INAUGURAL–DISSERTATION

submitted to the
Combined Faculty of Mathematics, Engineering, and Natural Sciences
of
Heidelberg University, Germany
for the degree of
Doctoral of Natural Sciences

Put forward by
Leon Radeck M.Sc.
Born in Forchheim

Oral examination: ……..

Collecting feedback and deriving requirements

Supervisors:

Prof. Dr. Barbara Paech	Heidelberg University
Prof. Dr. Norbert Seyff	University of Applied Sciences and Arts Northwestern Switzerland

ABSTRACT

[Context] User feedback plays an important role in software development, improving system acceptance, reducing project failures and enhancing customer loyalty. To achieve these benefits, software organizations actively collect, analyze and validate feedback to derive changes to existing requirements or new requirements. However, feedback collection and requirements derivation are hindered by several problems that are reported in literature. Feedback from a small selection of users can result in biased or incomplete requirements; vague or ambiguous feedback makes it difficult to map it to requirements; feedback lacking details to propose a change to the application cannot be used to derive requirements; gathering feedback at specific times is not possible by relying solely on user initiative and feedback that is used for requirements derivation must be validated to ensure support among users. **[Objective]** The goal of this thesis is to introduce an approach for requirements engineers that enables them to collect feedback and derive requirements without facing the mentioned problems. The approach consists of a process to collect feedback and derive requirements, as well as the platform "smartFEEDBACK" that supports the process. **[Methods]** To achieve the goal, this thesis follows the design science methodology consisting of problem investigation, treatment design and treatment validation. The problem investigation consists of a systematic mapping study to understand the current state and practice of collecting feedback over platforms. Platforms are online tools that facilitate the collection of feedback from multiple stakeholders and enable exchange about that feedback among them. The results of the problem investigation are the basis for the treatment design (our approach). The treatment validation validates whether the approach is feasible and effective, as well as whether the users are satisfied with it and whether the approach can be improved. For the treatment validation the approach is applied in the large-scale research project SMART-AGE that examines the use of four interconnected apps developed for older adults to improve their quality of life. **[Contributions]** We contribute our approach that enables researchers and practitioners to collect feedback and derive requirements without encountering the mentioned problems. Additionally, we contribute our mapping study that can serve as a foundation for future systematic mapping studies or as an orientation for designing individual feedback platforms. We also offer a dataset of change requests collected in SMART-AGE, providing insights into real-world feedback from older adults. This dataset is useful for researchers and practitioners aiming to understand the specific needs and preferences of this user group. Lastly, we contribute a validation of the approach's feasibility, effectiveness, user satisfaction and improvement, offering a benchmark for researchers to compare their approaches.

ZUSAMMENFASSUNG

[Kontext] Nutzerfeedback spielt eine wichtige Rolle in der Softwareentwicklung, da es die Akzeptanz von Systemen verbessert, Projektausfälle reduziert und die Kundentreue positiv beeinflusst. Zur Realisierung dieser Vorteile, sammeln, analysieren und validieren Softwareorganisationen aktiv Feedback, um Änderungen an bestehenden Anforderungen oder neue Anforderungen abzuleiten. Allerdings wird die Feedbacksammlung und Anforderungsableitung durch mehrere in der Literatur beschriebene Probleme erschwert. Feedback von einer kleinen Anzahl an Nutzern kann zu nicht repräsentativen oder unvollständigen Anforderungen führen; vages oder unklar formuliertes Feedback erschwert die Zuordnung zu Anforderungen; Feedback, das nicht detailliert genug ist, um Änderungen an der Anwendung vorzuschlagen, kann nicht zur Ableitung von Anforderungen verwendet werden; der Zeitpunkt für das Sammeln von Feedback kann nicht kontrolliert werden, wenn man nur darauf wartet, dass die Nutzer selbstständig Feedback abgeben; und Feedback, das zur Ableitung von Anforderungen verwendet wird, muss validiert werden, um die Unterstützung durch Nutzer sicherzustellen. **[Zielsetzung]** Ziel dieser Arbeit ist es, einen Ansatz für Anforderungsingenieure vorzustellen, der es ihnen ermöglicht, Feedback zu sammeln und Anforderungen abzuleiten, ohne auf die genannten Probleme zu stoßen. Der Ansatz besteht aus einem Prozess zur Sammlung von Feedback und der Ableitung von Anforderungen sowie der Plattform „smartFEEDBACK“, die diesen Prozess unterstützt. **[Methode]** Um das Ziel zu erreichen, folgt die Arbeit der Design-Science-Methode, bestehend aus Problemuntersuchung, Lösungsentwurf und Lösungsvalidierung. Die Problem-untersuchung umfasst eine systematische Mapping-Studie, um den aktuellen Stand und die Praxis der Feedbacksammlung über Plattformen zu verstehen. Plattformen sind Online-Tools, die die Sammlung von Feedback von mehreren Stakeholdern und den Austausch über dieses Feedback ermöglichen. Die Ergebnisse der Problemuntersuchung bilden die Grundlage für den Lösungsentwurf. Der Lösungsentwurf umfasst das Design des Prozesses und der Plattform. Die Lösungsvalidierung überprüft, ob der Ansatz machbar und effektiv ist, ob die Nutzer zufrieden sind und ob der Ansatz verbessert werden kann. Für die Lösungsvalidierung wird der Ansatz im Forschungsprojekt SMART-AGE angewendet, das den Einsatz von vier miteinander vernetzten Apps untersucht, die für ältere Erwachsene entwickelt wurden, um deren Lebensqualität zu verbessern. **[Beiträge]** Wir leisten einen Beitrag durch die Bereitstellung unseres Ansatzes, der es Forschern und Praktikern ermöglicht, Feedback zu sammeln und Anforderungen abzuleiten, ohne auf die genannten Probleme zu treffen. Darüber hinaus tragen wir mit unserer Mapping-Studie bei, die als Grundlage für zukünftige Mapping-Studien oder als Orientierung für die Entwicklung individueller Plattformen dienen kann. Wir bieten auch einen Datensatz, der Einblicke in die Änderungswünsche der Nutzer von SMART-AGE ermöglicht. Dieser Datensatz ist nützlich für Forscher und Praktiker, die die Bedürfnisse dieser Nutzergruppe verstehen möchten. Schließlich tragen wir mit einer Validierung der Machbarkeit, Effektivität, Nutzerzufriedenheit und Verbesserungsmöglichkeiten des Ansatzes bei und bieten so eine Basis auf der Forscher ihre Ansätze vergleichen können.

