Requirements Engineering (2020) 25:493–514
https://doi.org/10.1007/s00766-020-00333-1

ORIGINAL ARTICLE

Explainability as a non-functional requirement: challenges and recommendations

Larissa Chazette[1] · **Kurt Schneider**[1]

Received: 1 December 2019 / Accepted: 19 May 2020 / Published online: 15 June 2020

Abstract
Software systems are becoming increasingly complex. Their ubiquitous presence makes users more dependent on their correctness in many aspects of daily life. As a result, there is a growing need to make software systems and their decisions more comprehensible, with more transparency in software-based decision making. Transparency is therefore becoming increasingly important as a non-functional requirement. However, the abstract quality aspect of transparency needs to be better understood and related to mechanisms that can foster it. The integration of explanations into software has often been discussed as a solution to mitigate system opacity. Yet, an important first step is to understand user requirements in terms of explainable software behavior: Are users really interested in software transparency and are explanations considered an appropriate way to achieve it? We conducted a survey with 107 end users to assess their opinion on the current level of transparency in software systems and what they consider to be the main advantages and disadvantages of embedded explanations. We assess the relationship between explanations and transparency and analyze its potential impact on software quality. As explainability has become an important issue, researchers and professionals have been discussing how to deal with it in practice. While there are differences of opinion on the need for built-in explanations, understanding this concept and its impact on software is a key step for requirements engineering. Based on our research results and on the study of existing literature, we offer recommendations for the elicitation and analysis of explainability and discuss strategies for the practice.

Keywords Explainability · Software transparency · Non-functional requirements · Software quality

1 Introduction

Software systems are the primary solution for a variety of tasks today, ranging from determining the best route from A to B, to supporting a bank manager in analyzing whether a customer has a suitable profile to obtain a loan. With fast technological advancements and a wide range of new software applications, our lives have become increasingly influenced by software-supported decisions.

However, in the age of machine learning, it is often difficult to understand how outputs and decisions are computed in these systems, since the underlying algorithms can be complex and lack transparency [66]. If the decisions taken by a software system are opaque, it is difficult to understand whether these decisions are fair and what factors were taken into consideration in the system's internal decision-making process. This can potentially perpetrate injustice and bias [79]. Furthermore, the lack of transparency can potentially result in lower user acceptance and satisfaction [27]. Therefore, transparency is becoming increasingly necessary as a non-functional requirement (NFR).

The recent *General Data Protection Regulation of the European Union* [86] regulates the use of personal data by algorithms and has strengthened the debate on the right to explanations. Goodman and Flaxman discuss the impact of this law [42]. They argue that computer scientists will have to take the lead in designing algorithms and frameworks that enable explanations, since there will be an increasing demand for transparency in algorithmic decision making.

Explanations are seen as an option to mitigate the lack of transparency in a system [33]. Through explanations, the comprehensibility of a system can be improved [68]. Hence, explainability—the ability to provide

✉ Larissa Chazette
larissa.chazette@inf.uni-hannover.de

Kurt Schneider
kurt.schneider@inf.uni-hannover.de

[1] Software Engineering Group, Leibniz University Hannover, Welfengarten 1, 30167 Hannover, Germany

explanations—can be considered as a way to achieve transparency.

Incorporating explanations can help users understand why a system has delivered particular outcomes, which in effect mitigates opacity and makes decision making more transparent. It also has an impact on trust and reliance on the system [13], and it may avoid that users become frustrated with it [98].

Although explainability has been addressed as a key requirement for software-supported decisions and a means of promoting transparency [2], there is a lack of studies that investigate the relationship between explanations and transparency at the level of NFRs. Furthermore, it is not clear whether end users actually see explanations as a way to better understand a system.

Some studies investigated the impact of explanations on user experience [12, 61]. Other studies have investigated in which situations some kinds of explanations are more appropriate [62, 82]. To the best of our knowledge, there are no studies that focus on the users' opinion on the need for explanations and on the perceived impact of explanations on transparency. Since users are an essential source of requirements, it is fundamental to understand their views and what they expect from explanations for more transparent systems. Given that consumers are an important source of requirements, knowing their views and what they expect from explanations for more transparent systems is key.

The aim of this study is to investigate and understand the users' views and expectations, as these are important steps toward addressing explainability as an NFR. We also want to explore the interaction of explainability with NFRs related to transparency, and what has to be considered to meet the needs of users. This knowledge can help engineers to determine potential trade-offs, costs and implications of the integration of explanations to improve the transparency of software systems.

We addressed this by asking users about the need for explanations in applications that they use on a daily basis. We conducted an online exploratory questionnaire with 107 participants. We analyzed whether aspects of the participant response could be associated with transparency-related NFRs described in the Transparency Softgoal Interdependency Graph (SIG) [34], to understand whether and how explanations can have an impact on transparency.

We were able to find correlations between explainability and other NFRs related to transparency and also the existence of a double-edged sword effect. Explanations may have a positive impact on some NFRs that affect transparency but must be carefully designed so that they do not have the opposite effect on software quality. We offer recommendations on how to avoid the negative effects of explanations on usability.

To understand how requirements engineers can deal with explainability in practice, we searched the literature to grasp the complexity of NFRs and the related challenges. We have found that, due to their perceived complexity and project constraints, such requirements are rarely addressed during the development process. We analyze this complexity in the context of explainability and identify factors that are of paramount importance and should be considered during the elicitation and analysis of requirements that support the achievement of explainability. We refer to these requirements as *explainability requirements*. Existing lightweight activities based on user-centered design are recommended in order to support the requirements engineering process of explainability.

This paper is structured as follows: In Sect. 2, we present the definition of important terms and discuss related work. In Sect. 3, we present our research goal, the derived research questions (RQs) and the research method. In Sect. 4, we present the results and in Sect. 5 the threats to validity. In Sect. 6, we discuss the double-edged sword effect of explainability. In Sect. 7, we focus the discussion on the relationship between explainability and usability and offer recommendations on how to avoid negative effects of explanations on a system. In Sect. 8, we talk about the challenges of dealing with NFRs in the development process. Next, we describe the dimensions that must be considered in the requirements engineering process of explainability. In Sect. 9, we recommend lightweight activities for the elicitation and analysis of explainability. Finally, we conclude this paper with an overall discussion in Sect. 10, our planned next steps in Sect. 11 and our conclusions in Sect. 12.

2 Background and related work

According to Lipton [69], *transparency* can be informally defined as the opposite of opacity or blackboxness. It means "seeing through," to understand the inner mechanisms by which an algorithm works or what was learned by a model (in the case of ML-based applications).

Interpretability is a related term and can be defined as the level to which the user understands and can make use of the explanations given by the system and the information provided [95]. It is also defined as the ability to explain or to present information in understandable terms to a human [35]. The former definition overlaps with the concept of *explainability*, which can be defined as the level to which a system can provide explanations for the cause of its decisions or outputs [95]. We adopt the former definition of *interpretability*, as it is more related to the subjective aspects of how users understand the presented information.

Understandability and *interpretability* are intertwined concepts, often treated as synonyms. Therefore, we consider

that **explanations** (objective factor) are operationalizations of explainability. They can be a way of improving the understanding of a system by **conveying information**, thus influencing interpretability or understandability (subjective factor).

2.1 Transparency

Leite and Capelli [34] discussed and defined transparency in the context of system engineering, but also as a broader concept, applied to processes and organizations. The authors defined transparency as a graph of NFRs, named Transparency SIG. The graph comprises 33 softgoals arranged in three levels according to the dependency relationship between the nodes. Transparency occupies the higher level of decomposition, and the second level consists of five derived softgoals which influence directly on the satisfaction of a degree of transparency, being those: **accessibility**, **usability**, **informativeness**, **understandability** and **auditability**. We used this graph in the later iterations of the coding process to identify those requirements in the participants' discourse.

Hosseini et al. [51] divided transparency into four facets to help clarify the concept and facilitate its inclusion in the engineering of information systems: stakeholders, meaningfulness, usefulness and information quality. *Meaningfulness* is defined as how the stakeholders understand the information and what are the actions and reasons behind it. It encompasses *data transparency*, which answers **what** information is needed and **who** are the stakeholders; *process transparency* answers **how** something is performed; and *policy transparency* answers **why** an action is performed in the context of transparency.

Cysneiros et al. [27] also discussed transparency as a requirement, stating that it is considered to be a key requirement for self-driving cars. They presented transparency as a prerequisite for producing more robust systems and improving the adoption rate of new technologies. The authors also agree that being transparent to the end user is a fundamental concern and pointed out that other related NFRs (e.g., trust and privacy) are extremely important and should also be considered.

Zinovatna and Cysneiros [101] investigated the interdependencies between transparency and privacy. They established them as two intertwined concepts that need to be correctly elicited in order to determine the necessary architectural decisions.

2.2 Explanations

The idea of embedding explanations in software systems is not new and has been already intensely investigated in the domain of knowledge-based systems (KBS) [61] and by the HCI community [84]. More recently, terms like **interpretable machine learning** or **explainable artificial intelligence** have emerged and proposals aimed at improving the intelligibility of machine learning algorithms have become a trending topic [60, 85, 98].

The use of explanations in those areas typically focuses on understanding the mechanics of the learned models during decision making [59, 90], visualizing the learned model [46, 54, 74] or, in the case of KBS, on supporting users to gain knowledge of a domain [49, 82].

Bunt et al. [12] examined the comprehensibility and the desire for explanations in lightweight systems (e.g., YouTube, Amazon, Facebook). Interviews and a diary study were conducted to understand the need for explanations in the context of daily use. They concluded that the participants rarely wanted explanations on such systems. The researchers also noticed that most users understood the general idea behind the system general behavior, but had little understanding of the rationale behind more complex decisions.

Kulesza et al. [62] explored how smart agents can justify themselves to users. They analyzed how the soundness and completeness of the explanations had an effect on the mental models that the users created from the system. They then compared the mental models to the actual system model.

Tintarev and Masthof [94] evaluated the impact of explanations on effectiveness and user satisfaction, in the context of recommender systems. They also set out seven possible qualities that can be achieved through explanations (e.g., *transparency*, *scrutability*, *trust*, *effectiveness*, *persuasiveness*, *efficiency* and *satisfaction*), complementing or contradicting each other. Doshi-Velez et al. [36] discussed how explanations can help to achieve accountability in AI systems and in which cases explanations should be integrated.

In the aforementioned works, authors defined characteristics inherent to transparency in the software context and the relationship between transparency and other requirements. Some works investigated the impact of explanations on quality aspects such as acceptability, trust and effectiveness [90, 94], while others explored how aspects of explanations impact on the understandability of a system [62].

A study on the desire for explanations and their comprehensibility has been carried out by Bunt et al. [12] on the context of lightweight systems. However, besides the desire and need for explanations in the context of these systems, we additionally explore the positive and negative aspects of receiving explanations. We also investigate how explanations relate to transparency at the level of NFRs. As transparency has been increasingly addressed as fundamental in systems that support or—in the future—make decisions, understanding this concept and its related dependencies is a key step.

3 Research goal and design

We applied the goal definition template by Wohlin et al. [99], to formulate the goal of our research.

> **Goal definition:** We *analyze* end users' perspectives about the need for explanations in software *for the purpose of* investigating which non-functional requirements are impacted by explanations and how they relate to software transparency *from the point of view of* end users *in the context of* an online questionnaire.

