satisfaction and fostering trust in the app. Moreover, it provides a foundation for systematic improvements in explainability, helping teams to prioritize actions that align with user expectations and demands.

Research question **RQ2** seeks to validate the relevance and applicability of this criteria list by involving requirements engineers in the evaluation process. Through interviews, these experienced professionals will assess the criteria based on different use cases and prioritise the criteria, based on their perceived importance for each use case. This will provide with a list of prioritised criteria that can be used as a foundation in the future when trying to implement the criteria in the respective use cases.

4.2 Approach

Several steps were taken in order to develop the final list of criteria for identifying explainability needs. The process began by reviewing related literature on explainability. Focus here was on how explainability has been categorized in previous research. From here, various categorization approaches were identified that were deemed potentially useful for this thesis. However, this initial review also highlighted gaps, revealing opportunities to introduce additional categories that were not fully addressed in prior work.

To do this, the dataset from Kupczyk [13] [25] was used. This dataset of app reviews was re-labeled with a focus on the specific explainability need expressed by the user. This re-labeling aimed to capture the explainability needs with greater precision, ensuring that the labels reflected the actual needs of users rather than general classifications. These labels were then organized into initial categories, which served as a foundation for further development.

This categorisation went into different iterations of evaluation and validation. This was done following a *Bottom-Up* approach, in which low level categories are continuously grouped into higher categories until until no further consolidation was necessary. Each step involved evaluating the clarity, granularity, and relevance of the categories to ensure that they accurately captured the range of explainability needs expressed in the dataset. This process also involved an evaluation from Explainability experts in the Software Engineering Department of the Leibniz University Hanover. Their feedback was incorporated and helped in shaping the final version of the created categories. This process will be covered more in depth in Section 4.6.

The created criteria were then joined to other criteria that were gathered from different sources. These included app-metadata, related works, external factors and textual information in the review that could be gathered through natural language processing. These could provide helpful insights regarding the wider context regarding the user and the review.

After the list of criteria was finalized, the next phase focused on preparing the

interviews, which would serve as a key component in validating the criteria. In this phase, the participants for the study were contacted and appointments were set accordingly. These interviews would serve as a final validation for the list of criteria. In these interviews, the participants would evaluate and prioritise the criteria for each of the use cases. By gathering expert feedback, the goal was to ensure the practical relevance and applicability of the criteria in real-world scenarios.

In addition to validating the existing criteria, the interviews aimed to uncover new criteria and use cases where the proposed approach could be used. Finally, the participants were also asked to evaluate the study itself, providing an opportunity to explore whether the proposed approach could be integrated into day-to-day work environments. The interviews, therefore, not only validated the criteria but also assessed the overall utility and potential impact of this approach in the industry. More detailed information regarding the interview process are discussed in Section 4.7. All the steps discussed above are summarised in Figure 4.1.

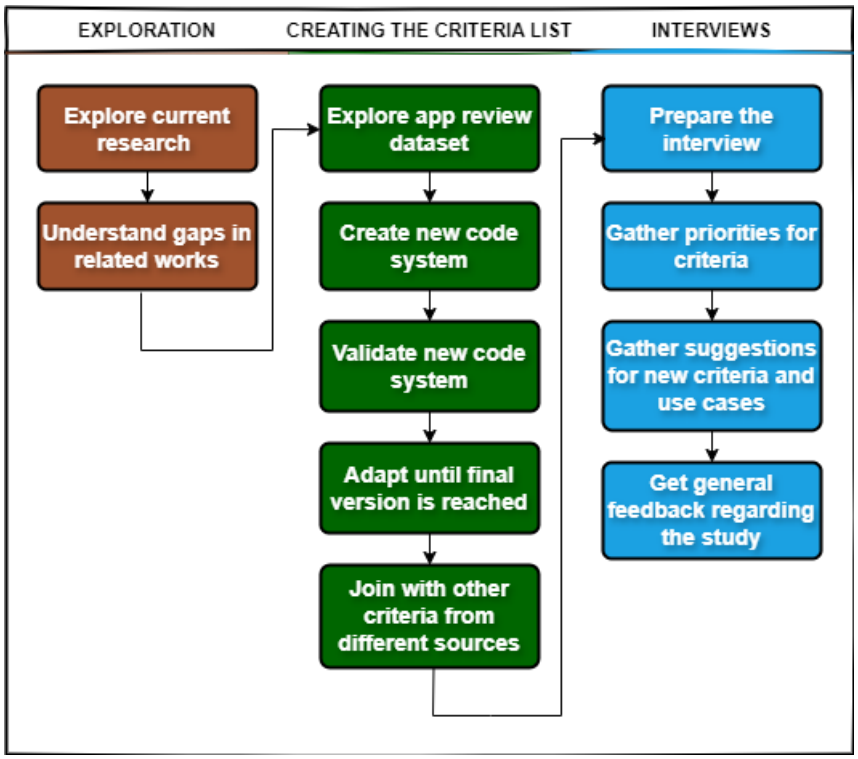

Figure 4.1: Study design

4.3 Stakeholders

In the context of categorizing explainability needs in app reviews, several stakeholders would benefit from this research. Understanding who these stakeholders are and why these criteria are relevant to them can help in shaping the design and implementation of tools or processes based on these criteria, when later used.

1. App Developers and Project Managers:

App developers and project managers are primary stakeholders, as they are responsible for the design, development, and ongoing improvements of the application. Explainability-related feedback in app reviews often contains critical insights into where users struggle to understand the app's behavior, functionalities, or decision-making processes. These criteria can help them prioritize areas requiring better transparency or improvements in the user interface. They can also pinpoint which aspects of the app are mostly seen as unclear for the users. For developers, this feedback can guide technical adjustments, while product managers can use it to shape user communication strategies and make informed decisions about feature roadmaps.

2. UX/UI Designers

User experience (UX) and user interface (UI) designers play a crucial role in making applications more intuitive and user-friendly. Explainability needs identified in user reviews often highlight areas where design choices may not have been fully understood or welcomed by the users. By using these criteria, UX/UI designers can systematically identify problematic areas and use them to design interfaces that present information more clearly, thus reducing user frustration and increasing satisfaction. This can essential for building trust and usability in complex systems, especially in data-driven apps or those leveraging machine learning models.

3. Customer Support

Customer support are often at the front line of managing user queries and addressing their frustrations. Understanding explainability needs can help these stakeholders in two ways: (1) by anticipating common questions or points of confusion and preemptively addressing them through knowledge bases or FAQs, and (2) by providing clearer and more user-centered explanations when responding to complaints or queries. Having a systematic way to categorize and prioritize these needs also enables them to escalate major usability issues to development teams more effectively.

4. Data Scientists and AI/ML Engineers

In applications that rely on data-driven decision-making, data scientists and AI/ML engineers are key stakeholders. Users of such applications

may leave feedback expressing confusion or mistrust around the model's outputs or decision-making process. By leveraging the criteria gathered in this thesis for categorizing explainability needs, these professionals can identify specific requirements for transparency and interpretability. This can drive the selection or refinement of models and algorithms to incorporate features like justifications, explanations, or visualizations that improve user understanding.

5. Regulatory and Compliance Officers

For applications operating in regulated sectors (e.g., finance, healthcare), ensuring transparency and compliance with ethical guidelines is very important. Regulatory officers need to ensure that the application provides adequate information to users, especially when automated decisions are involved. By identifying and categorizing explainability needs, these stakeholders can assess whether the app meets necessary standards for user rights and data transparency. This, in turn, supports compliance with regulations such as GDPR (General Data Protection Regulation) [37] or the EU AI Act [9].

6. End Users

Finally, end users are the ultimate beneficiaries of improving the explainability of applications. When explainability needs are identified and addressed, users gain a clearer understanding of the app's behavior, which in turn can improve their satisfaction and trust in the application. By incorporating user-centric feedback into product development, developers and designers can create more transparent and reliable applications, fostering a stronger user-app relationship and reducing frustration or distrust.

4.4 Use cases

The list of criteria for identifying, categorizing, and prioritizing explainability needs in app reviews has three main use cases: elicitation of explainability requirements, manual categorization of the explainability needs, and the automatic categorization of these needs. Each of these use cases addresses a different stage of working with explainability-related feedback in app reviews, thereby supporting diverse stakeholders in systematically enhancing app transparency and user satisfaction.

1. Elicitation of Explainability Requirements

Eliciting explainability requirements from app reviews is a crucial use case, as it involves extracting requirements from different stakeholders in the initial phases of the development of an application. Since explainability is a relatively new field of study, it can be hard to gather precise requirements

from these stakeholders in this context. The stakeholders might have an "Why-Not" approach, where they could tend to request explanations just because they "can not hurt". This of course, would bring with it higher development costs and also a higher cognition effort for the users [6]. The criteria gathered in this thesis can offer a more precise frame in dealing with this problem. They could serve as basis for discussion by confining the aspects in which explanation requirements can be requested. They can also serve as an example from which the stakeholders can take inspiration when thinking about possible requirements.

2. Manual Categorization of Explainability Needs in App Reviews

The manual categorization of explainability needs involves developers or requirements engineers reviewing app reviews to systematically classify feedback based on the developed criteria. This process is particularly useful for understanding the diversity and distribution of explainability-related concerns among users. By manually categorizing feedback, researchers can identify common patterns, categorize reviews into predefined explainability types (e.g., confusion about app navigation, difficult to understand UI etc.), and evaluate their relative importance. This manual approach is valuable during the early stages of developing a new feature or revamping an existing one, as it provides a nuanced understanding of user sentiment and context. Additionally, manual categorization helps validate and refine the criteria, ensuring that they are robust and applicable to a wide range of scenarios. This deep qualitative analysis is essential for identifying subtle yet impactful explainability needs that might be overlooked by automated tools, thus complementing the automated approach.