ACKNOWLEDGEMENTS

I want to express my gratitude to my supervisor Professor Barbara Paech for providing me the opportunity to pursue my doctoral studies and to be part of her Software Engineering Group at Heidelberg University. I appreciated always that I could reach out to her with any questions and that she took a genuine interest in the progress of my work by continuously giving me valuable feedback. I learned from her to question more deeply the reasons behind why and how something is done, rather than simply doing it. Furthermore, I learned a lot about presenting my own work and about the importance of documenting insights from discussions in a structured manner. I am also very grateful to Norbert Seyff for agreeing to take on the role of second examiner and I hope he doesn't regret it after seeing the page count of this thesis. I would like to thank all my (former) colleagues: I want to thank Michael for the conversations we shared over the past four years and his support. I would like to thank Anja for going to the Mensa frequently with me and for all the tips and advice she shared. I want to thank Willi for his support regarding tasks of the SMART-AGE research project, especially for managing the order of large amounts of tablets and licenses. I want to thank Stephanie for her support with all questions I had, for reserving rooms, for giving me letters and for the many conversations we had. I want to thank Astrid for helping me onboard the software engineering group and for her help with any questions. Also, I want to thank Eoin for all the table tennis sessions. For the help regarding the design of the process to collect feedback and derive requirements, I want to thank Dennis Scherbatschenko. For the support regarding the development of smartFEEDBACK, I want to thank Loris Wilwert, Jonas Roos and Benjamin Tuna. I am also grateful to Daniela Tratz-Weinmann for researching how usage data can be analyzed, which inspired the validation of the approach. I would also like to thank many people that helped me in the research project SMART-AGE. I want to express my gratitude to Niklas Kern and Roman Eiser for designing the help materials. Furthermore, I would like to thank David Schwenke for developing the SMART-AGE portal and the reminder system. Special thanks also go again to Niklas and David for their assistance in automating a large part of the tablet configuration, which was absolutely essential. I want to thank Zaynab and Dionysios for their help with configuring hundreds of tablets. I want to thank Robert Jakobs and Moritz Buehrer for the implementation of smartIMPULS and smartVERNETZT. I would also like to thank all the people who supported me outside of my work. I want to thank my family for their constant support, encouragement and care during my life and I want to thank all my friends—especially those from "Innerer Kreis", the "HD Family" and my shared flat, as well as all my friends in general—for the many enjoyable experiences we have shared and continue to share.

Table of Contents

PART I

PRELIMINARIES

Chapter 1

Introduction

This Chapter gives an introduction into the relevance of user feedback and problems that occur during feedback collection and requirements derivation in Section 1.1. It describes research methodology in Section 1.2. It presents contributions in Section 1.3. It gives an outline of the thesis in Section 1.4 and it lists previous publications which content is used in this thesis in Section 1.5.

User feedback improves system acceptance (Kujala, 2008), reduces project failure (El Emam and Koru, 2008) and increases customer loyalty (Kabbedijk et al., 2009). User feedback is especially essential for the continuous development of software, because it contributes substantially to the elicitation of requirements (Bajic and Lyons, 2011). Software organizations actively collect, analyze and validate feedback so that changes to existing requirements or new requirements can be derived (Li et al., 2024). In academic research, the importance of feedback collection and requirements derivation has led to the development of Crowd-Requirements Engineering (CrowdRE), a research field dedicated to automating the collection of feedback from large user groups, referred to as the "crowd" with the goal of deriving validated user requirements (Groen et al., 2017). However, feedback collection and requirements derivation are hindered by several problems. We present the problems in Section 1.1.

1.1 Problem context

This Section describes three problems from literature that are associated with feedback collection and requirements derivation: *P1: Completeness of feedback*, *P2: Control of timing of feedback collection* and *P3: Support of change requests among users*. *P1* is the most important problem as it contains multiple different subordinate problems (*P1.1: A lot of feedback can be collected from a lot of users*, *P1.2: Feedback can be mapped to requirements* and *P1.3: Feedback contains change requests*). There are more problems in the literature than we want to solve here. For example, one big problem is that users are not motivated to give feedback (Kolpondinos and Glinz, 2020) and one prominent strategy for motivating users to give feedback is the use of gamification (Kolpondinos and Glinz, 2020). However, we do not address this problem, because we know from

literature that our users, who are older adults, may not be receptive to gamification elements (Sardi et al., 2017). Furthermore, our users are participating in a study where providing feedback is an integral part, meaning they are already inherently motivated to contribute. In general, we do not focus on problems that are not relevant for the conduction of our approach in the context of our research project SMART-AGE (see Section 2.2). For example, we also do not focus on the problem of identifying users who want to give feedback about software (Kolpondinos and Glinz, 2020), because we have plenty of users already through the recruiting in SMART-AGE. The problems that our approach aims to treat (*P1*, *P2* and *P3*) are described in the following.