Using the Goal–Question–Metric Paradigm [5], the goal of our study is derived into four research questions and respective metrics, organized in the goal tree in Fig. 1. We formulated **RQ1** with the goal to assess users' perspectives about the need for explanations. We wanted to understand whether users see the need for explanations in situations of uneven objectives: when users expect something from the interaction with the system, but this expectation is not met. To address this point, we provided a navigation example as a hypothetical situation. We asked users whether they would be interested to receive an explanation in the situation. To those who expressed interest, we asked for a suggestion as to what would be a useful explanation in the case. Our purpose was to understand what information users consider relevant when receiving explanations, which is essential knowledge for the operationalization of explainability. We also included a question to understand the overall need for explanations. We asked the participants when explanations should be presented: whenever requested, just when something exceptional happens or in both cases.

Through **RQ2**, our goal was to understand whether and how participants had issues with the software systems they use in their daily lives. We addressed this by asking users to report a situation where they did not understand the system behavior. Analyzing their answers helped us understand how current issues can have a negative impact on transparency and *which NFRs* can be impacted by those issues.

Through **RQ3**, we wanted to assess the perceived advantages and disadvantages of receiving explanations. This helped us to identify (1) the NFRs impacted by explanations, (2) what are the pros and cons that need to be taken into account while considering explainability and (3) whether the identified NFRs can have a positive or negative impact on the transparency of the system.

We also expected to find out, through **RQ4**, whether there are discrepancies between the perspectives of different age groups on the need for explanations. Digital natives are said to trust technologies more easily, while digital immigrants (born before 1980 [92]) may face problems while operating software systems. This could represent the explicit need for different requirements varying according to the age group.

3.1 Survey design

Based on the questions and metrics, we designed and implemented an online questionnaire using LimeSurvey. It contained 16 questions (11 multiple choice, five open-ended): three on demographics, one self-assessment question on software skills, four on software use, three on problems with software use, three on explanation needs, one on the frequency and one on the presentation of explanations.

Figure 2 summarizes the process of survey design and data analysis. We performed the initial testing by using the

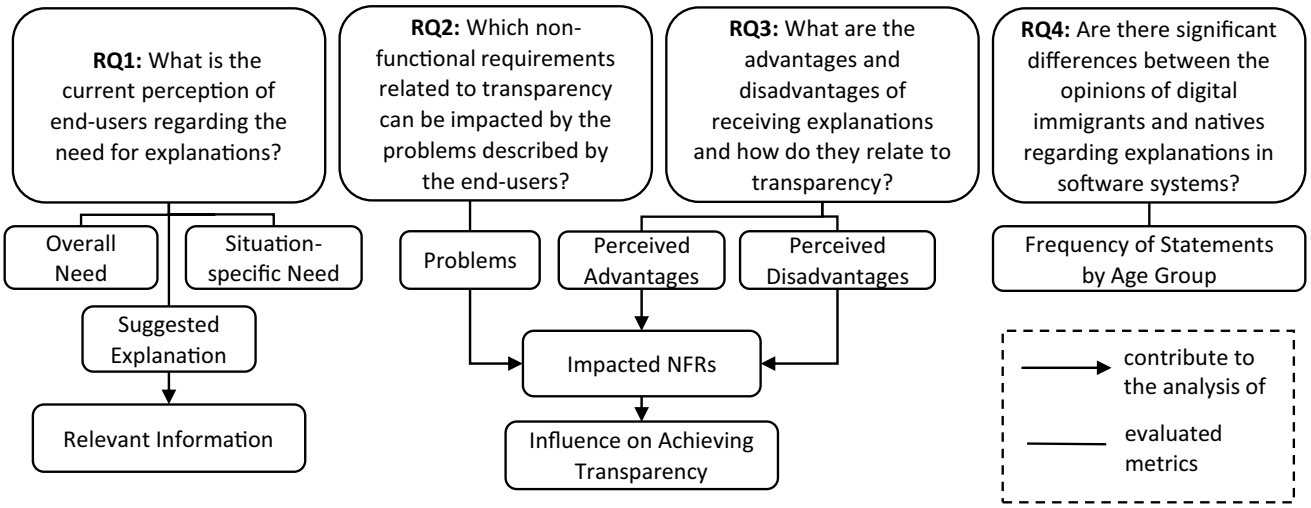

Fig. 1 Research questions and related metrics

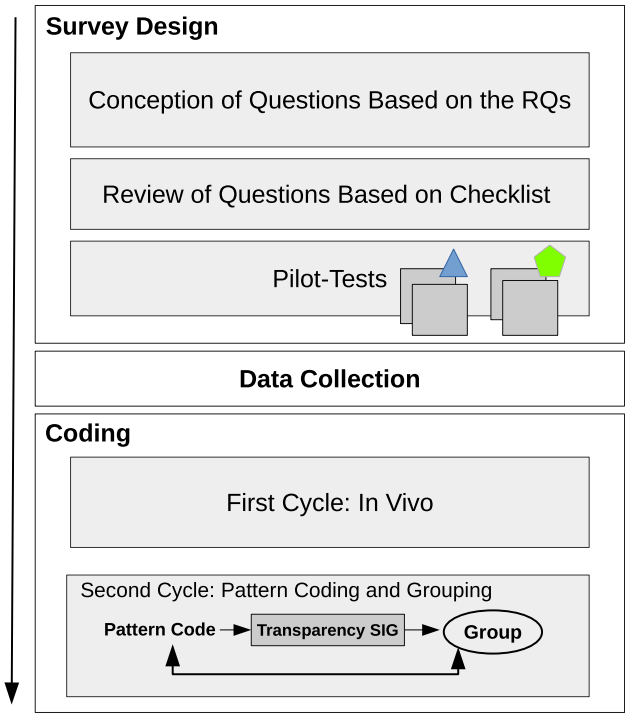

Fig. 2 Survey design and data analysis

checklist provided by Lessmann [67] to review every question. This was followed by four rounds of pilot tests: two of these with members of the target population and two with members of our research group (indicated by the different symbols in Fig. 2). In each pilot test, the respective participant completed the survey and we discussed how the questionnaire could be improved. The full instrument can be found in our online material [17].

3.2 Data collection

In late 2018, we shared the questionnaire using different channels: LinkedIn, Twitter, Facebook, and academic mailing lists. The questionnaire was publicly shared using our personal networks. We asked people to share the questionnaire with their networks in order to try to reach more people. Our target population included adult end users of all ages, with different occupations, since the focus was to understand what end users with different backgrounds have to say about the topic.

Based on our sampling strategy, we expected the main share of participants to come from Brazil and Germany. Therefore, the questionnaire was provided in three languages: Portuguese, German and English. Participants had to agree to an informed consent including a confirmation that they are at least 18 years old. An important factor during this phase was to gather a considerable and balanced number of participants to compose two different groups: the digital natives and the digital immigrants.

From the 171 that started the survey, 107 completed it. We analyzed the responses from those that completed the survey. As the qualitative questions were optional, some respondents did not provide answers to all questions. In this case, we were still able to analyze the responses to the qualitative questions that were answered in order to gain insight into the research questions.

3.3 Analysis process

We analyzed the open-ended questions using an open-coding approach, as described by Saldaña [87]. It consists of a qualitative data analysis with two consecutive coding cycles. Each cycle consisted of three phases that can be repeated iteratively. In the first coding cycle, we used *in vivo coding*, a first cycle coding method which is known as a way to preserve the views of participants in the code [16]. We have identified the key elements related to our questions in their responses. A single answer could result in more than one code, depending on its size and meaning.

In the second cycle, one researcher grouped the initial *in vivo codes* according to their similarities. Next, we applied the pattern coding approach by Miles and Huberman [73]. This approach is a way of grouping the initial codes into a smaller number of themes or constructs. Categories were created based on the main theme expressed by the codes in each group. A second researcher was involved during this phase to discuss possible meanings and reach an agreement on the themes. We tried to preserve what was said by the participants while defining the categories. Hence, although some categories seem very similar and could belong to one another, we kept them separate to preserve the original connotation.

Subsequently, the categories were classified into groups. If there were any correspondences between the category and an NFR in the Transparency SIG, the category would be identified as related to this NFR, forming a group. In the absence of a match with a quality aspect listed in the SIG, a new group would be created based on the meanings and connections of the codes. During this phase, two researchers coded the data independently into the final categories to guarantee consistency. After this, they compared the two coded datasets, discussed differences in the coding process and reached an agreement about the codes. The degree of agreement was calculated using the Cohen's kappa statistic [23]. The calculated value of $\kappa = 0.88$ indicates an almost perfect agreement [65].

4 Results

From the 107 valid responses, 90 (84.11%) came from Brazil and 17 (15.88%) from Germany. Of the participants, 45.25% were born before 1980 and, thus, classified as digital immigrants, while 54.75% were born after it, classified as digital natives.

On average, participants reported a good proficiency in the use of software systems. To assess this proficiency, we included a self-assessment question in which respondents had to indicate which of the tasks they were able to complete. It also included tasks corresponding to certain skill levels. Most respondents claimed to have a high level of proficiency, including participants who work with computers, are programmers or have programming skills. This shows that, in general, participants are comfortable with the use of software systems. This leaves out the population of unskilled users or people who face difficulties in dealing with technology. In any case, our population consists of a relevant subset of all users of complex software: technically affine end users of different age groups.

When asked whether they use software applications more for work or for personal reasons, 42.99% of the respondents affirmed they use an equal amount for both work and private life, while 42.99% responded that they use them more for work and 14.02% more in private life. This highlights the ubiquitous nature of software systems in the lives of these individuals.

4.1 RQ1: Need for explanations

4.1.1 Situation-specific need

A hypothetical situation was presented according to the device and the applications that the participant indicated as frequently used. The purpose of this question was to analyze the need for explanations in situations where there is a discrepancy between the objectives of the users and what the system presents, for example when users' expectations are not met. Therefore, it is a necessity related to a specific context: if what the user expects is different from what is presented in the system.

This premise is also discussed in the work of Doshi-Velez and Kim [35]. The authors discuss why and in which situations explanations may be necessary and helpful. They argue that this need may arise from a state of incompleteness in the information provided, where people may need explanations to minimize gaps in understanding.

In the hypothetical situation, users would use a navigation system while driving on a route they have traveled before, and the system would present them with a different route than usual. We asked participants whether they would be interested in an explanation for the route change. To analyze the situation from two different perspectives, where requirements may be different, this question had two variants: (1) one where the user would be in a vehicle using an onboard navigation system (OBNS) to guide during the drive and (2) another where the user would be a pedestrian, using public transport and depending on the navigation system in the smartphone to consult better routes and alternative transportation.

All participants answered the question with the smartphone scenario, since they indicated that they had access to this type of device. Twenty of the 107 participants also indicated using OBNS and, therefore, also answered the question corresponding to this scenario. Of the resulting 127 responses, 71.65% (91) answered that they would be interested or extremely interested in receiving an explanation about the obscure situation. There was a variation in the expressed degree of interest according to the type of device: 95% (19) of OBNS users answered they would be interested or extremely interested, while one would not be interested at all. 67.29% (72) of smartphone users answered they would be interested or extremely interested, 18.69% (20) slightly interested, 11.21% (12) indifferent and 2.80% (3) not interested at all. This may reflect that OBNS users have a more urgent need to receive explanations, strongly related to the type of system they use. While driving, users have less time to make a decision and, therefore, a more urgent need for more interpretive results.