3. Automatic Categorization of Explainability Needs in App Reviews

Automating the categorization of explainability needs is the third use case, which aims to scale the analysis of large volumes of app reviews. With the increasing volume of user feedback, manual analysis alone can often be impractical for identifying explainability needs in a timely manner. By leveraging natural language processing methods and machine learning models trained on the developed criteria, automatic categorization can rapidly scan, categorize, and prioritize reviews based on some pre-defined context-specific rules. This automation is particularly beneficial for continuous monitoring, allowing developers and product managers to receive real-time insights into emerging explainability issues. Automated categorization also helps in highlighting patterns across different app versions or user demographics, making it easier to track the impact of feature updates on user understanding. While automation lacks the nuance of manual categorization, it provides a scalable solution that enables stakeholders to focus on the most critical issues and act quickly to address them. By integrating the

criteria into automatic categorization models, organizations can streamline the process of identifying and responding to explainability needs, thus being more efficient when dealing with them.

4.5 Requirements

The main objective of this thesis is to create a list of criteria that can be used to in the automatic identification, categorisation and prioritisation of explainability needs in app reviews. With this goal in mind, several key requirements emerge.

First, the criteria must be designed for implementation in an automated context. This means they need to be suitable for algorithmic integration and recognizable by a computer. The criteria should be clear, measurable and possible to extract with computational methods so that they can effectively be used in machine learning models and other automated tools.

Second, the prioritization process during expert interviews should follow a "knapsack" approach, where participants are encouraged to prioritize criteria that maximize benefits while minimizing costs. Each additional criterion incorporated into a machine learning algorithm increases development and computational costs, so participants must consider this trade-off when selecting the most important criteria.

Furthermore, the criteria in the list should be as distinct from one another as possible. This means that criteria should have as few overlaps with one another as possible, in order to avoid redundant information. Minimizing overlap between criteria is essential to avoid redundancy and ensure that each of them provides unique value in identifying and addressing explainability needs. The criteria uniqueness is covered more in detail in Chapter 5.

4.6 Creating the new criteria

In this section, all the steps taken to create the new code system will be presented. These steps are also summarised in Figure 4.2.

4.6.1 Creating the first labels

In order to fill in the gaps of previous studies when categorising explainability needs, the dataset from Kupczyk [13] was re-labeled. For the evaluation of the dataset and the re-labeling process, the software MAXQDA [1] was used. These labels that were created during this step were mostly of two forms.

[1] https://www.maxqda.com

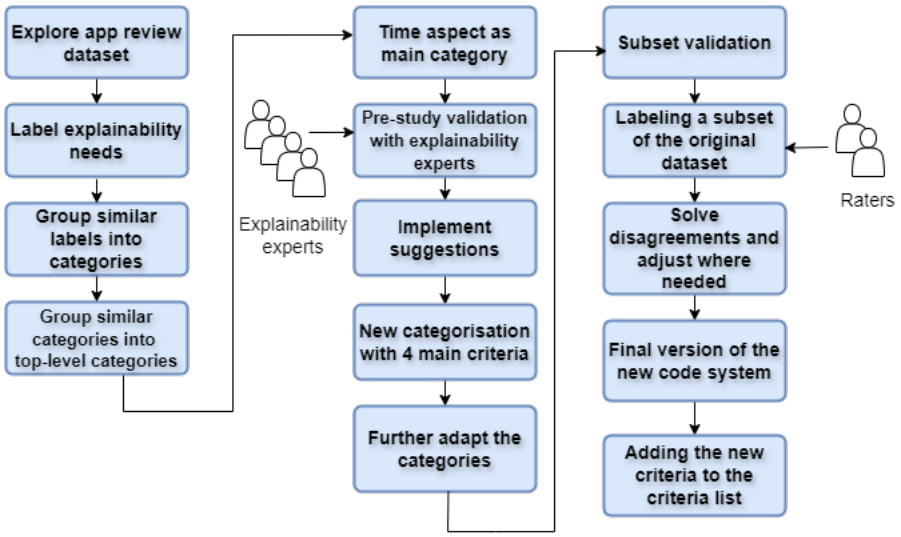

Figure 4.2: The process of creating the new code system

The first type involved summarizing the sentences where the explainability need was expressed in just a few words. For example, if a user wrote:

> "I do not understand why would you move the search bar from the top to the bottom. It was way better as it was."

than the label would be "search bar moved". This method worked well for needs that were straightforward to summarize and often allowed similar explainability needs to be grouped under a single label. Similar to the example above, there were over 40 other reviews in the dataset that addressed the same concern and they were all categorised under the same label. This demonstrated that common explainability needs could easily be grouped when they addressed the same issue.

The second labeling method involved using the entire sentence expressing the explainability need as the label itself. This method was mostly used for cases where the needs were too specific to be summarized and could not be combined with other similar reviews.

After all reviews with explainability needs in the dataset were labeled, the next step was to group these labels into categories. This resulted in the creation of the first coding system which comprised 15 top-level categories. Most of the top level categories were separated into sub-categories. In total, 38 categories were created in the first iteration. An overview, together with examples for each category, is shown in Table 4.1. This first system served as the foundation for the next iterations.

However, in this first system were noticed some problems. These were mostly

Top-level category	Bottom-level category	Example
Account related	Account	cant download music on basic account
	Login	cant login
Bug related	Bugs	app freezes
	Unexpected behaviour	lost progress
Connectivity and device support		sync app data between devices
Content related	General content of app	feed shows suggested posts more than chronological order posts
	Ads	Ads show up too often
Costumer support		How to Contact support
Features	Feature request	feature request- return of story pins
	Feature changed	feature changed- need to download image to zoom
	Feature removed	removed feature- swipe to dismiss
	Missing features	no dark mode
General app idea		one mistake takes one heart
Hardware		high ram use
Money related	Financial concept	why high prices
	Money	app doesnt return money
	Cards	cant verify card
	Payments	charged for something i didnt book
	Refund related	need refund
Orders		cant change order
Security	General security concerns	too much tracking
	Permissions	why need for birthday information
	Verifications	why does app needs many verifications
UI-UX related	Unhappiness of user	unhappy with app organisation
	UI	change font
	UX	app takes long time to load
Unclarity, tutorial needed	General question	What is COP?
	Change the state of something	change profile picture
Update related codes		search bar moved
Unclear-review-no need		

Table 4.1: First iteration of new criteria

Interaction type	**Aspect**	**Subcategory**
Direct interaction	UI/UX	
	Features	Notifications
	Content	Ads, Feed content selection
	Account	Login
	Bug/Unexpected behaviour	
Indirect interaction	General app idea	
	Costumer support	
	Business	Orders, Financial concept
	Security	Permissions, Verifications
	Definitions	General questions

Table 4.2: Aspects and subcategories of *No time mentioned*

related to the fact that the distinctions between the categories were not always clearly separated from one another. This would mean that some explanation needs could possible be grouped in more than one category. This would violate the mutual exclusivity principle of a hierarchical coding system. Most of the problems could be caused by the *Update related codes* category. Codes that were grouped into these category often overlapped with other categories. To tackle this problem, some changes were taken to the created code system.

The *Time aspect* was moved as the primary categorisation aspect. This included three categories: *Updates*, *Future* and *No time mentioned*. In the *Future* category, two sub-categories were included: *Feature requests* and *Plans*. The Update category included the *Feature missing*, *Feature changed*, *Bugs* and *Unexpected Behaviours*. For reviews not tied to a specific time frame, two major categories were created the *Direct* and *Indirect Interactions*. Both of them were further divided into sub-categories, which mostly consisted of the remaining categories from the first iteration. A graphical depiction of these subcategories is included in Table 4.2. The results from this iteration were used as basis of discussion for the pre-study evaluation.

4.6.2 Pre-study

The pre-study served as a step of validation of the current steps undertaken until now. The participants in the pre-study were four explainability experts.The pre-study was structured into two main parts: a presentation

and a discussion. In the presentation, the objective of the thesis was presented and the current progress was shown. This was followed by a discussion, where the experts provided feedback and made suggestions regarding possible changes and improvements that could be undertaken in the newly created code system.

Following changes were undertaken after the discussions:

- *No time mentioned* was renamed into *Time independent*
- Bugs and *Unexpected Behaviour* was moved into a separate category, which was named *Behaviour*
- Time aspect was also reclassified as its own category rather than being the top level of the coding system.

These points of discussion also led to the separation of the feature-related labels from the rest of the aspects and the creation of another new category, the *Feature* category. This would be similar to *Time Aspect* and *Behaviour* in being independent of the other categories. Further labels were then added to this category. Furthermore, following points were taken in consideration and although they were not reflected directly into the next iteration of the criteria, they were implemented in a later iteration:

- *Feature request* and *Feature missing* are too similar and should be merged
- *Costumer support* and *Business* should be in another category than the rest of the system aspects since they are not similar with the rest

This resulted in the development of a new code system. Unlike the previous version, which had a single main category (*Time Aspect*), the updated system consisted of four independent top-level categories. Instead of assigning each explainability need a single label from the hierarchical structure, each explainability need could be now categorised into four different categories. These categories were: *Time Aspect, Behaviour, Feature* and *App Aspect*. A representation of the sub-categories included in each of the top-level categories is displayed in Table 4.3. All these top-level categories will serve as elements for the final list of criteria which are to be gathered in this theses.

The difference of this coding system, from the previous iterations can be made easily noticeable with an example, like the one used in Section 4.6.1. We get a review where the explainability label is "search bar moved". With the first categorisation displayed in Table 4.1, this label could be categorised in both *Update-related* and *UI/UX*. This would not be optimal because it would violate the mutual exclusivity principle of a hierarchical coding system. In the second iteration, this label could be categorised under *Update* and than *Feature changed*, but the aspect which was changed would not be clear.