P1: Completeness of feedback

We define feedback as *complete*, when a lot of feedback can be collected from a lot of users *(P1.1*), when it can be mapped to requirements (*P1.2*) and when it contains change requests (*P1.3*).

P1.1: A lot of feedback can be collected from a lot o**f** users

We want to collect as much feedback as possible from a large number of users in order to understand their needs. If feedback comes from only a small subset of users, it may not represent the full spectrum of user needs. This can lead to the requirements engineers (REengs) receiving only the needs of a minority, while missing out on the broader needs of the majority of users (Tizard et al., 2020). The feedback also may be biased toward specific demographics or use cases, resulting in software that works well for a subset of users but poorly for others. For example, demographic differences, such as gender or age, affect who provides feedback, potentially leading to solutions that are not inclusive of all users (Tizard et al., 2021). Furthermore, as only a small percentage of feedback addresses the desire for change or for new features (Panichella et al., 2015) and because these change requests are relevant for deriving requirements, we want to collect as much feedback as possible from each user.

P1.2: Feedback can be mapped to requirements effortlessly

To derive changes to existing requirements based on feedback, the feedback must be mapped to these requirements. The mapping to requirements requires that the feedback is comprehensible. However, feedback often times lacks detail or is ambiguous, making it difficult to understand and to map to requirements (Laporti et al., 2009; Lai et al., 2014; Chevalier and Buckles, 2019; Van Oordt and Guzman, 2021; Lim et al., 2021; Li et al., 2024). For example, feedback that states that "the filter function is not working" potentially does not provide enough details to map the feedback to a requirement, because it could refer to any filter functionalities within the application, such as search filters, sorting filters or other content filtering functionalities. However, even when feedback is not ambiguous, the effort of analyzing it and mapping it to requirements is high (Van Oordt and Guzman, 2021). This is why we want to reduce this effort.

P1.3: Feedback contains change requests

According to (Van Oordt and Guzman, 2021) practitioners from the industry that collect feedback often find feedback "not helpful". Feedback often lacks the necessary details to propose a change to the application, which is important for requirements derivation (Panichella et al., 2015). For example, the feedback "I don't like the app" is not detailed enough to derive any concrete changes to the app. In our approach we want to collect feedback that addresses change in more detail, so that we can use it for requirements derivation.

P2: Control of timing of feedback collection

During feedback collection it is important to collect feedback at specific, targeted times. For example, after special events such as the release of a software update, it is necessary to receive timely feedback in order to assess whether the users are satisfied with the new software version or whether they desire changes. Also, in our research project SMART-AGE we have specific times when we want to collect specific feedback. For example, we want to collect feedback about the usability of an app not directly when the users started to use the app, but after they already have used the app for a while. This is because asking too early for feedback might disturb the users (Fotrousi et al., 2018). Collecting feedback at specific times is not possible by relying on feedback given autonomously, as it depends on the initiative of the users (Maalej et al., 2009).

P3: Support of change requests among users

According to (Van Oordt and Guzman, 2021) some users are very passionate about a specific feature that they wish would exist, but according to industry practitioners this does not mean that the feature is beneficial for the vast majority of the users. To decide whether a change request of the users should be implemented, it is important to know whether this change request is supported among the users.

1.2 Research methodology

This research in this thesis follows the *design science* methodology by Wieringa (Wieringa, 2014). The research methodology is explained in general and regarding the specific instance of our approach.

[Design Science] Design science is the *design* and *investigation* of *artifacts* in *context*. The design and investigation are the two major activities of design science and the artifact is the object that is studied. The artifact can be software, hardware, an organization, a business process or a method. Essentially, anything that can be designed by a design researcher.

Instance

In our case, the artifact is our approach which collects feedback and derives requirements based on the feedback. The approach consists of the process to collect feedback and to derive requirements and the platform called "smartFEEDBACK" (SF) that supports the process.

The context of the artifact refers to the environment in which the artifact operates. This context may include other software, hardware, organizations, business processes, and methods, as well as entities that cannot be designed, such as people, values, desires, fears, goals, norms, and budgets. The *problem context* refers to the part of the context that relates to the problem that is solved. The problem context contains the *social context* and the *knowledge context*. The social context contains the stakeholders who may affect the project or may be affected by it. Stakeholders include users, operators, maintainers, and others associated with the artifact. The knowledge context consists of existing theories from science and engineering, specifications of currently known designs, facts and other knowledge relevant for design and investigation of the artifact.

Instance

In our case, the social context is the users of our platform and us, the REengs. The knowledge context consists of the problems *P1*, *P2* and *P3*, that we identified in literature and that we selected based on our knowledge regarding the research project and the user group (e.g. we did not focus on the problem of user motivation, because of the study rewards and the inherent motivation of the users). Furthermore, the knowledge context consists of our insights identified through the conducted mapping study in Chapter 3 and our own knowledge regarding requirements engineering and software development.