4.1.2 Which questions should be answered by explanations

Based on the hypothetical situation presented, we asked participants to give a suggestion of what would be a good explanation in the case. We investigated whether specific elements were present in the responses, such as references to specific data (e.g., time and route), or the level of abstraction of the suggested explanation. In the coding process, we considered to which questions the identified elements matched: *what*, *why* or *how*. These questions were also present in the work of Hosseini et al. [51], mentioned in Sect. 2.1, in which the authors present three questions whose answers contribute to transparency. Eighty-seven participants completed this question, which resulted in 103 codes.

34.95% (36) of the codes refer to the *what* question. Participants expressed desire in knowing which specific piece of information supported and influenced the decision: data-related aspects contained in that information and used during decision making. Some code examples include *"which information led the software to take this decision,"* and *"which variables are influencing the choice."* Some answers referred to specific data, such as in this participant's statement: *"If*

there is some kind of incident, show me the route, time, possible accidents, etc."

In 11.65% (12) of the codes, participants refer to the *how* question. It reflects the users' desire to understand the inner reasoning process of the algorithm. Users also expressed the wish to be able to audit or verify the behavior of the system or to find out more about the internal model the system built on the user. Some examples are *"evaluate the capacity of the system of generating better routes," "to see how the algorithm detected the changes"* and *"the reason the logic changed."*

In 53.40% (55) of the codes, participants refer to the *why* question. Participants expressed willingness to understand **why** something happened, i.e., to better understand the reasons behind a decision or event, or existing policies. It usually requires knowledge about **what** data are involved and **how** the information was inferred. To understand exactly how this question must be answered, the level of abstraction of the explanation needs to be assessed. Explaining why something happened may either need a more specific answer, considering many variables **or** a very general answer, with a higher abstraction level. Some code examples are *"why the route is not being suggested"* and *"benefits of the new route when compared to the usual."*

4.1.3 Overall need

We asked the participants about when explanations should be presented. 66.36% (71) answered that explanations should be presented only on demand. This reflects that users are interested in explanations, but want to have total control about when to receive it. It also imposes a new challenge, since systems must have the ability to explain much of their behavior in order to provide explanations by request. 28% (30) answered that explanations should be shown just when something exceptional happens (e.g., in the case of mismatched objectives). 3.73% (4) answered that they would like to receive explanations in both situations (automatically, when something exceptional happens and by request), and 1.87% (2) answered that explanations should never be presented.

4.2 RQ2: Perceived problems in understanding software

To answer **RQ2**, we asked respondents to indicate whether they could remember having problems understanding the behavior of any software they previously listed as of regular use. Then, we asked whether they could report a situation when this happened.

Twenty-four participants answered the open-ended question, which resulted in 19 valid answers. We interpreted each answer and tried to identify whether they could be associated with a quality characteristic listed in the NFR framework. This association indicates whether the perceived problems could negatively impact on NFRs related to transparency. A negative impact on an interrelated NFR can result in a negative impact on transparency, since the related goals contribute to the achievement of transparency. Usability, for example, is one of the NFRs related to transparency. Hence, problems classified as impacting on user-friendliness indicate a usability problem and may require better usability engineering, in order to contribute to a higher level of software transparency.

68.42% (13) of the responses were related to the NFRs *usability*, with explicit correspondences to a perceived impact on its sub-dependencies *uniformity*, *simplicity*, *intuitiveness*, *adaptability* and *user-friendliness*. 31.58% (6) were related to *informativeness*, impacting on the sub-dependencies *clarity*, *completeness*, *correctness* and *consistency*. These sub-dependencies are NFRs that need to be fulfilled in order to achieve the requirements in the higher level. For more details about the meaning of each NFR, please consult the Transparency SIG [34].

While *usability* refers to the quality of presentation and interaction between the user and the system, *informativeness* refers explicitly to the information presented. It is already well known that usability problems may impact on the user understanding of the system and prevent the user from successfully completing a task. These problems may have negative consequences and may result in the abandonment of the use of the system [47]. Participants also reported problems while trying to obtain information from a system. They reported situations where they could not understand the information presented, being by lack of *completeness*, as evidenced in the following quote from one of the participants: *"The system was not explicit about the public transport routes, nor if it was necessary to take more than one line"* or *clarity*, as in *"I tried to identify which transport I could take. The information usually comes a bit muddled. I cannot always get the information I need."*

4.3 RQ3: Advantages and disadvantages of explanations

To answer **RQ3**, we asked participants to name three advantages and three disadvantages of receiving explanations. Ninety-one participants answered the question related to the advantages of receiving explanations, resulting in 214 valid responses. As one answer can generate more than one code (as explained in Sect. 3), their responses resulted in 231 codes. Eighty-five participants answered the question related to the disadvantages, resulting in 164 valid responses. After the coding process, 176 codes were generated from it.

Figures 3 and 4 show tree maps of the advantages and disadvantages, respectively, organized in groups and the

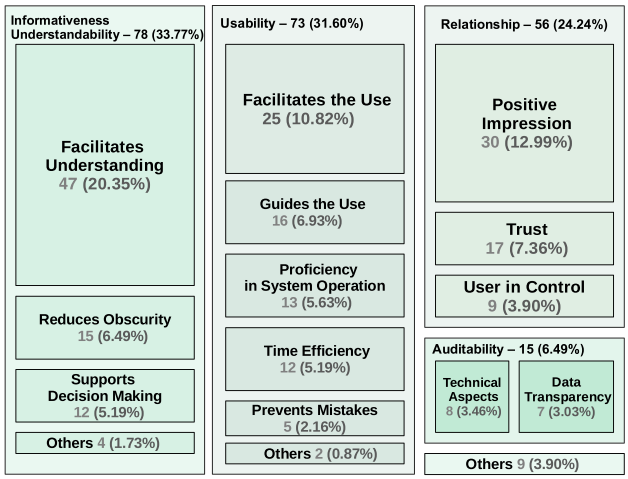

Fig. 3 Groups of categories related to the perceived advantages

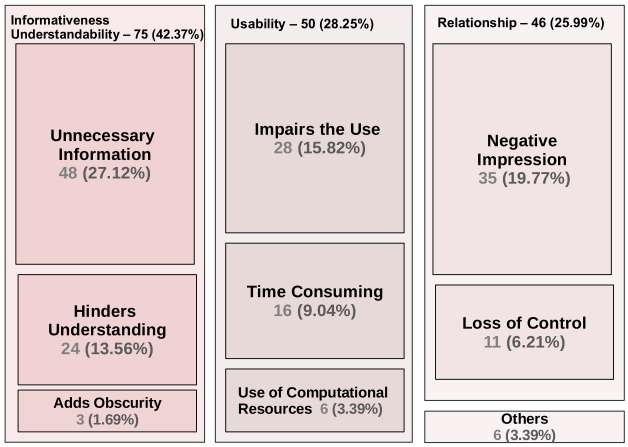

Fig. 4 Groups of categories related to the perceived disadvantages

underlying categories. Each category is followed by the number of codes and its respective percentage relative to the total number of codes.

By analyzing the participants' responses, we could identify associations with the *usability*, *informativeness*, *understandability* and *auditability* requirements in the Transparency SIG. These associations indicate how explainability can impact these NFRs. We have created the *relationship* group to gather all categories of responses in which participants expressed their personal impressions about the possibility of explanations having a negative or positive impact on their relationship with the system.

4.3.1 Informativeness and understandability

These two qualities are grouped together, because their concepts overlap at times. Informativeness can be defined as the quality of providing or conveying information to, for example, facilitate understanding. This information, however, must be correctly formulated (in comprehensible language), so it can be understandable.

Advantages 20.35% (47) of the 231 codes correspond to responses in which users perceived receiving explanations as a way to **facilitate the understanding** of a system by conveying information. This understanding can be either specific, related to the current situation, or a piece of data, or more general, related to the whole context of the system. Some sample quotes are: (understanding) *"how the software works"* and *"what is being shown."*

6.49% (15) of the codes correspond to responses in which users perceive explanations as a way to **reduce obscurity or clarify doubts**. This category encompasses responses where participants explicitly see explanations as a way of mitigating system's obscurity, providing clearer information. 5.19% (12) of the codes express users' beliefs that explanations may **support during decision making**. Both codes could be identified in this participant's statement: *"(Explanations) allow my decisions to be made on the basis of clear information."*

1.73% (4) of the responses refer to other codes, including those that explicitly mention information as an advantage. The aforementioned results show how users believe that explanations can convey information better and lead to an overall better understanding of different software aspects. Explanations may have a positive influence on the *informativeness* of a system and, therefore, on its transparency level.

Disadvantages 13.56% (24) of the 177 codes are related to the concern that explanations may actually, rather than facilitate, **hinder understanding**. This may be the case, if explanations are not provided in a language appropriate to the user or are poorly elaborated. This can be noted in the following statement: *"If the explanation comes in a very technical language, the user may not understand it."*

27.12% (48) of the codes express the users' belief that explanations can actually only bring **unnecessary information**. They affirmed that explanations may be too lengthy, repetitive, irrelevant or useless. One participant stated: *"Explaining what is already known makes information boring and irrelevant."* 1.69% (3) express users' concerns about explanations **failing to reduce obscurity** or even adding more. Both categories impact directly on the understandability, since information must be concise and comprehensible enough in order to be well understood.

4.3.2 Usability

Advantages 10.82% (25) of the 231 codes comprise responses in which participants considered receiving explanations as a way to **facilitate the use** of a system. They also see explanations as a way to support them to better operate

the system. Some quote examples are: *"better operation"* and *"increases the usability of the device."*

6.93% (16) evidence the users' belief that explanations are a way to **guide the use** of a system, enabling faster familiarization or working as a tutorial. 5.63% (13) referred to the possibility that explanations help the user to become **proficient in the operation of the system**, knowing all available features and mastering its operation. 5.19% (12) express the belief that explanations may support **time efficiency**, assisting the user in making faster decisions or having more agility while operating the system. 2.16% (5) refer to the possibility that explanations may be a way to **prevent users from making mistakes**, supporting them during decision-making situations. Other responses (0.87%) mention simplicity as an advantage and affirm that explanations may help users to know that the system is working properly.

Disadvantages In 15.82% (28), participants expressed worry about explanations **impairing the use** of a system. Users were concerned that the UI becomes polluted with the excess of explanations or notifications, with the interruption of the workflow, and with explanations being too distracting. 3.39% (6) express concern with the **use of computational resources** when incorporating explanations into a system, consuming storage space, memory and CPU resources, or data volume. 9.04% (16) refer to the possibility that receiving explanations may be **time consuming**, as users may have to invest time to consume explanations. Some participants also expressed the opinion that explanations may be a waste of time.

4.3.3 Relationship

Advantages 12.99% (30) of the responses express the **positive impression** of the participants regarding explanations in a system. Some state that explanations improve the experience of using a system and avoid frustrations. 7.36% (17) state that receiving explanations may help to **establish a relationship of trust** with the user. A participant affirmed that explanations may help to *"increase confidence in software and its developers."* 3.90% (9) expressed the users' view of explanations as a way to **put the user in control**.