Time aspect	Behaviour	Features	App aspect
Future	Bug / Unexpected behaviour	Feature changed	Account
Update	Not categorised	Feature removed	Connectivity
Time independent		Feature missing	Content
		Feature request	Costumer support
		Unclarity about feature	General app idea
		Definition	Hardware
		Unhappiness about feature	Business
		No Feature	Security
			UI/UX
			Main app functionality
			Accessibility
			Meta

Table 4.3: Code system after pre-study

With the current version of the coding system, this label would receive four categories: *Update* in the *Time aspect* criterion, *Not categorised* in the *Behaviour* criterion since it does not mention any bug or unexpected behaviour, *Feature changed* in the *Feature* criterion and *UI/UX* in the *App aspect* criterion. Through these criteria, a developer could directly understand that the review is related to an issue after an update, regarding a feature that was changed in the UI, without having to read the full review at all. This is summarised in Table 4.4.

Example	First iteration codes	Second iteration codes
"search bar moved"	UI? Update related?	Time: Update
		Feature: Changed
		Behaviour: Not categorised
		Aspect: UI

Table 4.4: Coding example for the different iterations

4.6.3 Adapting the new criteria

In this iteration, there were also some problems that were noticed, that prompted for some further changes. This included joining the *Unexpected behaviour* and the *Bug* category in the *Behaviour* criterion, since a clear separation between them was not possible. The *Features* criterion was expanded by adding the *No feature* category, which would tackle the missing of a category for explainability needs that did not mention any feedback regarding a feature in particular. Furthermore, the *App aspect* criterion was also expanded by adding the categories: *Main app functionality* which covers the main functions of an app, *Accessibility* and *Meta*. Finally, the category *UI/UX* was separated in *UI* and *UX*.

4.6.4 Subset Validation

Before using the newly created criteria in the interviews, another validation step was completed. A subset of the original dataset was labeled manually by the writer and the supervisor of this thesis. The goal of this validation was to check the completeness of the created criteria when comparing to the first iteration and the taxonomy from Droste et al. [15]. For this step, a total of 104 reviews was selected from the dataset. This subset included at least three reviews, when possible, for each category in the taxonomy from Droste et al. [15] and from the first iteration of the code system, shown in Table 4.1. Following results were reached after the labeling from the two raters:

In total 416 codes were distributed with 119 disagreements. The total agreement rate was 71,39%. For the *Time aspect* category, there were only three disagreements. The Inter-rater agreement was 97,11% while the Cohen's Kappa was 0,91 which hints to an "Almost perfect agreement" according to Cohen. In the *Behaviour* category there were 28 disagreements, which meant an Inter-rater agreement of 73,07%. The Cohen's Kappa value was 0,46 which hints to a "Moderate agreement". The *Feature* criterion had the lowest agreement rates, with 48 disagreement labels, meaning an Inter-rater agreement of 53,84%. The Cohen's Kappa value was 0,38 which means a "Fair agreement rate". The *App Aspect* category had 40 disagreements. The Inter-rater agreement was 61,53% and the Cohen's Kappa value was 0,56. This means a "Moderate" agreement was reached. The results are summarised in Table 4.5.

Most of the disagreements in the *App Aspect* criterion came because of an insufficient definition of the *Content* and *UX* category beforehand. In the *Feature* criterion, most of the disagreements came because of the *No Feature* and *Unclarity about feature* categories. To tackle some of the problems encountered during this validation step, some final changes were undertaken

Category	# Disagreements	Inter-rater agreement	Cohen's Kappa
Time aspect	3	97.11%	0.91
Behaviour	28	73.07%	0.46
Feature	48	53.84%	0.38
App Aspect	40	61.53%	0.56

Table 4.5: Agreement between raters

to the coding system. Through this validation step though, it was clear that these criteria could cover both of the aforementioned classifications. The final form of the coding system is presented in the following section.

4.6.5 Final adjustments

Several changes were made to the previous version of the new coding system. Some of the categories were renamed. *Main app functionality* was renamed to *Base functions*. This included the subcategories *Notifications*, *Data exchange and synchronisation*. *General app idea* was renamed *Development rationale* since it considers the reasons behind a decision when developing the app. The *App Aspect* category was separated in to main subcategories, *System aspect* and *Non-system aspect*. The latter includes *Hardware, Business and Costumer Support*. The *System aspect* category includes the remaining subcategories that are directly related to the system or application. These aspects were seen as separated from the rest because they do not involve a concrete aspect of the application itself but rather the surrounding part of it.

The *Features* criterion was renamed into *Features feedback*. The new label *Feature present?* was added to cover the cases in which the users ask whether a feature is actually present in the app or not. This substitutes the *Feature missing* category, which was deemed redundant. The *Definition* category was substituted with *Tutorial needed*. *Unhappiness about feature* was removed since it was covered from another criterion (Sentiment of review) in the final list.

During the labeling from the two raters, a rule emerged which connects the *Features feedback* and *Aspect* criterion. If the aspect selected in a *Non-system aspect* than in the *Feature feedback* criterion, the value *No feature mentioned* should be selected, since it does not affect a feature of the app.

The *Behaviour* criterion was renamed into *Unexpected system behaviour*. This criterion was restructured and expanded. The two main categories were *Bug* and *Not categorised*. The *Bug* category would be expanded with the subcategories *Performance degradation, Data loss, Incorrect output, Unexpected interaction outcomes, Inconsistent behaviour*. These subcategories

would try to cover most kinds of bugs that can happen in an app.

The *Time aspect* criterion would remain unchanged structurally and only get renamed to *Time context*. The final form of the newly formed criteria, together with a short definition and an example will be presented below. Table 4.6 summarises all categories in a more concise way.

The *Time context* criterion consists of the categories *Future, Update and Time independent*.

1. Future

   Definition: Labels all explanation needs regarding future plans of developments or updates for the app.

   Example: Are you going to add a search function?

2. Update

   Definition: Can be used to label all questions regarding update related problems.

   Example: Why is the app so unstable since your last update?

3. Time independent

   Definition: Every review that does not have a specific time anchor.

   Example: How can I upload a new profile picture?

The *Unexpected system behaviour* criterion consists of the categories *Bug* and *Not categorised*. The latter can be used to label reviews which are not mentioning a concrete bug. The category *Bug* meanwhile, is further divided in the following subcategories:

1. Performance degradation

   Definition: Regards all reviews which mention a worsening performance.

   Example: Why does the app take so long to download a song?

2. Data loss

   Definition: Can be used to label reviews that address issues regarding data loss from the users.

   Example: My notes have been erased completely. How can that be?

3. Incorrect output

   Definition: Can be used to label explainability needs regarding false output.

Example: Why does the app show me that I walked only 500 steps when I actually did over 3km?

4. Unexpected Interaction Outcome

   Definition: Can be used to address unexpected output issues.

   Example: I always get an error popup when I try to reply to a comment. Why is that?

5. Inconsistent behaviour

   Definition: Can be used to label explainability needs regarding problems with inconsistent behaviour from the app.

   Example: Why is it that everytime I open your app, the first time I go into a chat I can not see the messages. It just keeps loading. After I close it and reopen it, it works? What am I doing wrong?

This criterion additionally includes the category *Other* for the cases that might not be covered in any of the categories mentioned above.

The *Features feedback* criterion is divided in eight categories.

1. Feature changed

   Definition: Regards all reviews which mention change in a feature.

   Example: Why did you move the search bar from top to bottom?

2. Feature removed

   Definition: Can be used to label reviews that mention the removal of a previous feature.

   Example: Why did they remove the option to clear all pages?

3. Feature request

   Definition: Can be used to label feature requests from users.

   Example: I don't understand why isn't there already a dark mode for this app. Can you add it in your next update?

4. Unclarity about feature

   Definition: Can be used to address some unclarities that the users could have about different features.

   Example: What does this animal actually do other than ask for food?

5. Feature present?

Definition: Can be used to label cases in which a user is unsure whether the feature is present in the app or not. It could also be a case in which the user just can not find a function.

Example: Is there a possibility to change my Email because the app is too unintuitive for me to find it myself.

6. Tutorial needed

   Definition: Regards reviews where the users need help when navigating the app or do not understand a concept in it.

   Example: How do I get to use the ECG to check if I have normal sinus rhythm?

7. No feature mentioned

   Definition: Can be used to label reviews that do not mention explicitly a feature. This can be used in relation to the *Non-system aspect* category of the *Aspects* criteria, since it does not cover app-aspects.

   Example: Why does this app consume so much RAM?

This criteria also has the category *Other* for the cases that might not be covered in any of the categories mentioned above.

The *Aspect type* criterion is divided into two main categories *System aspects* and *Non-system aspects*. The categories and subcategories will be presented below. The division in subcategories is not complete. This means that if a review can not be labeled with one of the subcategories, it just receives the label of the category. The subcategories serve the purpose of better gathering similar explanation needs but do not cover all faucets of each category.

Firstly, the *System aspects* will be presented. The *Account* category is divided into *Login* and *Profile data*. This category tries to group all explanation needs related to an account that a user made for an app.

1. Login

   Definition: This subcategory gathers all labeled reviews which address questions regarding the login process.

   Example: Since yesterday I cant login in your app. What is going on???

2. Profile data

   Definition: Here, questions regarding the profile data of a user can be gathered. This might include for example, changing some personal data information or even resetting a password.

Example: How can I reset my password on your app?

The *Content* category aims to address issues that the users might have with the content that an app provides to its users. The most common cases can be divided in the *Advertisement* and *Feed* subcategories.

1. Advertisement

   Definition: This subcategory includes unclarities regarding the ads that an app might show.