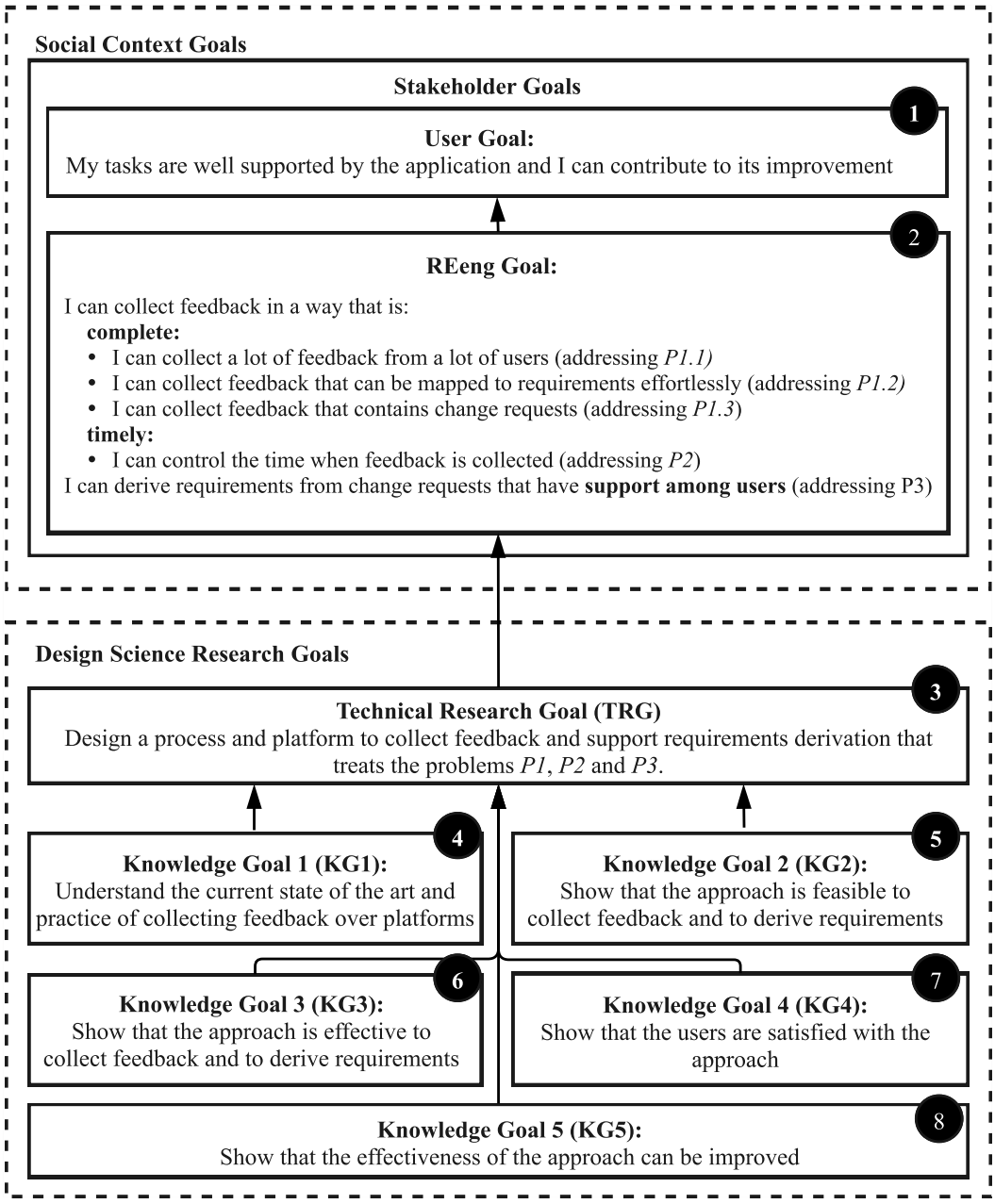
[Goals of a design science research project] The goals of a design science research project consist of the *social context goals* and the *design science research goals*. The social context goals consist of the *stakeholder goals*. The design science research goals include *technical research goals* and *knowledge goals*. Technical research goals improve the performance of an artifact in context. Knowledge goals describe phenomena and explain them. Knowledge Goals are refined into *knowledge questions*.

Instance

We describe our goal structure in Figure 1.1. There are two stakeholder goals (❶ and ❷) and six design science research goals (❸, ❹, ❺, ❻, ❼, ❽). The stakeholder goals are associated to the stakeholders, who comprise the users and

the REengs. The users' goal is that their task is well supported by the application and that they can contribute to its improvement (❶).

Figure 1.1: Goal structure of this thesis. Arrows are indicating that a goal contributes to another goal.



The goal of the REengs (❷) is to collect feedback in a way that is complete (addressing *P1*) and timely (addressing *P2*). Regarding requirements derivation, the REengs have the goal to derive requirements based on change requests that have support among users (addressing *P3*). This thesis aims to support the REengs goal, which contributes to the achievement of the user goal. The technical research goal (TRG) (❸) contributes to the achievement of the REengs goal. The TRG is to design a process and platform to collect feedback and support requirements