This advocates for the positive impact of explanations on the relationship with the system. By providing explanations, users may feel more comfortable and satisfied. Also, by disclosing the reasoning behind a decision, explanations can be used to increase trust in the system. Some participants affirmed that explanations allow them to decide whether the system decision can be accepted. The responses also indicated the desire of the participants to have control of the system (*locus of control*). This is a phenomenon in psychology [40] and is a factor considered by usability designers to ensure this sense of control to the end users. It is also an important aspect on the human–computer interaction that impacts on the perceived quality of the system [77].

Disadvantages 19.77% (35) of the responses address the **negative impression** of the participants about receiving explanations in a system. In this case, participants expressed concerns about explanations being annoying, inconvenient, tiring or boring. 6.21%(11) of responses state that receiving explanations may result in **loss of control**.

Explanations may have a negative impact on the relationship with the users and trigger negative feelings (e.g., feeling annoyed). Users may feel uncomfortable while receiving explanations, and a bad relationship with the system may cause them to abandon the use. Participants also perceive as negative if they do not have the option to disable explanations when they are not desired. This can be observed in the following statement: *"It would be interesting if I could request the explanation just when I wanted. If it is not requested, makes it inconvenient."* This conclusion is in accordance with the feeling of being in control, previously discussed. It is also supported by the findings presented in Sect. 4.1.3, when participants expressed the desire to receive explanations whenever they request them.

Auditability 3.46% (8) of responses address the possibility of explanations leading to a better understanding of the **technical aspects of the system**. This category includes answers in which participants relate explanations with the ability to understand the internal process of the algorithm, as well as being a way to check its behavior. They also consider explanations as a way to find out more about the internal model that the system has created of the user. 3.03% (7) related explanations to more **data transparency**.

Auditability is the group with less answers, which can indicate that end users may not be so interested in knowing specific details about the inner workings of systems. From the 11 responses, seven are related to data transparency. Participants show a level of concern about what is happening with their data. To mitigate this, explanations may be a way to inform users about how their data are being processed and for what purpose they are being collected. In 3.90% (9) of the responses, participants explicitly mentioned **transparency** as an advantage. This is, once more, a clear indication of the influence of explainability on achieving transparency.

4.4 RQ4: Perception according to age group

While digital natives grew up in the digital age and interact with digital systems since their childhood, digital immigrants acquired the familiarity with digital systems in adulthood. Digital natives are suggested to have experienced a specific technological socialization shaped by a distinct ICT environment, which can be assumed to result in characteristic cognitive and behavioral patterns [83]. Supporters of this concept believe that they relate to technology in a different

way than the earlier generation. They are also said to be more comfortable with new technologies and have a stronger tendency to trust them easier [50]. To understand whether those generational differences were found to be related with different opinions and expectations regarding explanations, the following hypothesis were formulated:

$H1_0$: There is no difference in the answers with respect to the advantages of explanations between digital natives and digital immigrants.

$H2_0$: There is no difference in the answers with respect to the disadvantages of explanations between digital natives and digital immigrants.

With regard to the advantages and disadvantages of explanations, no statistically significant differences were identified between the two groups [18]. Nor have we been able to find any significant differences with regard to a different impact on the relationship with the user or any of the other quality aspects. However, the respondents of this questionnaire had a high degree of technical knowledge, which may have had a considerable impact on the results. The population of unskilled users or people facing difficulties in dealing with technology may have different requirements in terms of explainability.

5 Limitations and threats to validity

The participant selection strategy resulted in some limitations. The questionnaire was distributed online in order to reach as many potential respondents as possible. However, responses of the reached participants may not reflect the needs and perceptions of the whole population. We only got responses from Germany and Brazil which also threatens the global generalizability of our findings. Another issue is that participants likely have a good level of technological literacy to answer an online questionnaire. This was confirmed by the findings in Sect. 4, where respondents affirmed to have a good technological proficiency level. Therefore, they may not represent people who face difficulties when using software systems and are likely to have different needs for explanations.

In view of these facts, our sample does not represent the population of Brazil or of Germany as a whole. It is rather a slice of the population of technologically literate end users. This may result in a limitation of the generalization of the results. Although we do not claim generalizability of our findings beyond the group of participants, they represent the perspectives of a part of this population, which may give us a hint about the overall perspective.

Some of our conclusions also might have been affected by limitations due to a small sample size. Although over a hundred participants provided a substantial body of responses, an even higher number could have led to more reliable results. However, our analysis process included coding which is labor intensive. Thus, we consider that 107 participants from the target population are a good start to exploring the field.

Further studies should be conducted to explore the need for explanations in the context of people with less or no technological literacy. Nevertheless, even with a high level of technological knowledge, the participants still expressed the need for explanations.

With respect to RQ2, we could identify some of the NFRs affected by the problems described, but we cannot be sure that there are no other NFRs affected. This can be due to a mono-operation bias. It is not possible to identify all potential impacted NFRs only by assessing the problems that participants had with daily applications. Further studies should also consider the impact of different contexts on the relationships with other NFRs.

The use of a questionnaire as an instrument, as well as the use of qualitative analysis, causes a mono-method bias. The impact of the questionnaire on this bias is that it is a single source of data and allows only a limited explanation of our findings. Further experiments need to be conducted, where the same questions can be evaluated in the context of a deployed system. Regarding the qualitative analysis, its subjective nature may have affected the findings. To mitigate this, we used *in vivo coding* to adhere closely to the language of the respondents. In addition, during the second cycle, two researchers coded the data into the final categories to ensure consistency. Both coders compared and discussed the results of their coding process in order to reach an agreement on the assigned codes, thus increasing the reliability of the findings.

Another potential threat was the use of a hypothetical scenario to ask questions. It may have produced responses that do not match the behavior of people who would be in the same situation in a real-world setting. We tried to minimize this by making sure that the questions were high in psychological realism: The participants confronted a situation they would likely experience in their everyday lives [6]. Since we wanted to understand the needs of the average end user, we focused on common everyday applications. Hence, another concern is that the findings might not reflect the effect of explanations in more sensitive scenarios (e.g., using software recommendation to make business decisions or decisions with critical/ethical consequences) due to the kind of applications we discussed. Indeed, the results cannot be generalized to the complex context of such systems. However, the fact that explanations are desired even in the context of lightweight systems may indicate that they can also be useful in more complex contexts.

Good question wording and instrumentation layout are also crucial for the results of a survey. We followed

guidelines and conducted pilot tests to ensure these aspects. Yet, the order of questions in the questionnaire may have had an impact on the participants' understanding of whether we were asking questions about the need to receive explanations in a general context or related to the more specific contexts of previous questions. We acknowledged, however, that this might be useful for participants who may have trouble imagining other scenarios in which they might need explanations.

6 The double-edged sword effect of explainability

One aspect that conveys the complexity and the challenges of dealing with NFRs is that they can be *interacting*. This means that the attempts to achieve one NFR can hurt or help the achievement of another [22]. In our study, we could identify how explainability interacts with other quality attributes related to transparency. Explainability can both help or hurt the achievement of other important NFRs, indicating the existence of a double-edged sword effect.

By investigating RQ2 and RQ3, we found that explanations may have an impact on NFRs related to transparency. We identified impacts on *usability*, *informativeness*, *understandability* and *auditability*. These requirements have an impact on the level of transparency, demonstrating that explainability and transparency are related. By investigating **RQ2**, we could assess how problems with understanding the behavior of simple software systems may have an impact on system transparency.

We could also understand how shortcomings in some NFRs (in this case, *usability* and *informativeness*) may have an impact on the user's understanding of the system. On the positive side, explanations may potentially help to solve these shortcomings. However, if not correctly elicited and analyzed, explanations may have a negative impact on the same quality aspects. The relationship between the user and the system, which relates mainly to feelings of control and trust, can also be affected either positively or negatively by explanations.

Explainability was perceived as a way of achieving *informativeness*, by conveying more information about aspects of the system. By considering explainability in a system, it is possible to provide a better level of interpretability, **facilitating the understanding** of the system or the current scenario. This is due to the fact that users are given more information about the system and its outcomes. Consequently, users may feel that they make more conscious decisions, since they can better understand what is happening.

Explanations may also help to improve the *usability* of the system, **facilitating the use** and teaching the user how to better operate it. They can **guide the user** during the use of the system, working as a tutorial, to introduce the software features. They can also help users **get acquainted with the system**, helping when they are stuck in a situation or to have a better understanding of all features available. Explanations may also help users to be more **time efficient**, accomplishing tasks faster.

The *auditability* of the system may also benefit from explanations, especially with regard to its **technical aspects** and **data transparency**. In the case of data transparency, participants expressed a desire to know how their data are used and how it contributes to algorithmic intelligence.

At the same time that explanations can be an advantage that facilitates the understanding of the system, they can also have the opposite effect if they are not correctly designed. Results have shown that explanations can, in contrast, **hinder understanding** if they are not displayed in a language that meets user's needs and expectations.

This highlights the need for attention when considering including explanations in a system. It is always necessary to consider the target users and which language is most appropriate. Otherwise, explanations may **add more obscurity** to the understanding of the information, instead of helping to mitigate it. Responses also pointed to the possibility of receiving **unnecessary information** as a disadvantage. This indicates that software engineers must pay attention to what is to be explained and if the user really needs to receive an explanation about it.

Appropriate design choices regarding explainability need to be made, so that explanations do not pollute the interface and harm the experience. Users expressed concerns about whether **the use of resources** such as CPU, memory, storage space, data volume and battery could be impaired by this additional feature. Software engineers must also pay attention to how the explanations can be integrated without compromising the performance of the system. Participants also expressed concern that they may need **too much time** to read and understand the explanations. This is antagonistic to the identified advantages of time efficiency, indicating that explanations shall not be time consuming.

On this basis, explainability clearly has a **double-edged sword effect**. It can act both as a synergistic NFR that contributes to the achievement of other NFRs related to transparency and as an antagonistic requirement. While considering the integration of explanations in a system, the goal may be to add transparency, facilitating the system use and its understanding, but it may result in the opposite effects. The outcome will strongly depend on the design choices during requirements analysis.

This highlights the need for a careful requirements analysis. A key step toward a successful outcome is the identification of the interdependencies among NFRs, assessing the relationship between explainability and other requirements, potential conflicts and trade-offs.

Having conflicts between NFRs means that fulfilling one requirement can affect another's achievement. The conflict between usability and security is often pointed as a classic example of such conflicts [25, 45, 57]. A system module may require security mechanisms, which may increase its complexity and, consequently, make the interaction with the system more complex [14].

Mairiza and Zowghi [70] identified three kinds of conflicts between NFRs: *absolute conflict*, when two NFRs are always in conflict; *relative conflict*, when a pair of NFRs are sometimes in conflict depending on factors such as stakeholders agreement and the architectural decision to operationalize the NFR; and *never conflict*, when a pair of NFRs never conflict.

Explainability was not included in the analysis in the study mentioned above. In our study, we were able to identify that the double-edged sword effect of explainability suggests a relative conflict. The positive or negative effect depends on how explainability is refined to more fine-grained requirements and how they interact with other NFRs.

We can notice this phenomenon when participants point to the advantages and disadvantages of receiving explanations. Taking the case of *usability*, explanations will either have a positive effect by supporting user-friendliness, guiding the user during the use of the system and supporting the user in achieving proficiency in the system operation **or** have the complete opposite effect and impair the use by providing too many notifications. This will depend on how the explanations are designed and operationalized and how this design is consistent with established usability principles.