   Example: Why are there so many ads your app??? Its impossible to do anything without being bombarded with ads.

2. Feed

   Definition: This subcategory groups questions regarding the content displayed on their feeds. This is especially relevant for social media apps, since mostly them have this feature.

   Example: Why do i see posts from people i do not know in my feed??

The *Development rationale* category gatherers reviews in which users have questions regarding why a specific decision was made in the development process.

Example: Why can we only make one mistake before having to wait to try again? How can we learn through this app if we are not allowed to make mistakes?

The *Security* category aims to group unclarities regarding the security features of an app. It is divided in to subcategories.

1. Permissions

   Definition: This subcategory gathers reviews in which users are not sure why the app needs some permissions from.

   Example: Can you tell me why does your app need access to my contacts? It does not make any sense.

2. Verifications

   Definition: Here, reviews with unclarities regarding the verifications that an app might require are grouped.

   Example: Why do i have to verify myself in two separate apps before managing to access your app? Is very cumbersome.

The *Base functions* category focuses on essential app features that support its core functionality. This category tries to cover functions that most of the

apps posses by default. The subcategories here are *Notifications* and *Data Exchange and Synchronisation.*

1. Notifications

   Definition: This subcategory gathers reviews where users are unclear about the app's notification settings or behavior, such as why certain notifications are received or how to manage them.

   Example: Why am I still getting notifications even after I turned them off in the settings? It's really annoying.

2. Data exchange and synchronization

   Definition: This subcategory includes reviews where users are unsure about how data is shared or synchronized between devices or platforms, including issues with cloud syncing or data transfer between apps.

   Example: Why are my notes not syncing between my phone and tablet? They always seem out of sync, and it is frustrating.

The *UI* (User Interface) category addresses user concerns and confusion about the app's design and layout, focusing on how users interact with visual elements like buttons, menus, and navigation structures.

Example: Why did you change the location of the search bar? It used to be at the top and it was way better like that.

The *Accessibility* category gathers reviews related to the app's usability for people with disabilities or those needing specific accessibility features, such as screen readers, larger fonts, or voice commands.

Example: I'm visually impaired, and the text in your app is too small. Is there any way to increase the font size?

The *Domain-specific* category groups explainability needs regarding domain-specific issues. This are related to the app in question only and can not be related to other applications. It can be divided in two subcategories.

1. App-specific functions

   Definition: This subcategory deals with reviews related to app-specific functions that are not available at most apps but rather only for the app for which the review is written.

   Example: How can i add a podcast to my favorites list? (App-specific function on Spotify)

2. Definition needed

Definition: This subcategory includes reviews where users ask about the definition specific terms they might encounter on the app.

Example: What does COP mean?

The *Meta* category gathers reviews where the explainability need is not very clearly expressed from the user.

Example: Can I please have the HBO max app back?

Finally, the *Non-system aspect* categories will be presented.

The *Hardware* category focuses on issues related to the compatibility and interaction of the app with physical devices and hardware components. It includes the following subcategories:

1. Device support

   Definition: This subcategory gathers reviews where users are unclear about the app's compatibility with specific devices or operating systems.

   Example: Why doesn't this app work on my tablet? It only seems to be optimized for phones.

2. Connectivity

   Definition: This subcategory includes reviews where users are confused about connection issues, such as problems with Bluetooth, Wi-Fi, or mobile data connections.

   Example: The app constantly loses connection to my smartwatch. Can you fix the Bluetooth sync issues?

The *Business* category addresses user concerns about the app's commercial and transactional aspects. It is divided into the following subcategories:

1. Financial Concept

   Definition: This subcategory includes reviews where users express confusion about the app's pricing model, subscription plans, or hidden fees.

   Example: Why am I being charged monthly when I thought it was a one-time payment?

2. Orders

   Definition: Reviews that discuss problems or uncertainties with placing or tracking orders are grouped here.

   Example: I placed an order through your app, but it's not showing up in my account. What's going on?

3. Payment

   Definition: This subcategory focuses on confusion or issues related to how users pay for services or products within the app.

   Example: Why can't I use PayPal to make payments? It only shows credit card options.

4. Refunds

   Definition: This subcategory groups reviews where users are unclear or frustrated about the refund process.

   Example: I requested a refund over a week ago, but I haven't seen any updates. How long does it take?

The Customer support category gathers reviews related to the quality and responsiveness of the app's customer service. It has the following subcategories:

1. Not responding

   Definition: This subcategory includes reviews where users complain about not receiving a response from customer support.

   Example: I have tried calling your customer support numbers during working hours but nobody picks up. How can i actually contact you?

2. Useless response

   Definition: This subcategory covers reviews where users feel that customer support responses were not helpful or did not resolve their issue.

   Example: Support told me to reinstall the app, but that did not fix my problem. Why is there not a better solution?

Time Context	Unexpected System Behaviour	Subcategories	Features Feedback	Aspect type	App Aspects	**Subcategories**
Future	Bug	Performance Degradation	Feature(s) changed	System aspect	Account	Login, Profile data
Update		Data Loss	Feature(s) removed		Content	Advertisement, Feed
Time Independent		Incorrect Output	Feature(s) Request		Development Rationale	
		Unexpected Interaction Outcomes	Unclarity About Feature(s)		Security	Permissions, Verifications
		Inconsistent Behaviour	Feature(s) present?		UI	
		Other	No feature mentioned		Meta	
	Not categorised		Tutorial needed		Base functions	Notifications, Data Exchange and Syncronisation
			Other		Domain-specific	App-specific Functions, Definition Needed
					Accessibility	
					Other	
				Non-system aspect	Hardware	Device support, Connectivity
					Business	Financial Concept, Orders, Payment, Refunds
					Costumer support	Not responding, Useless Response
					Other	

Table 4.6: Final version of the newly created criteria

4.7 Interviews

To validate the final list of criteria, a series of interviews were conducted with six software engineers. The interviewees were between the ages of 25 and 38. Four of them work at medium-sized companies, while two work at global conglomerates. Most of them had 1 to 4 years of experience with requirements engineering, while one of them has 18 years of experience. The interviewees were also asked how much they have to do with explainability needs in their day-to-day work. They selected between a 1 to 5 scala, where 1 stands for *very much* and 5 stands for *very little*. The results were mixed, ranging from 1 to 4. The average of the answers was 2.5, which is the middle ground of the scale. Each interview was structured into three distinct parts.

In the first part, a brief presentation of the thesis was provided, outlining its objectives, introducing the concept of explainability, and presenting the proposed criteria along with the identified use cases. This ensured that the interviewees had a clear understanding of the context and purpose of the research.

In the second part, the participants were asked to prioritize the criteria according to their relevance for each use case, providing valuable feedback on their practical applicability. To do this, the interviewees gave a rating from a scale ranging from 1 to 4, where 1 stands for *high priority* and 4 stand for *low priority*. The scale with only 4 values instead of 5 was selected in order to encourage the interviewees to select whether they though the criteria was rather helpful or rather not, instead of relying on a neutral value in their evaluation. This way the results can be more polarised and more insightful. If the interviewees thought that a criteria was not at all helpful or related to one of the use cases, they were marked with a -.

Finally, the third part included questions about the interviewees' demographic backgrounds, which were covered above, and regarding suggestions for potential extensions or improvements to the study in general. The interviewees were asked if they though that these criteria could be useful in another use cases which was not covered in the interview. Furthermore, they were asked if they would use these criteria in their current work environment. They were also asked if they could think of further criteria that were not already mentioned, that could be helpful in these use cases, or in the use cases that were added from them.

Finally, the interviewees were asked about their general thoughts about the interview and the study. The questions included asking for suggestions regarding possible changes in the study, how useful they though this research was and finally, whether they had any other notes regarding the interview or research. The main steps of the interview process are summarised in Figure 4.3 through the Flow-Method [32] [31].

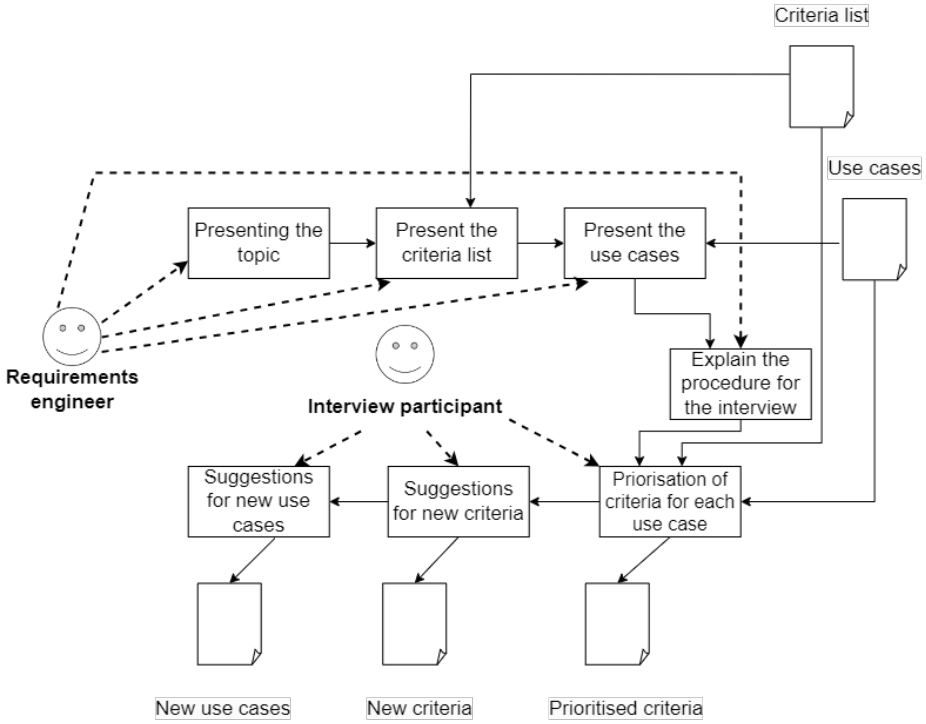

Figure 4.3: Interview process in the Flow-Method

Chapter 5

Results

In this chapter, the results of this thesis will be discussed. Firstly, the final list of criteria that was gathered will be presented. This includes the definitions for each criterion, their purpose and how the criteria differentiate from one another. Finally, the results of the interviews will also be presented. This includes the prioritisation part, in which the participants were asked to prioritise the criteria regarding their use in the different use cases and the more general questions that were asked regarding further extensions of the criteria and the use cases and also the usefulness of the study.