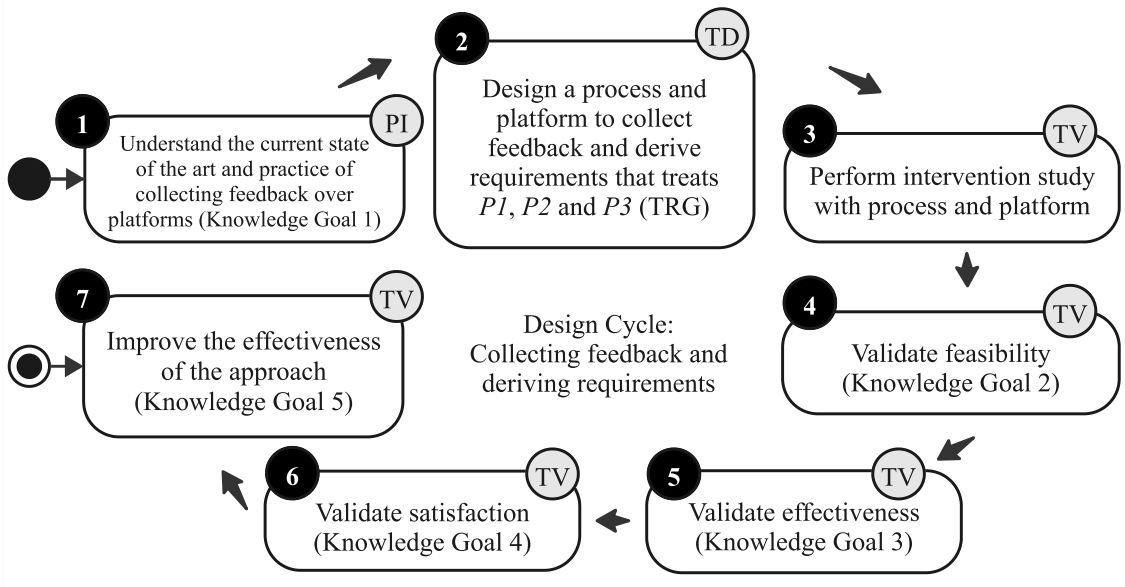
derivation that treats the problems *P1*, *P2* and *P3*. The following knowledge goals support the technical research goal. Knowledge goal 1 (❹) is about understanding how feedback can be collected over platforms. To achieve knowledge goal 1, a systematic mapping study about the state of the art and practice of collecting feedback over platforms is conducted. Knowledge goal 2 (❺) validates whether the approach is feasible to collect feedback and to derive requirements. Knowledge goal 3 (❻) validates whether the approach is effective in collecting feedback and deriving requirements. Knowledge goal 4 (❼) validates the user satisfaction of the approach and knowledge goal 5 (❽) shows how the effectiveness of the approach can be improved.

[Design Cycle] The design activity of a design science project can be decomposed into three tasks, namely, problem investigation, treatment design and treatment validation. During problem investigation, the researchers try to understand more about the problem context or they investigate existing solutions. During the treatment design one or more artifacts are designed that interact with the problem context to treat the problem. The treatment validation checks whether the problem is treated by the artifacts. The set of these three tasks is called *design cycle*.

Instance

The design cycle for this thesis is shown in Figure 1.2 along with references to the research goals.

Figure 1.2: Design cycle of this thesis as UML activity diagram. The activities represent the achievement of each research goal. The activities belong to the design science tasks: Problem Investigation (PI), Treatment Design (TD) and Treatment Validation (TV).



The design cycle starts with the problem investigation (❶). During problem investigation, a systematic mapping study is conducted, which analyzes the state of art and practice for the collection of feedback over platforms. The results of the mapping study are the basis for the treatment design (❷). For the treatment validation, the treatment design is applied in the research project SMART-AGE (❸) and validated for feasibility (❹), effectiveness (❺) and satisfaction (❻). It is also shown how the effectiveness of the approach can be improved (❼).

1.3 Contributions

We make several contributions that are helpful for other researchers and practitioners who want to collect feedback and derive requirements from a large number of users.

The most important contribution is our validated approach, consisting of the process to collect feedback and derive requirements (see Chapter 4) and the platform SF that supports this process (see Chapter 5, addressing the *TRG*). Our approach enables REengs to systematically collect feedback and derive requirements regarding any desired app.

Second, we contribute a systematic mapping study about the current state of the art and practice of collecting feedback over platforms (see Chapter 3, addressing *knowledge goal 1*). The mapping study investigates what types of feedback are collected at what times, what the context of feedback collection is, how feedback collection over platforms is evaluated and what the results are, as well as how the platforms work in general. The results of the mapping study influence the process to collect feedback and derive requirements (see Chapter 4) and they influence the design of SF (see Chapter 5). The results of the mapping study also help other researchers that also want to design feedback platforms.

Third, we contribute a validation of the feasibility of our approach to collect feedback and derive requirements (see Chapter 7, addressing *knowledge goal 2*). The validation reports whether it is feasible to collect feedback, whether it is feasible to derive requirements and whether the usage of SF is feasible. The results provide other researchers with valuable benchmarks for assessing the feasibility of their approach to collect feedback collection and derive requirements.

Fourth, we contribute a validation of the effectiveness of our approach (see Chapter 8, addressing *knowledge goal 3*). The validation reports whether the feedback that is collected is complete (addressing *P1*), whether the feedback can be collected timely (addressing *P2*), and whether requirements can be derived that have support among the users (addressing *P3*).

Fifth, we provide a validation of the satisfaction of users with the approach (see Chapter 9, addressing *knowledge goal 4*). The validation includes the satisfaction of the users with the platform, with our questions and with the process of asking FUQ. The validation of the satisfaction can be interesting for other researchers that want to refine and optimize their feedback platform.

Sixth, we contribute an analysis of whether the effectiveness of our approach can be improved (see Chapter 10, addressing *knowledge goal 5*). The insights are helpful for other researchers that also want to use our approach, as they can use the provided insights to improve its effectiveness.

1.4 Structure of this thesis

The thesis is composed of six parts and 12 chapters. Table 1.1 gives an overview of the structure. Part I gives an introduction, Part II describes the problem investigation that addresses *knowledge goal 1*. Part III presents the treatment design, which addresses the *TRG*. Part IV describes the treatment validation, which addresses *knowledge goal 2*, *knowledge goal 3*, *knowledge goal 4* and *knowledge goal 5*. Part V gives a summary and an outlook. Part VI contains the appendix.