Although it is certainly rewarding to investigate all interactions of explainability with other NFRs, this would go beyond the scope of a single publication. Since usability is an essential aspect of software systems and fundamental for user satisfaction, in Sec. 7 we explore the relationship between explainability and usability in more details. We also outline recommendations for the design of explanations, based on state-of-the-art heuristics, to avoid a negative impact of explainability on a system.

7 Usability through and despite explanations

There was a time when usability was a secondary concern: when computers were so expensive and used by only a small amount of people who mostly performed very specialized tasks. The popularization of computers shifted this perception, as then all sorts of consumers had access to personal computers. Nowadays, user interfaces are a major way to differentiate products in the market and it is what adds value to a software product [75].

Usability is also rated as one of the most important NFRs by practitioners [4]. In an empirical study which investigated the importance of quality requirements in the industry, usability was among the top five in importance for product managers, project leaders and developers, for all types of projects in various domains [31]. According to a study by Groen et al. [43], it was one of the most frequently identified software qualities in the feedback of users from app stores.

Usability also has an effect on trust, since it favors a better comprehension of the contents and tasks that the consumer must realize to achieve a goal and reduces the likelihood of error [38]. According to the ISO/IEC 25010 [53], trust is perceived as a key pillar to the user satisfaction levels and it has a strong impact in the continuity of use.

Explainability is linked to both transparency and usability. Explanations may help improve the usability of the system and increase the level of trust, impacting on user satisfaction. Yet, not every kind of software should provide explanations to make its usability clear. Good usability design is about having intuitive software, without the need for explicit explanations on how to use it. However, in situations where the system is too complex, explanations can be a good way to mitigate the complexity of the system and help the user to better operate it. On the one hand, the higher the usability or transparency of a system, the less the explanation is needed. On the other hand, well-designed explanations may increase the usability of highly complex or opaque systems.

We analyzed the participants' answers on the disadvantages of explanations and selected the negative characteristics that they identified. Our goal was to understand what were the most commonly identified issues and whether they could be prevented by applying principles of usability engineering.

We found out that, by using well-known usability heuristics, as suggested by Nielsen [75], most of the negative effects reported by the participants can be prevented. This finding endorses two of our assumptions: (1) the close relationship between explainability and usability and (2) how state-of-the-art heuristics can prevent the possible negative effects of explanations in a system, without the need for unknown new methods.

We summarize the findings in Table 1. We list the perceived negative aspects corresponding to each class, as described in the participants' answers. Then, we link the issue to a usability heuristic that can mitigate the negative effects.

Usability Heuristic "Simple and Natural Dialogue": The participants mentioned both the possible negative characteristics of the explanations and the undesirable aspects of the information provided by the explanations. Explanations may be repetitive, unnecessary, inopportune, long and not objective, while information may be redundant, useless, irrelevant and excessive.

Table 1 Negative effects identified in participants' answers and the correspondent usability heuristics that may mitigate them, following Nielsen's usability principles [75]

		Negative effect	Usability heuristic [75]
Informativeness/ understandability	Unnecessary information	Explanation: repetitive, unnecessary, inopportune, lengthy, lack of objectivity	Simple and natural dialogue
		Information: unnecessary, excessive, redundant, useless, irrelevant	
	Hinders understanding	Confusing, misinformation	Speak the users' language
		Language: difficult, complicated, technical, incomprehensible	
	Adds obscurity	Vague, obscure, unclear, ambiguous	
Usability	**Impairs the use**	Distractive, lack of focus	Simple and natural dialogue
		Disrupts flow, interruption	
		Excessive information on screen, polluted UI	
	Time consuming	Loss of time, time consuming	
		Decreases dynamism and speed	

These characteristics are contrary to the idea of presenting only **essential content** to the user, as recommended by the principle of *simple and natural dialogue*. According to this principle, user interfaces should be simplified as much as possible presenting exactly the information the user needs, when it is needed.

One of the elements included in this principle is the concept of *less is more*. This concept recommends to identify the information that is really important for the user and that will help the user to perform the task, avoiding unnecessary extra information.

Participants also pointed to some aspects that may negatively impact usability. They mentioned that receiving explanations may be distractive, interruptive, or pollute the UI with excessive notifications. In this case, the interface should be kept as clean as possible, avoiding cluttered information.

The amount of information will also have an influence on user performance. Any piece of information is something that users will have to look at while navigating, so their performance will be slowed down and they can perceive it as being time consuming.

Usability Heuristic "Speak the Users' Language": Participants also mentioned negative aspects of explanations that may hinder understanding. In their perspective, the information presented may lead to confusion or misinformation, while the language in which the information is presented may be too difficult, complicated, technical or incomprehensible. These aspects suggest that the language used is not an **accessible language**, something that is recommended by the usability principle of *speaking users' language*.

The identification of an appropriate vocabulary for the interface must take into account aspects such as the needs and expectations of different users, cultural factors and the vocabulary used in the domain.

Following this heuristic, it is necessary to avoid the use of technical terms that may be unfamiliar to the user and to pay attention to how the explanation is designed so that there is no ambiguity or vagueness.

8 The dimensions shaping explainability

NFRs are not mere descriptions of the quality characteristics of the system. They are central to understanding how these quality characteristics translate into functional requirements and constraints that must prevail and are fundamental aspects for the design of a system [64]. NFRs are difficult to fix later on in a project and should be considered from the start. According to Cysneiros et al. [26], dealing with NFRs from the very beginning of software development and integrating this knowledge with functional conceptual models lead to cost savings and to higher customer satisfaction.

NFRs, however, are traditionally a difficult topic due to their fuzzy nature and trigger debates among requirements engineers about their meaning and scope [41]. NFRs have three main aspects that transmit their complexity: They can (1) be **subjective in nature**, since some solutions to NFRs may be considered accomplished by some people, but not by others; (2) be **relative in nature**, since the degree to which they are perceived as met also varies according to the person and the context; (3) be **interacting**, since the attempts to achieve one NFR can hurt or help the achievement of another [22].

In addition to these challenges, factors spread across different dimensions need to be considered during the requirements engineering process. As explainability is still an emerging requirement, there is no structured knowledge of which factors should be considered during the analysis of this NFR. Our goal is to convey knowledge that facilitates the elicitation and analysis of explainability. We present the factors that will either influence the consideration of

explainability as a necessary NFR within a system, or the design choices toward its operationalization.

In order to identify these factors and the challenges surrounding NFRs, we manually reviewed the existing literature by searching for papers on NFRs published in two key requirements engineering sources: the *Requirements Engineering Journal* and the proceedings of the *IEEE Requirements Engineering Conference*. For a better coverage, we also looked for papers of interest in the listed references. We combined the findings of this search with the findings of our survey and the knowledge found in the explainability literature.

As a result, we present below the dimensions that affect the elicitation and analysis of explainability. Figure 5 illustrates these dimensions.

8.1 Users' needs and expectations

The first factor is to consider the users' needs and expectations with regard to explanations. Different groups of users will certainly have different expectations, experiences with technical systems, personal values, preferences and needs. Such aspects also mean that individuals can perceive quality differently.

As part of **RQ1**, we investigated the need for explanations in both a general and a more specific context, in situations of uneven objectives. The participants expressed interest in receiving explanations in both of the contexts under investigation. In a situation of uneven objectives, where their expectations are not met, most of the participants expressed interest in receiving an explanation. The difference between navigation on smartphones and on OBNS also had an impact on the answers, suggesting that the need for explanations varies according to the context in which the user is using the application.

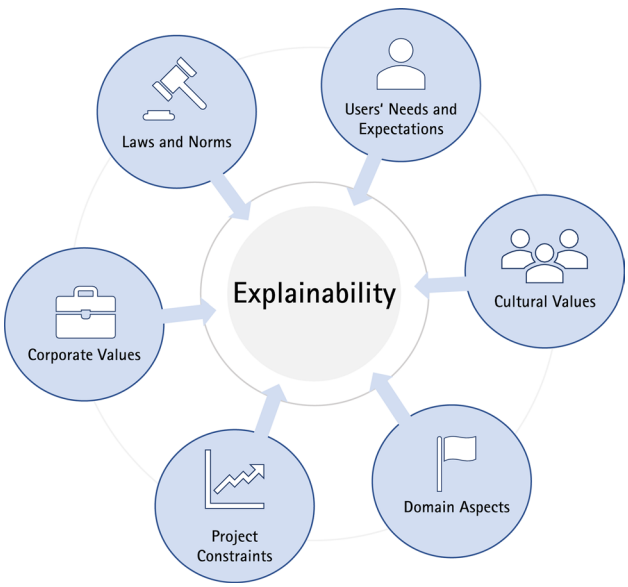

Fig. 5 Different dimensions shaping the analysis and operationalization of explainability

Users' individual needs and expectations may also have an effect on the need for the granularity of the information to be provided. With regard to the questions to be answered, the respondents were especially interested in getting answers to **what** and **why** questions. These answers have a higher granularity level than answers to **how** questions, which give more details about the system's inner reasoning process.

Based on our findings, we assume that answers to **what** and **why** questions might be more important when presenting explanations to the non-expert user. Knowing specifics of the internal workings cannot be so desirable, which certain stakeholders might perceive as a positive outcome. This is because some companies might be reluctant to disclose information on lower-level rationale to the public as it could harm their competitive advantage [20].

Generational differences are an example of an aspect which could impose different requirements. However, by investigating **RQ4**, we did not find any significant variations in the perspectives of the different generations of participants on explainability.

In addition, negotiating requirements is not only about considering and balancing the individual expectations of the stakeholders, but also balancing those expectations with a number of other factors: cultural values, laws and norms, corporate values, domain aspects, and more practical project constraints such as time and budget [30, 78]. Rich elicitation approaches (e.g., empirical methods) can be used to understand social and human factors as well as context-dependent aspects that will later on be translated into more specific requirements.

8.2 Cultural values

Culture is the collective mindset that distinguishes the members of one group of people from another. Different cultures require different types of information, process it differently and require different designs of information systems [19]. Systems are influenced by the respective national environments in which they are deployed, such that the respective culture strongly influences their design [63].

Cultural values refer to the *ethos* of a group or society [80]. They influence the need for a given system quality and how it should be operationalized [63]. The European *Ethics Guidelines for Trustworthy AI* [37] are an example of values that represent the common vision of a group. Such values vary between cultures. For instance, while some cultures are more concerned with data privacy and the ethics of software systems, the largest proportion of internet users in some emerging economies claim to trust the internet [1]. Hence, in

such cultures the need for explanations and what is expected in terms of explanations can be perceived differently.

8.3 Corporate values

Corporate values refer to the strategic vision and values of the organization [93]. With the omnipresence of software systems and the age of artificial intelligence, the importance of integrating human values into software becomes more evident. Organizations have made considerable efforts to define their public values statements, including values such as corporate integrity, respect and honesty [97].

However, integrating such values in the systems we produce is not a task embedded in the everyday routine of a project. As Whittle [97] observes, even in cases in which companies consider values during software development, the approach is limited to creating a values-driven culture instead of having it integrated in software.

While explainability can be a means of providing greater transparency in software systems, if there are no clear laws requiring the company to satisfice a certain level of explainability, having explanation as an NFR would depend on the company's own values and interests. In the case of self-driving cars, according to Cysneiros et al. [27], different car manufacturers are likely to prioritize NFRs according to specific selling points of their brands. For instance, the decision of satisficing explainability within a system may stem from a corporate interest in achieving more transparency, to improve the users' trust in the system, or to provide a better user experience.