5.1 Final list of criteria

This list is created by gathering criteria through a mixture of sources. Some of the criteria are app-specific metadata that can be gathered from the reviews metadata, like the app version or the hardware used by user that wrote the review. Other criteria can be gathered by using NLP techniques on the reviews. Examples here are the sentiment of the review, the length or punctuation of the review. Another source to gather criteria were related works, such as the research from Droste et al. [15] and Unterbusch et al. [34]. Finally, the criteria created in this thesis are also added to the list. The list of criteria as well as their contribution to categorising and prioritising explainability needs, together with the origin of each criteria is presented below:

1. Username

   Contribution: Enables identification of frequent or influential users who may provide detailed feedback. Repeated feedback from the same users may highlight persistent issues that may require closer attention.

Origin: Review metadata

2. App version

   Contribution: Helps assess if explainability needs are linked to specific app versions. This can prioritize updates or fixes to newer releases, or flag recurring issues in older versions.

   Origin: Review metadata

3. Hardware used

   Contribution: Helps assess if explainability needs are tied to specific devices. Device-specific issues, such as layout differences or performance problems, can be prioritized differently than other issues.

   Origin: Review metadata

4. OS

   Contribution: Identifies OS-specific explainability needs. If a certain platform (e.g., Android) has more issues, targeted support or updates can be created for that OS.

   Origin: Review metadata

5. Date of review

   Contribution: Tracks trends over time, allowing teams to see if certain updates or seasons cause spikes in reviews with explanation needs. Reviews from key dates like after major updates can be prioritized in order to fix the possible problems that came with the update.

   Origin: Review metadata

6. Genre

   Contribution: Helps identify patterns in explainability needs based on app type. Apps in different genres (e.g., gaming vs. finance) may require varying levels of explanation, helping teams focus on genre-specific requirements. This criterion is mainly useful when doing a more broad research, which touches different types of apps, instead of just one app. So for the purpose of evaluating only the own app, it might not be relevant.

   Origin: Review metadata

7. Name of app

Contribution: Allows for tracking which apps generate more explainability needs. Frequent issues tied to specific apps can guide targeted updates or additional user education. Similarly to the Genre, this criterion also is more helpful when dealing with different apps and can be used to group explainability needs based on the specific apps. It may not be relevant in a context where only one app is being evaluated.

Origin: Review metadata

8. Frequency of explanation need

   Contribution: Explanation needs that are repeated in a lot of reviews show that the same problem might be affecting a number of users, which might suggest a widespread confusion. Addressing these ensures a broad impact, improving the experience for many users.

   Origin: Review metadata

9. Sentiment of review

   Contribution: Negative sentiment highlights critical areas where users are frustrated or confused. Prioritizing negative feedback ensures the most urgent explainability needs are addressed first.

   Origin: Review text, can be gathered through NLP methods

10. User experience (UX)

    Contribution: Can help categorise the described user experience in the review. If the user mentions that they had an awful user experience when using the app, than the review might need to be prioritised higher. This can sometimes overlap with the sentiment of review, since a negative user experience can reflect a negative sentiment in the review. But there might be cases in which the two might differ, depending on how the user writes their review, therefore this criterion can still be helpful.

    Origin: Review text, can be gathered through NLP methods

11. Length of review

    Contribution: Longer reviews might often indicate more complex issues, which may require more attention. Also, the users that take the time to voluntarily write a long review, describing their issues they had with the app, might be more committed to the app and therefore, prioritizing these reviews can help resolve deeply rooted user misunderstandings and increase user satisfaction.

    Origin: Review text, can be gathered through NLP methods

12. Punctuation

    Contribution: Reviews with a lot of punctuation like several question marks or exclamation points can signal urgency or frustration. Prioritizing these reviews may help address pressing user concerns.

    Origin: Review text, can be gathered through NLP methods

13. W-Questions

    Contribution: Can help indicate the interaction that caused the explanation needs for the user. Can be more helpful when combined with the Verbs criterion.

    Origin: Review text, can be gathered through NLP methods

14. Verbs

    Contribution: Indicate what the user are trying to do that causes them issues. When combined with the W-Questions, we can get questions like "How...change", or "Why...did remove" which provide a context and the action that created the issues for the user.

    Origin: Review text, can be gathered through NLP methods

15. Costs

    Contribution: Supports decision-making by weighing the cost of addressing certain explainability needs. This can be expressed in either a fixed amount of money or in "programmer days". High-impact, low-cost solutions can be prioritized, in order to optimize available resources. The way that this criterion is implemented and how the estimations are made depend of the needs of the user of this list.

    Origin: Can be estimated from an expert depending on the context of the user of this list

16. Explanation need expression

    Contribution: The explanation need expression can be either explicit or implicit, as mentioned in the Droste et al. taxonomy [15]. Explicit explanation needs issue directly what their problem is, while implicit ones need more work because it might not be directly clear what the issue is.

    Origin: Criterion taken from the research of Droste et al. [15]

17. Targeted interaction type

Contribution: This criterion also derives from the work of Droste et al. [15] as it is a category of the taxonomy presented by them. They define it as the explanation needs that arise when a user wants to know how a certain operation with the software can be performed. They divide it into three subcategories: Operation, Navigation and Tutorial. Tutorial is covered in the criteria *Feature Feedback* and *App Aspects* through the subcategory *Definitions*. Therefore, only the first two categories are relevant in this list. This category can also have overlaps with the mixture of the categories W-Questions and Verbs. This is not guaranteed though, because it is dependant on how the users write their reviews. Also in the cases in which the explanation need expression is implicit, this category is more useful than the other two mentioned previously, because they can not indicate an implicit explanation needs sufficiently.

Origin: Criterion taken from the research of Droste et al. [15]

18. Prioritization of explanation need

    Contribution: This is a criterion taken from the Unterbusch et al. taxonomy [34]. The implementation here, similar to the costs, depends on the context and the needs of the users of this list. The reviews and the explanation needs can be categorised in Primary and Secondary Concerns, based on the classification from Unterbusch, or they can be classified differently depending on the specific needs of the evaluation process. This criterion can help in the prioritisation process by better dividing the available resources.

    Origin: Criterion taken from the research of Droste et al. [34]

19. ISO-25010 Quality model

    Contribution: The quality model [1] is the very important part of a product quality evaluation system. It determines which quality characteristics will be taken into account when evaluating the properties of a software product. Aligning explainability needs with recognized quality standards helps ensure the app meets key attributes like usability and reliability and that it fulfills world-known standards. This can guide prioritisation process.

    Origin: Quality model ISO-25010 [1]

20. Time Context

    Contribution: This criterion was added during the development of this thesis, similar to the rest of the criteria presented below.

It can help in determining whether the explanation needs are related to an update or rather about the future plans for the app development.

Origin: Criterion created through the Bottom-Up approach in this thesis

21. System Behaviour

    Contribution: Can be helpful in determining what kind of unexpected system behaviour is causing the explanation needs for the user.

    Origin: Criterion created through the Bottom-Up approach in this thesis

22. Features feedback

    Contribution: Determines what kind of problems the users have with the features of the app that caused the explainability needs for them.

    Origin: Criterion created through the Bottom-Up approach in this thesis

23. Aspect type

    Contribution: Explains which aspect is confusing to the users. It can be a system aspect(e.g. UI) or a non-system aspect (e.g. Customer support). This helps in categorising and prioritising depending on the most affected aspects and the ones that are deemed most important. Solving explainability needs in both cases can improve user satisfaction.

    Origin: Criterion created through the Bottom-Up approach in this thesis

5.2 Criteria uniqueness

In order to evaluate if the criteria had overlaps between one another, a criteria uniqueness matrix was created. This matrix had N rows and columns, where N is the number of criteria in the list. Both x and y axis included all the criteria mentioned above. In the cells in which a overlap could be possible, a brief note was inserted, while in the rest of the cells where no overlap was present, a "-" was added. The diagonal was marked with an "x", since the same criteria was present in both axis. Following overlaps were noticed:

As previously mentioned, the criteria Sentiment and User Experience can be similar, as a positive or negative user experience often translates into a

similar sentiment in user reviews. However, this is not always the case, as it depends on how the user chooses to express their thoughts in the review. Despite the overlap, it remains useful to consider both criteria separately, since they can provide different insights based on user feedback.

The category *Unclarity about feature* might often translate to a bad *UX*, therefore an overlap can also exist in this case. However, this is considered a relatively weak overlap, as it depends heavily on how the user responds to the unclarity. Another possible weak overlap for the *UX* criterion, and maybe also *Sentiment*, can be with the criterion *Punctuation*. For example, excessive use of question marks or exclamation points may signal frustration or confusion, which could imply a negative sentiment or poor UX. On the other hand, this could simply be a user's writing style, with no significant connection to their actual experience.

The criteria *Date of review* and *Time context* can also have overlaps, since they both are related to the time context of the review. These overlaps might mainly come from the *Update* category which is present in the *Time context* criterion. Both these criteria can be used to trace problems with an update. The *Date of review* criterion however, can be relevant not just for this case, but also for other cases, which are not covered in the *Time context* criterion, such as a server failure for example that might trigger several reviews with explanation needs. Furthermore, the *Time context* criterion, includes the *Future plans* category which can not be related to the *Date of review* criterion.