Table 1.1: Structure of the thesis

	Preliminaries	**Chap.**
Part I	**Introduction**	1
	Foundations	2
Part II	Problem Investigation	
	State of the art and practice: Collection of feedback over platforms *Knowledge Goal 1*: Understand the current state of the art and practice of collecting feedback over platforms	3
Part III	Treatment Design	
	Technical Research Goal (*TRG*): Design a process and platform to collect feedback and support requirements derivation that treats the problems *P1*, *P2* and *P3*	
	Process to collect feedback and derive requirements	4
	smartFEEDBACK (SF) - Platform that supports the process	5
Part IV	Treatment validation	
	Study context	6
	Validation of feasibility *Knowledge Goal 2*: Show that the approach is feasible to collect feedback and derive requirements	7
	Validation of effectiveness *Knowledge Goal 3*: Show that the approach is effective in collecting feedback and deriving requirements	8

	Validation of satisfaction *Knowledge Goal 4*: Show that the users are satisfied with the approach	9
	Improvement of the approach *Knowledge Goal 5:* Show that the effectiveness of the approach can be improved	10
Part V	**Conclusion**	11
	Outlook	12

1.5 Previous Publications

Some chapters of this thesis include already published findings. Table 1.2 lists the publications which contain the findings along with the chapter where these findings are used.

Table 1.2: Previous publications

Citation	Publication	Chapter
(Radeck et al., 2022)	Radeck et al., Understanding IT-related Well-being, Aging and Health Needs of Older Adults with Crowd-RE, *Proceedings of the International Workshop on Requirements Engineering for Well-Being, Aging, and Health*, IEEE, 2022.	2, 4, 5
(Radeck and Paech, 2023)	Radeck L, Paech B, Integrating Implicit Feedback into Crowd Requirements Engineering – A Research Preview. *Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality*, ACM, 2023	4, 5
(Radeck and Paech, 2024)	Radeck L, Paech B, Channeling the Voice of the Crowd: Applying Structured Queries in User Feedback Collection. *Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality*, Springer Nature Switzerland, 2024	4, 5, 7
(Memmer et al., 2024b)	Memmer et al. 2024b. *SMART-AGE Study Protocol: A Complex Intervention to Increase Social Participation, Physical Fitness and Health Awareness Among Older Adults*. BMC Trials, 2024	2

Chapter 2

Foundations

This Chapter provides an overview of the different user feedback types in Section 2.1 and it presents the research project SMART-AGE in Section 2.2 which is the context in which we develop our approach to collect feedback and derive requirements in Part III.