In Sect. 6, we saw that explainability can help the auditability of a system by conveying information about its inner working or technical aspects. In this case, software companies must weigh how much information they are willing to disclose about the system's internal behavior. This is necessary in order to balance the needs of users with the disclosure of valuable proprietary information and trade secrets.

8.4 Laws and norms

Cultural values resonate in the conception of laws and norms. The European Union has adopted a specific data protection legislation (GDPR) to protect citizens' data. A certain degree of explainability is also required under this law. Other countries either have their own separate data protection law or no specific legislation.

Laws and norms may impose constraints that have to be met despite the corporate values. Software systems are required to comply with them or face sanctions. These laws, regulations and policies need to be analyzed and accommodated during the definition of requirements for a new system [91]. They have a direct impact on how companies prioritizes qualities and can influence the system architecture [11].

8.5 Domain aspects

Domain aspects are an essential aspect on the analysis of NFRs, since needs change depending on the domain [21]. Each software domain has quality characteristics that are of particular importance and may deserve more attention. The relative priority among NFRs may change as the characteristics of the environment in which the system operates differ [96].

Domain aspects dictate whether explanations are more urgent and how they should be designed. Explainability in some areas may be an optional requirement, focusing on enhancing user experience, while in others it may be a fundamental quality. The degree and extent to which an explanation is needed to support or justify a decision vary according to the criticality of the domain.

Critical systems impose different sets of NFRs when compared to non-critical ones. In more critical domains such as medical, financial and autonomous systems, explainability may be more urgent. For instance, in the case of a simple navigation system, explanations that inform the user about the chosen route or the causes of a route change can be viewed as an additional feature that impacts on user experience and product satisfaction. However, in the case of systems that support medical diagnosis, the lack of explanations of the reasons for a given diagnosis can have a much more dramatic impact, with ethical consequences [79].

8.6 Project constraints

Project constraints are more practical aspects (also known as *non-technical aspects* [15]), such as available resources (e.g., time, money, technologies, manpower). Such aspects have a strong influence during requirements engineering and may take precedence over others [4].

The project constraints have to be considered during the elicitation and analysis of explainability. In the end, all dimensions need to be balanced considering these non-technical aspects. It is quite idealistic to claim that we must consider all dimensions, without bearing in mind that every project faces resources limitations that must be balanced against the desired quality.

The need for explainability and the effort involved in meeting it must be balanced with what it represents for the company in terms of resources. Excessive quality can lead to unnecessary costly design of the software system, unnecessary use of the resources needed to operate the system and trade-offs where other important attributes are negatively affected [39].

9 Recommendations: user-centered design for explainability

Developing software-based systems that can be operated intuitively is a fundamental concern of software engineering and human–computer interaction (HCI). Seffah et al. [88] discuss human-centered software engineering, where the focus in system development shifts toward putting the goals, needs and wishes of the users in the first place. The idea of human-centered software engineering is to implement some of the techniques used in human-centered design (HCD).

The HCD philosophy encompasses user-centered design (UCD) principles. In Sect. 7, we have discussed how heuristics based on well-known usability principles can avoid some of the possible negative effects of explanations. These heuristics are commonly used during usability engineering and are based on UCD principles. Due to the close relationship between usability and explainability, we propose that the elicitation, analysis and design of explainability be integrated with usability engineering to avoid the undesired negative effects of explanations.

We encourage the use of UCD techniques, since usability engineering (UE) should always be incorporated into any software project in order to favor a good user experience. However, we are also aware that many software development teams are still underusing these techniques [89]. Software and UE professionals sometimes find it difficult to define whether and why certain UE tools and methods are better suited in a specific development context than others [88].

Another reason for encouraging this is the challenge of aligning research and practice. This challenge is familiar to researchers, since it is often difficult that practitioners find a way to integrate research proposals in the busy day-to-day life of the industry [5]. There are indications that the methods and techniques coming from the research community have been rarely adopted by practitioners [4].

As researchers, we have to take action in investigating methods and techniques which are aligned with the practice and offer advantages for practitioners instead of more overhead. Therefore, we propose to concentrate on well-investigated concepts and user-centered approaches, instead of conceiving a complete new process with unfamiliar activities for the engineering of explainability. In view of this, we suggest lightweight UCD activities which do not add too much overhead into the process.

9.1 A short summary of UCD

UCD is a multidisciplinary design approach to interactive systems development that aims to make systems usable and useful by focusing on the users, their needs and requirements. It considers human factors and uses the knowledge and techniques of usability engineering, being an effective approach to overcome the limitations of traditional system-centered design [71]. UCD increases effectiveness and efficiency and enhances human well-being and user satisfaction. It also counteracts possible adverse effects of use on human health, safety and performance [52].

Seffah and Metzker [89] discuss ways of filling the existing gap between software and usability engineering practices. This gap primarily concerns the non-involvement of usability experts in the process and the lack of user-centered practices within software development. By integrating more user-centered practices, there is a shift of focus in systems development toward putting the goals, needs and wishes of the users in the first place. Established UCD best practices and UE methods should be a core part of every software development activity [88]. Hehn and Uebernickel argue that usability engineering should be integrated into the requirements engineering process, combining the human-oriented aspect of the former with the more formal, technology-driven aspect of the latter [48].

Table 2 summarizes the main differences between the traditional and user-centered practices. Traditional practices focus on the system itself and its features, while user-centered practices focus on understanding how the humans using the system perform their tasks and how the system can support them.

Table 2 Traditional practices in comparison with human-centered practices, adapted and extracted from Seffah and Metzker [89]

Traditional practices	User-centered practices
Technology/developer-driven	User-driven
System component focus	User solution focus
Individual contribution	Multidisciplinary teamwork
Focus on internal architecture	Focus on external attributes
Product quality	Quality in use
Implementation prior to human validation	Implementation based on user-validated feedback
Establishing the functional requirements	Understanding the context of use

We considered existing UCD practices and selected a set of essential activities that can be useful in the elicitation and analysis of explainability requirements. We call these activities essential, because we believe they are the basic set of activities that are needed for a meaningful design of explainability. We decided to focus on activities rather than on the process itself, as this enables activities to be integrated into different development processes (e.g., traditional and agile) and gives our recommendations more flexibility. Furthermore, as these activities are part of usability engineering, the elicitation and analysis of explainability can be merged with the usability engineering tasks, if existing.

According to Cooper et al. [24], UCD activities roughly consist of:

- Understanding users' desires, needs, motivations and contexts.
- Understanding business, technical and domain opportunities, requirements and constraints.
- Using this knowledge as a foundation for plans to create products whose form, content and behavior are useful, usable and desirable, as well as economically viable and technically feasible.

These activities allow the analysis of all the dimensions necessary for the proper design of explainability (Fig. 5): laws and norms, users' needs and expectations, domain aspects, corporate values, cultural values and project constraints.

We consider the specific activities of two existing usability engineering processes in order to build up our proposed essential set: The goal-directed design process proposed by Cooper et al. [24] and the usability engineering lifecycle by Mayhew [72]. We combine them with classic requirements engineering activities as described by Alexander and Beus-Dukic [3].

The activities in our set are organized in four groups that can be iteratively performed. We start by exploring how qualitative techniques can be used to discover the NFRs within the system and how they support **discovering the need for explainability**. After the need for explainability is identified, user research needs to be carried out where aspects about users, context and domain are better understood and **personas** can be designed. Next, a **negotiation and trade-off analysis** phase takes place, in which the interactions between requirements are analyzed, and the requirements are prioritized accordingly. Afterward, a **prototyping** phase is needed to observe the effect of explainability and its operationalizations on user experience.

9.2 Discovering the need for explainability

Explainability must only be considered if it is really required. Qualitative research techniques can be used to support this discovery [3]. The purpose is to gather as much information as possible about stakeholders, users, their environment, their expectations, the domain and the project itself.

Qualitative techniques help to understand behaviors, attitudes and details about the domain such as the technical, business and environmental contexts. Interviews, workshops and ethnographic field studies are some of the techniques that can be used for this purpose. Doerr et al. [32] propose a method that includes workshops for capturing the important quality aspects and eliciting NFRs. Conducting workshops with stakeholders is an opportunity to identify the main goals, challenges, as well as qualities and constraints.

During this activity, the dimensions cited in Sect. 8 strongly influence this discovery and should be taken into account. It may be the case that the corporate values influence the need for explainability, arising from a corporate commitment to transparency or a desire to provide a better user experience. Or that legal restrictions apply and require the company to consider explainability and implement it in the system. It also may be the case that the degree of criticality of the domain requires the system to be explainable. The needs and expectations of the user, as well as the culture in which the system operates, are also key factors to consider. Lastly, all these aspects must be considered in relation to the project constraints.

In this phase, we also recommend the use of NFR catalogues and SIGs for the identification and negotiation of NFRs [81]. They can be used to capture knowledge and also allow documentation of the interdependencies found within the project, thus facilitating traceability of requirements [28].

9.3 Explanations for different personas

Understanding the user is the main focus of UCD and one of the main aspects that influence a positive impact of explainability in a system. The engineer must understand the users, their cognitive behavior, their attitudes and the characteristics of the tasks that they must perform [44]. Once again, qualitative techniques such as interviews and ethnographic studies provide important information on the actual **needs and expectations of users** with regard to the system and the information to be received.

Creating personas is a useful technique to identify and define different types of potential users for a system. It supports the identification, description and prioritization of user groups [44]. In the first step, personas hypotheses can be created on the basis of experience and vision. Hypotheses may be used as a first draft to identify groups of users to participate in a more thorough analysis.

By modeling different personas, it is possible to model different user groups and consider each group's particular needs in terms of explanations. Personas may be used as a

support during the identification and operationalization of requirements, to check whether the requirements meet the needs of the identified groups.

Modeling personas requires consideration of aspects such as user roles, level of user domain and technical expertise and **cultural values**. **Domain aspects** also need to be understood in order to understand the particularities of a domain and what users need to know in order to perform their tasks within a system. Using personas, it is also possible to identify the main challenges, what elements of the process really need to be explained, and how to communicate information in a user-friendly manner. As discussed earlier, the domain language also needs to be considered and can be matched to specific user profiles.

In our study, we were able to observe the importance of contextual aspects when we were able to detect variations in the need for explanations based on the context in which the participant was inserted. It is therefore essential to understand the context within which the system is being used and how the system fits into the everyday life of the user. This understanding makes it possible to assess when an explanation may be needed and what needs to be explained in different contexts of use.

Using task analysis and ethnographic techniques, it is possible to observe the needs that users are unable to externalize in interviews and to gain a deeper understanding of the context and domain. Contextual inquiries allow interviews to be conducted in the user environment, so that all aspects are analyzed in a real context [8].

9.4 Requirements negotiation and trade-off analysis

To analyze the impact of explanations on other NFRs, it is necessary to refine explainability into more fine-grained requirements (e.g., softgoals). As previously mentioned, a positive or negative impact depends on how explainability is refined to the functionality level and how these functionalities interact with other NFRs in the system. In practice, the elicitation of NFRs, functional requirements and architecture has to be intertwined, since sometimes a refinement of non-functional aspects is not possible without detailing the functionality or architecture [32].