The criteria *Verbs* and *W-Questions* can indicate a possible interaction that the user wanted to achieve with the app. This can also be covered from the *Targeted interaction type* criterion. There can be cases though when the interaction type can not be expressed through questions or verbs, like for example, when the explanation needs are expressed implicitly. In such cases, these criteria serve different functions, making it valuable to retain all of them for a more comprehensive analysis.

Finally, the *ISO-25010* [1] criteria also has some overlaps with other criteria. One of these overlaps is the presence of the *Security* category, which appears in both the *ISO-25010* and *App Aspects* criteria. However, the subcategories within each criterion differ, suggesting that while they cover similar concerns, they do so from distinct perspectives.

Furthermore, the categories *Performance efficiency* and *Reliability* in the *ISO-25010* criteria are similar to the *Bug* and *Unexpected system behaviour* subcategories. However, *Performance efficiency* and *Reliability* extend beyond just detecting bugs, as they assess the overall system's ability to function as expected under different loads and environments, addressing more structural or architectural issues that may not always be visible through

isolated bugs. The *Bugs and Unexpected System Behaviour* criterion on the other side, serves to group reviews based on the type of bug the user encountered in the app that made them have explainability needs. Keeping all of these criteria on the list can offer a broader perspective, although some overlaps might exist.

5.3 Interview results

In this section, the results of the interview process will be presented. These will be divided in two subsections. Firstly, the priorities of the criteria for each of the previously mentioned use cases will be discussed. In order to keep the presentation of the results in this section concise, only the most important criteria per use case will be discussed. This includes the criteria that have an average and median lower or equal than 2, which stands for a rather high priority. Lastly, the answers to the more general questions will also be presented. These include possible extensions to the use cases and criteria, and also the opinions of the participants regarding this research.

5.3.1 Criteria priorities

1. Use case: Elicitation of requirements

For this use case, most of the criteria, especially the ones related to the review metadata and those that can be gathered from NLP methods, were considered not relevant since these criteria can not be used to gain requirements from the stakeholders. Examples of such criteria include the *App version, Hardware, Date of Review, Costs* or *Punctuation*, *Verbs* and *W-Questions*.

From the rest of the criteria, *Aspects* and *Targeted interaction type* were deemed the most relevant for this case. Additionally the *ISO-25010 Quality model* and the *System Behaviour* criteria were also prioritized highly. These criteria were viewed as particularly useful, because of the fact that these criteria provide categories and subcategories that can aid in the process of eliciting explainability requirements for an application. These criteria can offer a strong basis when gathering requirements which can lead to a better development process.

2. Use case: Manual categorisation of explainability needs in app reviews

For this use case, the criteria *Aspects, System behaviour* and *Targeted interaction type* were seen as the criteria with the highest general priority. The participant mentioned that through these criteria, you could a get a good perspective on what the issue in the review is.

Furthermore, the *App version* and the *Costs* were also seen as very important. The *App version* criterion helps in aligning user-reported issues with specific versions of the app, making it easier to identify and address problems efficiently. The *Costs* criterion depends strongly on how it can be implemented and which person estimates the costs. If this can be implemented correctly by the user of this list, it can be a very important criterion when prioritising app reviews.

Lastly, the *User experience, Time context* and *Features feedback* criteria were also rated as high priority criteria. These criteria were also seen to offer interesting insights that can be helpful in the categorisation and prioritisation of explainability needs in app reviews.

3. Use case: Automatic categorisation of explainability needs in app reviews

For this use case, the criteria selected were mostly similar to the manual categorisation use case with a few differences. The *Costs* and *User Experience* criteria were assigned lower priority in this use case, as these aspects were considered more suited for manual assessment by a person rather than being reliably handled by automated systems.

Furthermore, the criteria *W-Questions and Verbs* were seen as important for this use cases. Participants noted that these criteria are well-suited for automation since they can be efficiently identified and processed by NLP algorithms, offering a fast and accurate way to categorize explainability needs in reviews.

The full list of results from the prioritisation part of the interview are summarised in Table 5.1. This includes the specific priority given by each participant for each use case (column I1-I6) as well as the average and median of each criteria per use case.

Criteria / Use cases	Elicitation								Manual categorisation								Automatic categorisation							
	I1	I2	I3	I4	I5	I6	Avg	Median	I1	I2	I3	I4	I5	I6	Avg	Median	I1	I2	I3	I4	I5	I6	Avg	Median
Genre	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Name of app	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Username	-	-	-	-	-	-	-	-	3	4	4	3	3	-	3.5	3.5	3	1	3	3	3	-	2.83	3
App version	-	-	-	-	-	-	-	-	4	1	1	2	1	2	1.83	1.5	4	1	1	2	1	2	1.83	1.5
Hardware used	-	-	-	-	-	-	-	-	1	3	2	2	2	4	2.33	2	1	3	2	2	2	2	2	2
OS	-	-	-	-	-	-	-	-	3	3	2	1	2	4	2.5	2.5	3	3	2	1	2	3	2.33	2.5
Date of review	-	-	-	-	-	-	-	-	3	2	3	3	4	2	2.83	3	3	2	3	2	4	2	2.66	2.5
Costs	-	-	-	-	-	-	-	-	3	1	1	2	2	1	1.66	1.5	3	4	2	4	2	1	2.66	2.5
Sentiment of review	-	-	-	-	-	-	-	-	3	2	2	3	3	2	2.5	2.5	2	2	4	2	3	2	2.5	2
User experience (UX)	-	-	-	-	-	-	-	-	1	2	2	2	2	1	1.66	2	3	2	4	1	2	1	2.16	2
Targeted interaction type	1	1	1	1	3	1	1.33	1	1	1	1	1	3	2	1.5	1	1	1	4	2	3	2	2.16	2
Length of review	-	-	-	-	-	-	-	-	4	4	2	4	2	3	3.16	3.5	3	4	2	3	2	3	2.83	3
Punctuation	-	-	-	-	-	-	-	-	4	4	4	4	3	3	3.66	4	3	4	4	3	3	3	3.33	3
Time context	-	-	-	-	-	-	-	-	2	2	2	2	3	1	2	2	2	2	3	1	3	1	2	2
Verbs	-	-	-	-	-	-	-	-	1	2	4	3	3	4	2.83	3	1	1	1	1	3	3	1.66	1
W-Questions	-	-	-	-	-	-	-	-	1	2	4	3	3	4	2.83	3	1	1	1	1	3	3	1.66	1
Frequency of explanation need	-	-	-	-	-	-	-	-	2	3	2	2	2	3	2.33	2	2	4	2	1	2	2	2.16	2
ISO-25010 Quality model	1	3	1	2	2	1	1.66	1.5	1	3	4	4	2	3	2.83	3	1	3	4	4	2	3	2.83	3
App Aspects	2	1	2	1	1	1	1.33	1	2	1	1	1	1	2	1.33	1	2	2	1	1	1	2	1.5	1.5
Features feedback	3	4	2	4	2	2	2.83	2.5	1	2	2	1	2	2	1.66	2	1	2	2	2	2	2	1.83	2
System Behaviour	4	1	1	3	1	1	1.83	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Priorisation of explanation need	-	-	-	-	-	-	-	-	3	3	4	2	3	2	2.83	3	2	3	4	3	3	2	2.83	3
Explanation need expression	-	-	-	-	-	-	-	-	1	3	3	4	4	4	3.16	3.5	3	3	4	3	4	4	3.5	3.5

Table 5.1: Full results from the interview process

5.3.2 Rest of the questions

In the final part of the interview, participants were asked for suggestions on extending the list of criteria and use cases and also whether they would use these criteria in their work environments. The first question asked participants if they thought the criteria list could be applied in additional use cases. The interviewees proposed several new use cases, which are summarised below:

- Ticket management system
- Categorising offer inquiries
- Categorise general user feedback
- Categorise general user user experience in service-oriented industries

The suggestions included the usage of the criteria within a ticket management system, such as JIRA[1] , where users could categorise the client-submitted tickets using these criteria. Another proposed use case was the usage of criteria to categorise and process offer inquiries. One participant explained that after an offer for the development of a requirement has been sent, often it returns with questions or unclarities from the client. These criteria can be used to categorise these questions. Other participants also mentioned that the criteria can be used to categorise general user feedback and user experience in service-oriented industries, by extending the scope of the usage of the criteria outside just categorising app reviews.

Furthermore, the participants were asked whether they would use these criteria in their work environment and in what contexts. They mostly answered that they would use them to categorise client feedback or for incident analysis. The criteria could be even more helpful if it could be integrated in the ticket management systems that are used commonly to categorise clients tickets. Only one of the participants answered with "No".

Another question was whether the participants could expand the criteria list with further criteria that had not been previously mentioned. Following new criteria were suggested:

- Escalation level
- Star rating
- Age of the user
- Hours of usage by the user of the app
- Level of expertise of the user

[1] https://www.atlassian.com/de/software/jira

- Priority label

The criteria *Escalation level* and *Priority label* would serve as a priority set by the client or user which writes the review or ticket, depending on the use case. These criteria could significantly impact the prioritization process. *Star rating* is related to the rating that the user gave when writing the review. This can be similar to the *UX* and *Sentiment criteria*. Reviews with a lower star rating could receive higher priority since they would be a bad advertisement for the application. The other suggested criteria included information regarding the user and their experience with the app. This would include the *Age of the user*, *Hours of usage by the user of the app* and the *Level of expertise of the user*, which could prove helpful especially in more complex application. This would allow for better categorising from who the review is coming from and if some explanation needs come from the same users of the same age, or from the ones with the same level of expertise.