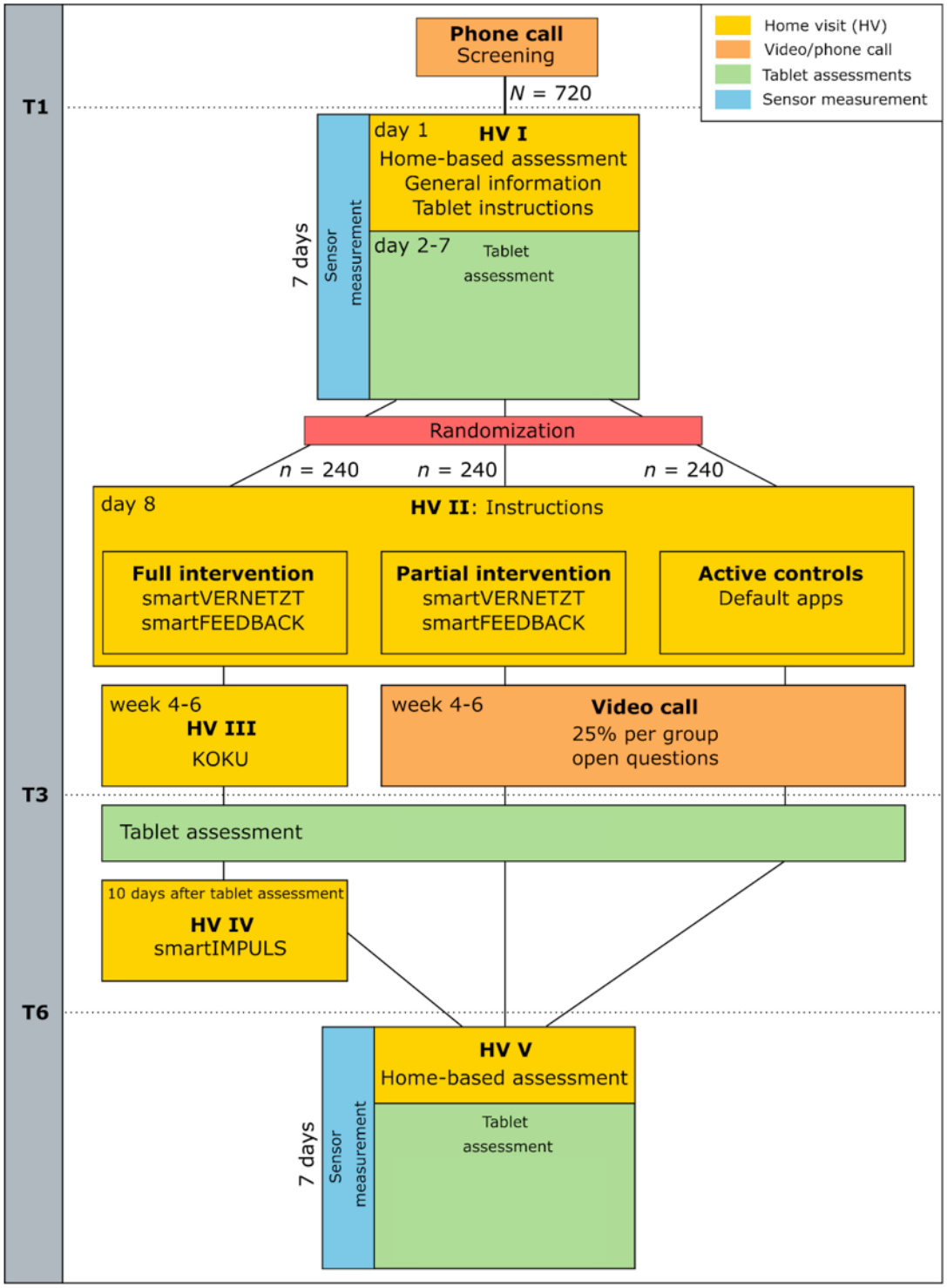
2.1 User feedback types

User feedback can be divided into feedback pushed by the user (*push*) or pulled from the user (*pull*), and feedback given with the intent to give feedback (*explicit*) or given unintentionally (*implicit*) (Maalej et al., 2009). Examples for explicit push feedback are reports about issues, bugs, enhancements or features that are sent autonomously by the users. Examples for explicit pull feedback are results of workshops, interviews and surveys. Examples for implicit push feedback are field observations and conversations with lead users. Implicit pull feedback is usage data recorded from the users.

2.2 SMART-AGE

The SMART-AGE research project ("Smart Aging in Community Contexts: Testing Intelligent Assistive Systems for Self-regulation and Co-regulation under Real-Life Conditions") is relevant for this thesis, because the treatment design (see Part III) is validated in this context. The SMART-AGE research project is a large-scale study that investigates the use of four interconnected apps for older adults who have moderate digital competence and no significant impairments. These apps aim to help older adults tackle challenges that affect their quality of life, such as loneliness, fall risks, declining health, digital exclusion, and the complexities of using digital devices. The study is structured as a three-armed, randomized controlled trial with repeated measures over a 6-month period. We refer to the older adults in the following as study partners (SP), to highlight their contribution as equal collaborators rather than passive subjects. Figure 2.1 provides an overview of the study procedure.

Figure 2.1: Overview of the study design (Memmer et al., 2024b)
N = total sample size, n = sample size in the treatment arms, CT = cognitive tasks, T1/T3/T6 = timepoint in the study



Home-based assessments are conducted at the time of study enrollment (T1) and again after 6 months (T6). Additionally, a brief online self-assessment takes place 3 months after enrollment (T3). At both T1 and T6, researchers visit the SP in their homes to conduct assessments, including a series of cognitive tasks and the use of state-of-the-

art body-fixed movement sensors to monitor physical activity for seven consecutive days. The SP are asked to complete a set of web-based questionnaires on their tablets within a week of each home visit. These questionnaires are divided into four sets of comparable length, with each set designed to take no longer than an hour to complete. The SP are instructed to complete one set of questionnaires per day. Following the second and fourth sets of questions, SP perform additional cognitive tasks on the tablet. The first set of questions, presented on the first day, includes the primary outcome measures and is repeated at T3 to provide more detailed data on the primary outcomes. One week after the initial home visit, the randomization has already taken place, and a second home visit is conducted for SP of all groups. During this visit, the tablets are modified allowing SP in both intervention groups (arm 1 and 2) access to two SMART-AGE specific apps: smartVERNETZT and SF. These groups also receive instructions and a manual on how to use these new apps. SP in the control group are given access to the standard applications on the tablet. However, detailed explanations are only given for the video chat tool. Four to six weeks after the second home visit, SP in the full intervention group additionally receive access to and are instructed in the use of the KOKU app. Approximately 10 days after the digital assessment at 3 months, the SP of the full intervention group are introduced to the SI app during home visit IV. Meanwhile, 25% of SP in both the active control and partial intervention group receive a video call four to six weeks after the second home visit to allow controlling for social enhancement effects solely due to interacting with the study team, which might affect primary outcomes.

2.2.1 Blinding

SP are always aware of which study arm they have been assigned to, as this determines whether they have access to none, two, or all four of the project apps, with the exception of the baseline assessment. To collect data, two key home visits, HV I and HV V, are conducted. It is essential that the researchers conducting these visits remain blinded to the SPs' group assignments. To maintain this blinding, the research team is divided into two separate groups: assessors and app instructors. The assessors are responsible for data collection during home visits HV I and HV V, while the app instructors manage home visits HV I, II, III, and IV. The first home visit, HV I, takes place before SP are randomized, allowing both assessors and instructors to participate in this initial stage of the study.

2.2.2 Enrollment procedures

We recruit SP aged 67 and older, who are evenly distributed across the three study arms. To identify potential SP, we obtained addresses from the city registries of Mannheim and Heidelberg and send out invitation letters with detailed study information. Interested individuals who reach out to the study team undergo a screening process via phone to determine eligibility based on specific inclusion and exclusion criteria. Those excluded from the study include individuals under the age of

67, those residing in nursing homes, those with severe cognitive limitations, no internet access, no experience with PCs/Tablets, severe medical conditions, significant visual or hearing impairments, poor knowledge of German, or those working more than 20 hours per week. As an incentive, study participants receive a tablet when they enroll in the survey, which they can keep upon completing the study.