Existing approaches and tools to support stakeholders in the identification of conflicts between NFRs can be either experience, model or mathematically based [7, 9]. Requirements catalogues can be used during various phases of software development projects, including elicitation and architecture design [70]. They can also be used during trade-off analysis to identify the interdependencies between requirements and understand how more fine-grained requirements (e.g., softgoals) may impact other NFRs [22]. In the context of this work, we used the Transparency SIG as a catalogue to help us gain a better understanding of the interplay between transparency and other NFRs.

The **project constraints** also need to be considered within this activity, including what is feasible given budget, time, technology constraints and other limitations. For instance, it may be that what is needed to satisfice explainability is limited by the project constraints, or that providing a specific type of explanation is not in the interest of the company, as trade secrets may be revealed. Stakeholders such as executives, managers, developers, usability designers, law scholars and system architects may be heard during requirements negotiation in order to reconcile possible conflicts between stakeholder objectives and to prioritize requirements.

9.5 Prototyping and testing

Prototype review has been especially effective in identifying usability issues and optimizing the design of users interfaces. Building prototypes allows the team to capture and validate assumptions about the desired software characteristics [39].

Prototyping is widely used in the field of usability engineering and is a lightweight way to test a system [3]. It allows to collect feedback from stakeholders during the requirements process to validate whether the requirements meet their expectations. In a survey study that included HCI practitioners, computer-based mock-ups and paper prototypes were the primary responses when asked about the usability tools, processes and languages they used in their organizations [55].

Usability tests are usually part of the development process. Prototypes can be validated with real user groups, assessing whether the explanations fulfill their needs or hurt usability. The use of mock-ups can help to identify design flaws and to assess the effect of explanations on user experience. It is also possible to compare whether the represented model of the system (e.g., how the system is presented to the user) matches the users' *mental models*.

The concept of *mental models* [56], which has long been studied and discussed in the HCI community in particular [58, 76], refers to the mental image that users have of the behavior of a system. Mental models offer a deep understanding of people's motivations and thinking processes [100]. The closer the system is to the user's mental model, the better the usability and understandability of the system. Think-aloud and other traditional methods of usability testing [72, 75] should be put in place to evaluate the effect of explainability.

10 Summary

Software systems are deeply integrated into daily life and are becoming increasingly complex. This increasing complexity results in a lack of transparency that hinders understanding and negatively affects trust [29]. In this way, it becomes more important to consider NFRs such as transparency in software systems [97].

In a study with 107 participants, we explored whether explanation is transparency related, whether explanations can help satisfice transparency and what are the advantages and drawbacks of built-in explanations based on end-user perception. We have been able to identify that explainability is not only a means of achieving transparency and building trust, but is also linked to other important NFRs. Explanations can, however, have both a positive and a negative impact on a system. We referred to this phenomenon as the *double-edged sword effect* of explainability.

Usability is one of the NFRs that can be either positively or negatively affected by explainability. It is an essential NFR for system quality and strongly influences factors such as user experience, user satisfaction and trust. Therefore, to prevent the negative impact of explanations on usability, we explored the relationship between explainability and usability in more detail. We provide concrete recommendations for reconciling explanations with usability through the use of UCD techniques.

Although NFRs are fundamental for software quality, they remain a challenge for practice. We reviewed the literature manually in order to gain a better understanding of the challenges of dealing with NFRs and how they are approached during requirements engineering. One of the challenges is that these aspects are often unknown to professionals and therefore difficult to understand and analyze. We refer to such aspects as dimensions. To support requirements engineers, we have identified the different dimensions that should be involved during the requirements engineering process when considering explainability. These dimensions should be considered during the elicitation, analysis and design of explainability.

User-centered techniques can support requirements engineers to elicit requirements that are more aligned with these dimensions, as well as with user needs and context. We therefore selected and suggested a selection of well-known lightweight UCD activities that can be easily incorporated into the requirements process. This strategy prevents the inclusion of complex steps in the development process, which would potentially be difficult to execute in practice.

We are aware of the difficulties in integrating UCD techniques into the development process and how HCI and SE are still sometimes seen as two separate fields [55]. However, as we move toward developing systems with more human-centered values, we need to start integrating such practices into our software development processes.

11 Future directions

While explainability has become an important design concern, it is still under-specified [10]. NFRs are generally poorly understood in comparison with other aspects of the software. There is often terminological confusion with respect to what a given quality aspect means and its characteristics. It is often the case where the same term is understood in different ways [4]. In the case of explainability, a common terminology needs to be investigated in order to facilitate the discussion and analysis of this NFR during the requirements engineering process.

There is also a need to investigate the interrelationships between explainability and other requirements to understand their interactions and taxonomy. In order to bridge this gap, we want to build a SIG of explainability and its interdependencies as a next step to support the requirements engineering process.

Experiments also need to be conducted to evaluate the effect of explanations in deployed systems. We also want to carry out a more in-depth analysis of how the design of explainability would work in practice and the challenges involved, validating our recommendations through empirical studies. Our key purpose is to develop more comprehensive guidelines for practice.

12 Conclusion

Non-functional requirements are difficult to elicit, negotiate and validate. Often, there are also trade-offs between NFRs. Usability may conflict with security: Entering a password may bring more steps to the interaction with the system but protects users' data at the same time. When considering software transparency, even aspects of the same quality attribute may conflict with each other. We concluded that the integration of explanations needs to be carefully evaluated to suit the expectations and particularities of users. Requirements engineers need to explore the costs and benefits of presenting explanations to the user and what they mean in terms of functional and non-functional requirements.

We focused on the relationship between usability and explainability because of the significant influence of usability on software quality. Although one should support the other, poor design and implementation of explanations could do more harm than good. Therefore, we recommend a set of lightweight user-centered activities to support the design of explanations. These recommendations are based on the

findings of our study and on existing literature and can be integrated into the requirements engineering process. We present them as an initial proposal. Empirical evaluations and refinements should be the next step toward more maturity and better understanding of this research topic.

In this paper, we explored explainability as an NFR and its interaction with other requirements related to software transparency. We hope that our findings and conclusions will help to establish realistic activities to cope with explainability in practice. These activities should support the elicitation of explainability requirements and help requirements engineers to conceive explainable systems that do not compromise user experience.

Acknowledgments Open Access funding provided by Projekt DEAL. This work was supported by the research initiative Mobilise between the Technical University of Braunschweig and Leibniz University Hannover, funded by the Ministry for Science and Culture of Lower Saxony. We thank colleagues Wasja Brunotte and Nils Prenner for their feedback on the manuscript.

References

1. 2019 cigi-ipsos global survey on internet security and trust. https://www.cigionline.org/internet-survey-2019 (2019). Accessed 30 Nov 2019
2. Abdollahi B, Nasraoui O (2018) Transparency in fair machine learning: the case of explainable recommender systems. In: Human and machine learning, pp 21–35. Springer
3. Alexander IF, Beus-Dukic L (2009) Discovering requirements: how to specify products and services. Wiley, Hoboken
4. Ameller D, Ayala C, Cabot J, Franch X (2012) How do software architects consider non-functional requirements: an exploratory study. In: 2012 20th IEEE international requirements engineering conference (RE), pp 41–50
5. Anderson N, Herriot P, Hodgkinson GP (2001) The practitioner-researcher divide in industrial, work and organizational (IWO) psychology: where are we now, and where do we go from here? J Occup Org Psychol 74(4):391–411
6. Aronson E, Wilson TD, Brewer MB (1998) Experimentation in social psychology. Handb Soc Psychol 1:99–142
7. Berander P, Damm LO, Eriksson J, Gorschek T, Henningsson K, Jönsson P, Kågström S, Milicic D, Mårtensson F, Rönkkö K et al (2005) Software quality attributes and trade-offs. Blekinge Institute of Technology, Karlskrona
8. Beyer H, Holtzblatt K (1997) Contextual design: defining customer-centered systems. Elsevier, Amsterdam
9. Boehm B, In H (1996) Identifying quality-requirement conflicts. IEEE Softw 13(2):25–35
10. Bohlender D, Köhl MA (2019) Towards a characterization of explainable systems. arXiv:1902.03096
11. Breaux TD, Vail MW, Anton AI (2006) Towards regulatory compliance: extracting rights and obligations to align requirements with regulations. In: 14th IEEE international requirements engineering conference (RE'06), pp 49–58. https://doi.org/10.1109/RE.2006.68
12. Bunt A, Lount M, Lauzon C (2012) Are explanations always important?: a study of deployed, low-cost intelligent interactive systems. In: Proceedings of the 2012 ACM international conference on intelligent user interfaces, pp 169–178. ACM
13. Bussone A, Stumpf S, O'Sullivan D (2015) The role of explanations on trust and reliance in clinical decision support systems. In: 2015 international conference on healthcare informatics, pp 160–169. IEEE
14. Carvalho RM (2017) Dealing with conflicts between non-functional requirements of UbiComp and IoT applications. In: 2017 IEEE 25th international requirements engineering conference (RE), pp 544–549. IEEE
15. Carvallo J.P, Franch X, Quer C (2006) Managing non-technical requirements in cots components selection. In: 14th IEEE international requirements engineering conference (RE'06), pp 323–326. IEEE
16. Charmaz K (2006) Constructing grounded theory: a practical guide through qualitative analysis. Sage, Thousand Oaks
17. Chazette L (2019) Survey data - perception of end-users regarding the need for explanations in software systems. https://doi.org/10.5281/zenodo.3261127
18. Chazette L, Karras O, Schneider K (2019) Do end-users want explanations? Analyzing the role of explainability as an emerging aspect of non-functional requirements. In: 2019 IEEE 27th international requirements engineering conference (RE). IEEE
19. Choe JM (2004) The consideration of cultural differences in the design of information systems. Inf Manag 41(5):669–684
20. Chromik M, Eiband M, Völkel ST, Buschek D (2019) Dark patterns of explainability, transparency, and user control for intelligent systems. In: IUI workshops
21. Chung L, Nixon BA (1995) Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In: 1995 17th international conference on software engineering, pp 25–25. IEEE
22. Chung L, do Prado Leite JCS (2009) On non-functional requirements in software engineering. In: Conceptual modeling: foundations and applications, pp 363–379. Springer
23. Cohen J (1968) Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit. Psychol Bull 70(4):213–220
24. Cooper A, Reimann R, Cronin D (2007) About face 3: the essentials of interaction design. Wiley, Hoboken
25. Cranor LF, Garfinkel S (2004) Guest editors' introduction: secure or usable? IEEE Secur Priv 2(5):16–18
26. Cysneiros LM, do Prado Leite JCS, Neto JDMS (2001) A framework for integrating non-functional requirements into conceptual models. Requir Eng 6(2):97–115
27. Cysneiros LM, Raffi M, do Prado Leite JCS (2018) Software transparency as a key requirement for self-driving cars. In: 2018 IEEE 26th international requirements engineering conference (RE), pp 382–387. IEEE
28. Cysneiros LM, Werneck VM, Kushniruk A (2005) Reusable knowledge for satisficing usability requirements. In: 13th IEEE international conference on requirements engineering (RE'05), pp 463–464. IEEE