The participants were finally asked to evaluate this study. To the question how reasonable did they find this study, 4 answered with "very reasonable" and 2 with "rather reasonable". Furthermore, they were asked whether they would change anything in this study. Four of the participants answered with "No". One participant suggested that the use cases could benefit from more differentiation, as the manual and automatic categorization use cases seemed similar. Another participant suggested leaving the prioritization and categorization tasks for the automatic categorization use case to a machine learning model, which could potentially yield more accurate results.

Finally, the users were asked if they had any other remarks regarding this study. One of the answers was regarding extending the prioritisation columns in order to account for different processes, such as the prioritisation and implementation of the criteria. For these two processes, the prioritisation of the criteria could be different, since some criteria might be more helpful but are harder to implement. Two participants found that conducting interviews rather than surveys was more helpful and advantageous, since it allowed for more interaction and allowed the participants to ask for explanations if something was unclear. The other three participants answered with "No".

Chapter 6

Discussion

In this chapter, the research questions will be answered and the results will be discussed and interpreted. Furthermore, the threats to validity will be presented.

6.1 Answering the research question

In Chapter 1, following research questions were presented:

RQ1 What criteria can support the categorization and prioritization of explanation needs in app reviews?

RQ2 Which of these criteria are prioritized highest by requirements engineers?

To answer **RQ1**, a list of criteria that can help in categorising and prioritising explainability needs in app reviews was developed. The development of this list involved multiple stages, with iterative improvements made throughout the thesis. The final list includes criteria was build through different sources. Firstly, some of the criteria were developed after a study of a dataset that contained around 3,000 reviews with explainability needs. Additional criteria were drawn from metadata that could be found within the reviews themselves, such as information on the user, date of review, and version of app and hardware that was used. Other considerations included review statistics and information obtained through (NLP) techniques. These additions enriched the criteria by introducing linguistic elements like sentiment analysis, punctuation , length of review, verbs used and question asked. Finally, external factors, such as costs associated with addressing the explainability needs, were also incorporated, providing a more practical perspective. Together, these criteria allow for a more structured, precise, and contextually informed approach to identifying and addressing explainability

needs in app reviews. The final list included the following criteria:

1. App Aspect
2. App version
3. Costs
4. Date of review
5. Explanation need expression
6. Features feedback
7. Frequency of explanation need
8. Genre
9. Hardware used
10. ISO-25010 Quality model
11. Length of review
12. Name of app
13. Operation system used
14. Prioritisation of explanation need
15. Punctuation
16. Sentiment of review
17. System Behaviour
18. Targeted interaction type
19. Time context
20. User experience (UX)
21. Username
22. Verbs
23. W-Questions

To these criteria, the suggestions from the participants in the interview can also be added. That included the following criteria:

1. Age of user
2. Escalation level
3. Hours spent from the user using the app
4. Priority label from the user or client

5. Stars given in the review
6. Users level of expertise

Through these criteria one can categorise and prioritise explainability needs in app reviews in a more precise and informed way.

The importance of each criterion varies based on the intended use case, an insight that was central to answering **RQ2**. To explore the criteria's applicability across various scenarios, interviews were conducted with requirements engineering experts who prioritized these criteria according to specific use cases. This process revealed that different use cases often led to different priority levels for the criteria. However, there were also criteria that were considered important for all use cases, such as *App Aspects, System behaviour* or *Targeted interaction type* criteria. Some other criteria that were also frequently highlighted as valuable were the *App version, User Experience* and *Feature feedback*. These criteria offer nuanced insights into the review content, user sentiment, and specific app functionalities that caused the need for explainability, therefore allowing for targeted and meaningful responses to users' explainability needs.

6.2 Discussing the results

From the interview results we can notice that depending on the use case, different criteria are seen as more useful. This is because different use cases need different types of information that is provided by the criteria. Furthermore, the criteria have different implementation availability and costs for different use cases. For example, for the Elicitation use case, metadata-related criteria were largely seen as unhelpful because they do not directly support the gathering of user requirements. Furthermore, the *Estimated Costs* criterion was preferred mostly for the manual categorisation, which is done by a physical person, instead of for the automatic categorisation use case. The participants argued that the estimation would be more accurate and helpful if it came from a person rather than from an algorithm.

The interview participants offered some interesting views on how to further extend this study. They proposed different new use cases in which the gathered criteria from this thesis can be used. The suggested use cases were ticket categorisation and processing in a ticket management system, categorisation of feedback from users or clients, incident management and inquiry processing. While these use cases generally revolve around feedback categorization, they vary by context and require different implementation approaches.

The participants proposed these use cases, because they are in contact with them in their work environments. They felt that this list of criteria could

be useful in their work context, if an implementation for the aforementioned use cases could be achieved. This shows that the criteria have the potential to be helpful to categorise explainability needs not only in app reviews but also in other contexts beyond the original scope of this study.

Furthermore, the interviewees also offered very interesting insights regarding extending the list of criteria. One of the suggestions was to include a type of client-priority indicator, which could be a *Priority label* or an *Escalation level*, which is to be set by the writer of the review or feedback, depending on the use case. This criterion can be useful to stakeholders, such as app developers and project managers, in helping them prioritize tickets or reviews by emphasizing issues deemed particularly urgent by users. Another recommended criterion was the *Star review* given alongside app reviews. This criterion can be useful to highlight reviews, especially in cases where the star review is low, which need to be tackled as soon as possible to avoid having negative reviews for the app, which might discourage potential future users.

This criterion is especially important for project managers, who are responsible also for the reputation of the app and need to make sure that the issues can be prioritised accordingly in order to keep the rating as high as possible. Furthermore, this criterion also allows another important stakeholder, the end users, to directly have an input in the prioritising process.

Finally, the other criteria that were suggested were related to information about the user that wrote the review or the ticket. These include the hours of usage from the user of the app, the age of the user and the expertise level of the user. These criteria can allow for better grouping from who the explainability needs come from and can allow from specific measures. For example, if a lot of the same type of explainability needs come from older users but not from younger ones, this means that there is an age-related usability gap that requires attention, especially if it is an age group that is considered as targeted audience. If some explainability needs come from users that have low levels of expertise, but not from those with a high level of expertise, this means that the app may benefit from more guidance for beginners.

Data scientists can combine these criteria, along with others from the list such as the *App Aspects* criterion, to gain a more comprehensive understanding of their user base and the specific needs that the users face. By analyzing these factors together, they can uncover patterns or trends that may not be apparent when viewed in isolation. For example, researchers might investigate whether there is a correlation between issues related to the *UI* criterion and the *Age of the user* criterion, which could reveal age-specific explainability needs related to the UI of the app. Such insights could help inform targeted improvements, refine user interface design, and

ultimately enhance user satisfaction across different demographic groups. This multifaceted analysis provides a valuable basis for prioritizing updates and tailoring solutions to meet the diverse needs of the app's audience.

The participants were also asked in the final part of the interview regarding their general thoughts about this study. The responded that they found the study to be very useful and that it showed a lot of potential. If the end-objective of creating a pipeline in which app reviews would be automatically categorised and than forwarded to the appropriate person of interest, or even to be automatically responded or acted upon, would be achieved, than this would be a huge step in service-oriented industries. Furthermore, the participants appreciated the choice of conducting this study through interviews rather than through surveys. This was mostly because of the fact that the participants could ask questions if something was unclear and also ask for examples, especially when dealing with the criteria list.

Finally, the participants made some suggestions regarding potential changes in the organisation of the study. It was suggested that for the automatic categorisation use case, the criteria could be selected by a machine learning model instead of a human, since this could produce more accurate results. Furthermore, the priority columns for the criteria could be extended to include different processes, such as the priority of the criteria but also the implementation feasibility. This is because some criteria can be seen as useful, but are in fact harder to implement, which can affect the general priority of a criteria for a use case. Therefore, a separated evaluation for each of these processes regarding the criteria can offer more precise information.

6.3 Threats to validity

In order to discuss the *Validity Threats* of this master thesis, the *Threats* presented by Wohlin et al. [38] will serve as the basis.

The number of participants in the interview process (6) could be relatively small. The participants mostly work in medium-sized companies, while the rest work at big companies. Also, the participants do not deal much with app reviews in their work environment. This relatively narrow range of backgrounds could limit the diversity of perspectives, which may impact the ability to generalise the findings of this study. This could influence the validity of the results when trying to generalise them. This results in a *Threat to Conclusion Validity*. By interviewing more people, also from a wider array of professional backgrounds, especially those with direct experience in app review processes and gathering more priorities and opinions, even more valuable results could be gathered, which would than mitigate this threat.

The validation process of the new criteria that were created in this theses

was done only with a subset of the full app review dataset and the reviews were labeled by only two raters. This could pose an *Threat to the Internal Validity*. This threat could be mitigated by labeling the full dataset by even more raters, so that a broader study can be done on the validity and completeness of the new criteria.

The results of the interviews could have also been influenced by the fact that most of the participants were not very familiar with the term and definition of explainability. Therefore, it might have been difficult for them to grasp the full extent of the usage of explainability. Furthermore, the use cases were also mostly new to the participants and this might also have an effect on their responses. Both of these factors result in a *Construct Validity Threat*. In order to mitigate this threat, the presentation phase of the interview included clear definitions of both explainability and the relevant use cases.

Finally, another potential threat is that not all criteria relevant to categorizing and prioritizing explainability needs in app reviews may have been identified. This would represent a *Construct Validity Threat*. While it is possible that additional criteria may exist, the ones presented here represent, to the best of current knowledge, a comprehensive range of information to support effective decision-making in categorising and prioritising explainability needs in app reviews.

Chapter 7

Conclusions

In this chapter, a summary of the overall contributions of this thesis will be presented, along with an outlook regarding the possible future works that can extend the scope of this study.

7.1 Summary

This thesis focuses on the development of a comprehensive list of criteria for categorizing and prioritizing explainability needs in app reviews. Addressing explainability in user feedback is essential for improving user satisfaction, enhancing app usability, fostering user trust in the software system and ensuring that developers can respond effectively to user concerns. To build this criteria list, several steps were taken. Firstly, an analysis of approximately 3,000 app reviews was conducted. This lead to the creation of several new criteria that were not fully covered in previous related works. These new criteria were validated through a pre-study with explainability experts. The validated criteria were added to the list which includes other elements such as metadata about the user, app, and review, as well as NLP-derived insights and external factors, such as estimated costs, for addressing user concerns.

To assess the relevance and prioritization of these criteria in real-world settings, interviews were conducted with six field experts. The experts were asked to evaluate the criteria based on predefined use cases, which were requirement elicitation, manual and automatic categorization. They provided feedback on the practical value of each criterion, offering insights into which criteria might hold the most utility depending on the specific use case. There were criteria that were deemed important for all three use cases and others that were prioritised differently based on the specific use case.

The interviewees were also asked to asses the usefulness of this study and

their feedback was largely positive, with experts confirming the potential value of these criteria in their work environments. They suggested a few extensions for future refinement, such as incorporating priority levels or user characteristics to better tailor criteria to different feedback scenarios. Additionally, several new potential applications for the criteria were identified, including ticket management and incident processing systems, highlighting the criteria's broader applicability beyond app reviews alone.

Overall, this study shows that there are different criteria that can help in different contexts regarding explainability and that a structured approach to categorising explainability needs has the potential to be helpful in addressing users issues. The study's conclusions offer a practical basis for further integrating explainability considerations into app feedback management and related fields, potentially making a significant impact on user-centered app improvement.

To summarise, the contributions of this theses were multiple. Firstly, a dataset that contained labeled reviews with explainability needs was relabeled with new labels that try to identify and summarise the exact explainability need. This lead to the creation of new criteria that were not covered in previous related works. These criteria went through several iterations of development, including several validation steps with explainability experts. Furthermore, a criteria list was developed which can help in categorising and prioritising explainability needs. This list was prioritised for different use cases by industry experts. The same experts also provided suggestions regarding potential extensions of this study, both in expanding the criteria list, as well as expanding the use cases. Finally, a subset of the initial dataset was relabeled again with some of the criteria that were prioritised most highly from the experts.

7.2 Outlook and Future works

An important avenue for future research in this area would be expanding the criteria list to accommodate a wider variety of app feedback contexts and refining it based on more diverse datasets. For instance, studies incorporating feedback from other platforms, such as social media or direct customer support channels, could reveal additional nuances in explainability needs. Integrating a more extensive NLP framework could also refine existing criteria, potentially enabling more detailed sentiment analysis and better contextual understanding of user feedback. Additionally, testing this criteria list across diverse app genres and platforms (e.g., mobile, web, desktop) would offer insights into genre-specific explainability needs, making the approach even more versatile and effective.

Another important future work lies in developing automated tools that incorporate machine learning to classify and prioritize explainability needs based on the proposed criteria. By training models on annotated datasets, researchers could automate criteria selection and ranking to improve consistency and scalability. This would create a streamlined pipeline where explainability-related reviews are immediately flagged and prioritized, saving developers time and effort. This would allow for a more efficient decision making process and would enable developers to act faster on pressing explainability issues. Future studies could also explore how user-specific data, such as user expertise or app usage frequency, might be incorporated into the prioritization process to deliver more tailored support.

Furthermore, involving a wider pool of field experts from various professional backgrounds—especially those directly involved in user support and app development could deepen the understanding of each criterion's value in different contexts. Broader industry participation could reveal additional criteria or suggest new use cases, enriching the criteria list and solidifying its relevance across industries.

Additionally, implementing this criteria-based approach in the aforementioned use cases, such as the manual and automatic categorisation and prioritisation of explainability needs or even in a ticket management system (e.g., JIRA) in the form of a plugin, could help implement this academic research in an industry context, offering developers and managers a practical tool to optimize the app feedback process while enhancing the user's experience and satisfaction.

Bibliography

[1] ISO/IEC 25010: Systems and software engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models. ISO/IEC JTC1/SC7/WG6, 2011.

[2] Wasja Brunotte, Larissa Chazette, Verena Klös, and Timo Speith. Quo vadis, explainability?–A research roadmap for Explainability Engineering. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 26–32. Springer, 2022.

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

[4] Laura Ceci. Number of apps available in leading app stores as of August 2024. `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`, 2024. [Last accessed: 28-September-2024].

[5] Larissa Chazette, Wasja Brunotte, and Timo Speith. Exploring explainability: A definition, a model, and a knowledge catalogue. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 197–208. IEEE, 2021.

[6] Larissa Chazette and Kurt Schneider. Explainability as a non-functional requirement: Challenges and recommendations. *Requirements Engineering*, 25(4):493–514, 2020.

[7] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[8] Carlo Combi, Beatrice Amico, Riccardo Bellazzi, Andreas Holzinger, Jason H. Moore, Marinka Zitnik, and John H. Holmes. A manifesto on explainability for artificial intelligence in medicine. *Artificial Intelligence in Medicine*, 133:102423, 2022.

[9] European Commission. Proposal for a Regulation of the European Parliament and of the Council laying down harmonised rules on Artificial Intelligence (Artificial Intelligence Act) and amending certain Union legislative acts. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206`, 2021.

[10] Arun Das and Paul Rad. Opportunities and challenges in Explainable Artificial Intelligence (XAI): A survey. *arXiv preprint arXiv:2006.11371*, 2020.

[11] Hannah Deters, Jakob Droste, Mathis Fechner, and Jil Klünder. Explanations on demand - A technique for eliciting the actual need for explanations. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 345–351. IEEE, 2023.

[12] Hannah Deters, Jakob Droste, and Kurt Schneider. A means to what end? Evaluating the explainability of software systems using goal-oriented heuristics. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, pages 329–338, 2023.

[13] Kupczyk Dominik. Automatisierte Detektion von Erklärungsbedarf in Nutzerfeedback zu Software. In *Master Thesis, Department Software Engineering*. Leibniz University Hannover, 2023.

[14] D Doran. What does explainable AI really mean? A new conceptualization of perspectives. *arXiv preprint arXiv:1710.00794*, 2017.

[15] Jakob Droste, Hannah Deters, Martin Obaidi, and Kurt Schneider. Explanations in everyday software systems: Towards a taxonomy for explainability needs. *arXiv preprint arXiv:2404.16644*, 2024.

[16] Standish Group. Chaos report. Technical report. 2014.

[17] Emitza Guzman and Walid Maalej. How do users like this feature? A fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162. Ieee, 2014.

[18] Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 41–44. IEEE, 2013.

[19] Maximilian A Köhl, Kevin Baum, Markus Langer, Daniel Oster, Timo Speith, and Dimitri Bohlender. Explainability as a non-functional requirement. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 363–368. IEEE, 2019.

[20] Marco Kuhrmann, Paolo Tell, Jil Klünder, Regina Hebig, Sherlock Licorish, and Stephen MacDonell. Helena stage 2 results. *ResearchGate*, 2018.

[21] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *Biometrics*, pages 159–174, 1977.

[22] Markus Langer, Daniel Oster, Timo Speith, Holger Hermanns, Lena Kästner, Eva Schmidt, Andreas Sesing, and Kevin Baum. What do we want from Explainable Artificial Intelligence (XAI)?–A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research. *Artificial Intelligence*, 296:103473, 2021.

[23] Kim Lauenroth, Erik Kamsties, and Oliver Hehlert. Do words make a difference? An empirical study on the impact of taxonomies on the classification of requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 273–282. IEEE, 2017.

[24] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *2015 IEEE 23rd International Conference on Software Engineering (RE)*, pages 116–125. IEEE, 2015.

[25] Martin Obaidi. Dataset: Gold standard dataset for explainability need detection in app reviews. In *Zenodo*. `https://doi.org/10.5281/zenodo.11522828`, September 2024.

[26] Jeroen Ooge and Katrien Verbert. Explaining artificial intelligence with tailored interactive visualisations. In *Companion Proceedings of the 27th International Conference on Intelligent User Interfaces*, pages 120–123, 2022.

[27] Dennis Pagano and Bernd Bruegge. User involvement in software evolution practice: A case study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 953–962. IEEE, 2013.

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

[29] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290, 2015.

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings*

of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1135–1144, 2016.

[31] Kai Stapel, Eric Knauss, and Kurt Schneider. Using FLOW to improve communication of requirements in globally distributed software projects. In *2009 Collaboration and Intercultural Issues on Requirements: Communication, Understanding and Softskills*, pages 5–14, 2009.

[32] Kai Stapel and Kurt Schneider. Flow-Methode-Methodenbeschreibung zur Anwendung von FLOW. *arXiv preprint arXiv:1202.5919*, 2012.

[33] Niko Tsakalakis, Sophie Stalla-Bourdillon, Trung Dong Huynh, and Luc Moreau. A taxonomy of explanations to support explainability-by-design. *arXiv preprint arXiv:2206.04438*, 2022.

[34] Max Unterbusch, Mersedeh Sadeghi, Jannik Fischbach, Martin Obaidi, and Andreas Vogelsang. Explanation needs in app reviews: Taxonomy and automated detection. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 102–111. IEEE, 2023.

[35] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology*, 85:43–59, 2017.

[36] Sira Vegas, Natalia Juristo, and Victor R Basili. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. *IEEE Transactions on Software Engineering*, 35(4):551–565, 2009.

[37] Paul Voigt and Axel Von dem Bussche. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[38] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.