29. Cysneiros LM, Werneck VMB (2009) An initial analysis on how software transparency and trust influence each other. In: WER
30. Damian DE, Zowghi D (2003) An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In: Proceedings of the 36th annual Hawaii international conference on system sciences, 2003, pp 10. IEEE
31. De La Vara JL, Wnuk K, Berntsson-Svensson R, Sánchez J, Regnell B (2011) An empirical study on the importance of quality requirements in industry. In: SEKE, pp 438–443
32. Doerr J, Kerkow D, Koenig T, Olsson T, Suzuki T (2005) Non-functional requirements in industry-three case studies adopting an experience-based NFR method. In: 13th IEEE international conference on requirements engineering (RE'05), pp 373–382. IEEE
33. Doran D, Schulz S, Besold TR (2017) What does explainable ai really mean? A new conceptualization of perspectives. arXiv :1710.00794
34. do Prado Leite JCS, Cappelli C, (2010) Software transparency. Bus Inf Syst Eng 2(3):127–139
35. Doshi-Velez F, Kim B (2017) Towards a rigorous science of interpretable machine learning. arXiv:1702.08608
36. Doshi-Velez F, Kortz M, Budish R, Bavitz C, Gershman S, O'Brien D, Schieber S, Waldo J, Weinberger D, Wood A (2017) Accountability of ai under the law: the role of explanation. arXiv:1711.01134
37. Ethics guidelines for trustworthy ai. (2019) https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai. Accessed 30 Nov 2019
38. Flavián C, Guinalíu M, Gurrea R (2006) The role played by perceived usability, satisfaction and consumer trust on website loyalty. Inf Manag 43(1):1–14
39. Fotrousi F, Fricker SA, Fiedler M (2014) Quality requirements elicitation based on inquiry of quality-impact relationships. In: 2014 IEEE 22nd international requirements engineering conference (RE), pp 303–312. IEEE
40. Frese M (1987) A theory of control and complexity: Implications for software design and integration of computer systems into the work place. In: Psychological issues of human computer interaction in the work place. North-Holland Publishing Co., NLD, pp 313–337
41. Glinz M (2007) On non-functional requirements. In: 15th IEEE international requirements engineering conference (RE 2007), pp 21–26. IEEE
42. Goodman B, Flaxman S (2017) European union regulations on algorithmic decision-making and a "right to explanation". AI Mag 38(3):50–57
43. Groen EC, Kopczyńska S, Hauer MP, Krafft TD, Doerr J (2017) Users—the hidden software product quality experts? A study on how app users report quality aspects in online reviews. In: 2017 IEEE 25th international requirements engineering conference (RE), pp 80–89. IEEE
44. Gulliksen J, Göransson B, Boivie I, Blomkvist S, Persson J, Cajander Å (2003) Key principles for user-centred systems design. Behav Inf Technol 22(6):397–409
45. Gutmann P, Grigg I (2005) Security usability. IEEE Secur Priv 3(4):56–58
46. Hamel L (2006) Visualization of support vector machines with unsupervised learning. In: Proceedings of 2006 IEEE symposium on computational intelligence in bioinformatics and computational biology
47. Hartson R, Pyla PS (2012) The UX book: process and guidelines for ensuring a quality user experience. Elsevier, Amsterdam
48. Hehn J, Uebernickel F (2018) The use of design thinking for requirements engineering: an ongoing case study in the field of innovative software-intensive systems. In: 2018 IEEE 26th international requirements engineering conference (RE), pp 400–405. IEEE
49. Herlocker JL, Konstan JA, Riedl J (2000) Explaining collaborative filtering recommendations. In: Proceedings of the 2000 ACM conference on computer supported cooperative work, pp 241–250. ACM
50. Hoffmann CP, Lutz C, Meckel M (2014) Digital natives or digital immigrants? The impact of user characteristics on online trust. J Manag Inf Syst 31(3):138–171
51. Hosseini M, Shahri A, Phalp K, Ali R (2016) Foundations for transparency requirements engineering. In: International working conference on requirements engineering: foundation for software quality, pp 225–231. Springer
52. ISO 9241-210:2019 Ergonomics of human-system interaction—Part 210: human-centred design for interactive systems. Standard, International Organization for Standardization, Geneva CH (2019)
53. ISO/IEC 25010:2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models. Standard, International Organization for Standardization, Geneva CH (2011)
54. Jakulin A, Možina M, Demšar J, Bratko I, Zupan B (2005) Nomograms for visualizing support vector machines. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining, KDD '05, pp 108–117. ACM, New York, NY, USA
55. Jerome B, Kazman R (2005) Surveying the solitudes: an investigation into the relationships between human computer interaction and software engineering in practice. In: Seffah A, Gulliksen J, Desmarais MC (eds) Human-centered software engineering—integrating usability in the software development lifecycle. Springer, Netherlands, pp 59–70. https://doi.org/10.1007/1-4020-4113-6_4
56. Johnson-Laird PN (1983) Mental models: towards a cognitive science of language, inference, and consciousness, 6. Harvard University Press, Cambridge
57. Yee Ka-Ping (2004) Aligning security and usability. IEEE Secur Priv 2(5):48–55
58. Kieras DE, Bovair S (1984) The role of a mental model in learning to operate a device. Cognit Sci 8(3):255–273
59. Kim B, Glassman E, Johnson B, Shah J (2015) ibcm: Interactive Bayesian case model empowering humans via intuitive interaction. Massachusetts Institute of Technology, Cambridge, MA
60. Koh PW, Liang P (2017) Understanding black-box predictions via influence functions. arXiv:1703.04730
61. Konstan JA, Riedl J (2012) Recommender systems: from algorithms to user experience. User Model User-Adapt Interact 22(1–2):101–123
62. Kulesza T, Stumpf S, Burnett M, Yang S, Kwan I, Wong WK (2013) Too much, too little, or just right? Ways explanations impact end users' mental models. In: 2013 IEEE symposium on visual languages and human centric computing, pp 3–10. IEEE
63. Kummer TF, Leimeister JM, Bick M (2012) On the importance of national culture for the design of information systems. Bus Inf Syst Eng 4(6):317–330
64. Landes D, Studer R (1995) The treatment of non-functional requirements in mike. In: European software engineering conference, pp 294–306. Springer
65. Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics 33(1):159–174

66. Lepri B, Oliver N, Letouzé E, Pentland A, Vinck P (2018) Fair, transparent, and accountable algorithmic decision-making processes. Philos Technol 31(4):611–627
67. Leßmann H (2017) Durchführung einer umfrage-studie zur nutzung von code-reviews in der praxis. Master's thesis, Leibniz Universität Hannover, Fachgebiet Software Engineering
68. Lim BY, Dey AK (2009) Assessing demand for intelligibility in context-aware applications. In: Proceedings of the 11th international conference on Ubiquitous computing, pp 195–204. ACM
69. Lipton ZC (2018) The mythos of model interpretability. Commun ACM 61(10):36–43
70. Mairiza D, Zowghi D (2010) Constructing a catalogue of conflicts among non-functional requirements. In: International conference on evaluation of novel approaches to software engineering, pp 31–44. Springer
71. Mao JY, Vredenburg K, Smith PW, Carey T (2005) The state of user-centered design practice. Commun. ACM 48(3):105–109
72. Mayhew DJ, Mayhew D (1999) The usability engineering lifecycle: a practitioner's handbook for user interface design. Morgan Kaufmann, Burlington
73. Miles MB, Huberman AM (1994) Qualitative data analysis: an expanded sourcebook. Sage, Thousand Oaks
74. Možina M, Demšar J, Kattan M, Zupan B (2004) Nomograms for visualization of naive Bayesian classifier. In: Boulicaut JF, Esposito F, Giannotti F, Pedreschi D (eds) Knowledge discovery in databases: PKDD 2004. Springer, Berlin, pp 337–348
75. Nielsen J (1994) Usability engineering. Elsevier, Amsterdam
76. Norman DA (1987) Some observations on mental models. In: Human-computer interaction: a multidisciplinary approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 241–244
77. Norman KL (1991) The psychology of menu selection: designing cognitive control at the human/computer interface. Ablex Pub. Corp., Norwood, NJ
78. Otto PN, Antón AI (2007) Addressing legal requirements in requirements engineering. In: 15th IEEE international requirements engineering conference (RE 2007), pp 5–14. IEEE
79. O'Neil C (2016) Weapons of math destruction: how big data produces inequality and threatens democracy. Penguin Randomhouse Ltd, New York
80. Pacey A (1983) The culture of technology. MIT Press, Cambridge
81. Paech B, Kerkow D (2004) Non-functional requirements engineering-quality is essential. In: 10th international workshop on requirements engineering foundation for software quality
82. Papadimitriou A, Symeonidis P, Manolopoulos Y (2012) A generalized taxonomy of explanations styles for traditional and social recommender systems. Data Min Knowl Discov 24(3):555–583
83. Prensky M (2001) Digital natives, digital immigrants. On Horiz 9(5):1–6
84. Pynadath DV, Barnes MJ, Wang N, Chen JY (2018) Transparency communication for machine learning in human-automation interaction. In: Human and machine learning, pp 75–90. Springer
85. Ribeiro MT, Singh S, Guestrin C (2016) 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. Association for Computing Machinery, New York, NY, USA, pp 1135–1144. https://doi.org/10.1145/2939672.2939778
86. Regulation (eu) (2016) 2016/679 of the European parliament and of the council of 27 april 2016 (general data protection regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj
87. Saldaña J (2015) The coding manual for qualitative researchers. Sage, Thousand Oaks
88. Seffah A, Gulliksen J, Desmarais MC (2005) An introduction to human-centered software engineering. In: Seffah A, Gulliksen J, Desmarais MC (eds) Human-centered software engineering—integrating usability in the software development lifecycle. Springer Netherlands, Dordrecht, pp 3–14. https://doi.org/10.1007/1-4020-4113-6_1
89. Seffah A, Metzker E (2004) The obstacles and myths of usability and software engineering. Commun ACM 47(12):71–76
90. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D (2017) Grad-cam: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision, pp 618–626
91. Siena A, Mylopoulos J, Perini A, Susi A (2009) Designing law-compliant software requirements. In: International conference on conceptual modeling, pp 472–486. Springer
92. Tapscott D (2010) Grown up digital: how the net generation is changing your world. Int J Mark Res 52(1):139
93. Thomsen S (2004) Corporate values and corporate governance. Corp Gov Int J Bus Soc 4:29–46
94. Tintarev N, Masthoff J (2012) Evaluating the effectiveness of explanations for recommender systems. User Model User-Adapt Interact 22(4–5):399–439
95. Tomsett R, Braines D, Harborne D, Preece AD, Chakraborty S (2018) Interpretable to whom? A role-based model for analyzing interpretable machine learning systems. CoRR arXiv:1806.07552
96. Vanwelkenhuysen J (1996) Quality requirements analysis in customer-centered software development. In: Proceedings of the second international conference on requirements engineering, pp 117–124. IEEE
97. Whittle J (2019) Is your software valueless? IEEE Softw 36(3):112–115
98. Winkler J.P, Vogelsang A (2017) What does my classifier learn? A visual approach to understanding natural language text classifiers. In: International conference on applications of natural language to information systems, pp 468–479. Springer
99. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2
100. Young I (2008) Mental models: aligning design strategy with human behavior. Rosenfeld Media, New York
101. Zinovatna O, Cysneiros LM (2015) Reusing knowledge on delivering privacy and transparency together. In: 2015 IEEE fifth international workshop on requirements patterns (RePa), pp 17–24. IEEE