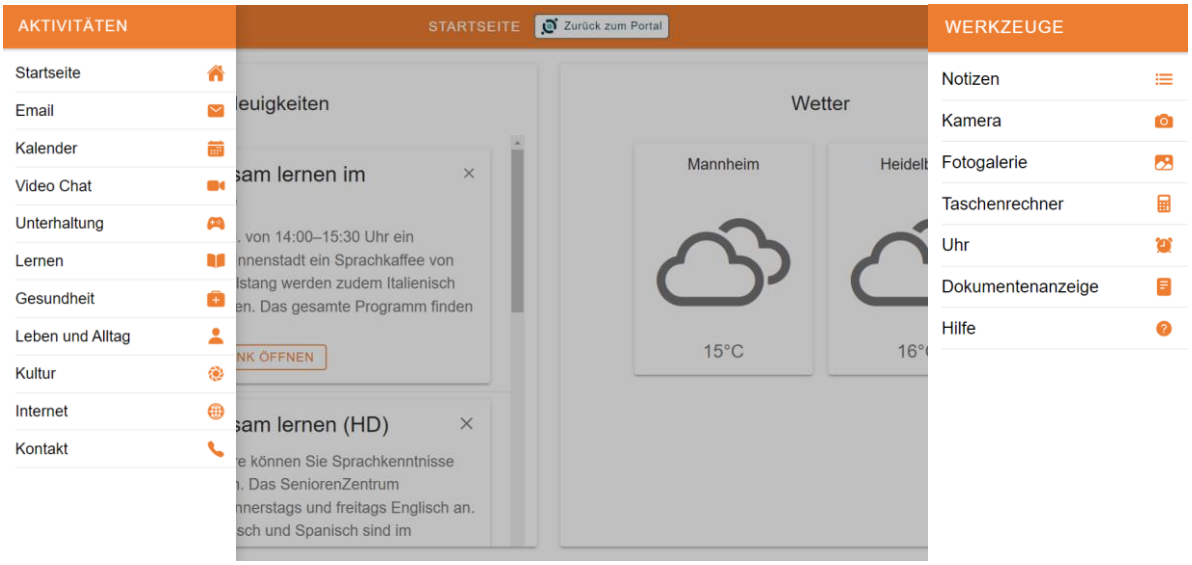
2.2.3 Apps

Pilot studies were conducted for all apps to evaluate their usability and feasibility (Memmer et al., 2024a). There is a fifth app called "Portal" which allows the access to the other apps. The development of the apps smartVERNETZT (SV) (Bührer, 2021), SF and smartIMPULS (SI) (Jakobs, 2021) was carried out by students with supervision of the Institute of Computer Science at the University of Heidelberg. KOKU (Stanmore, 2021) was initially created at the University of Manchester and translated by the SMART-AGE project team, with Reason Digital, based in Manchester, UK, handling the technical implementation. These apps are accessed using a Lenovo M10 FHD Plus tablet. The usage of the apps is monitored, and if there is no activity for two weeks, a single reminder email is sent to the tablet for each app. Alongside digital and printed user manuals and instructional videos, a helpline (accessible via email and phone) has been established to provide support.

2.2.3.1 smartVERNETZT (SV)

In collaboration with Sara Czaja (Czaja et al, 2018) and her CREATE research team, a German app-based version of PRISM (SV) was developed. The primary goal of SV is to enhance social connections, encourage local involvement, and promote participation in both digital and offline activities among older adults, ultimately reducing feelings of loneliness. The app offers a curated selection of links and applications focused on health topics, leisure activities, cognitive games, and learning opportunities. Additionally, it includes features such as an internet browser, a calendar with a reminder function, a contact book, and provides weekly updates on social participation, events, health, exercise, and technology. It also facilitates social networking through email, social networks, and video chat. Localized links were added in collaboration with municipal stakeholders. In total, SV includes information on 25 apps and 148 website links. Figure 2.2 shows a screenshot of SV.

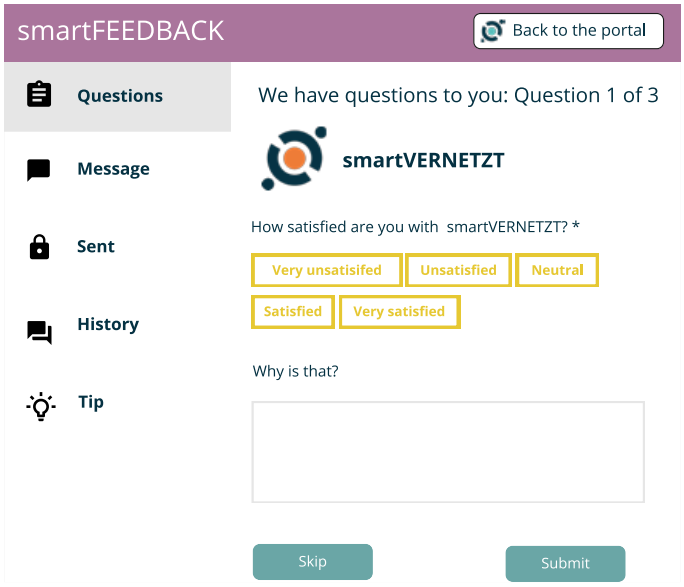
Figure 2.2: Screenshot of SV showing the sidebar with activities on the left and tools on the right. In the background news are presented and information about the weather.



2.2.3.2 smartFEEDBACK (SF)

SF allows the older adults to give feedback about the apps SV, SI and SF itself. Feedback can be given by answering questions or by sending messages. With the collected feedback, requirements are derived so that the apps can be improved. The approach to collect feedback and derive requirements is explained in Chapter III. An example screenshot is given in Figure 2.3. All screenshots of SF are presented in the appendix (from Figure B.2.21 till Figure B.2.40).

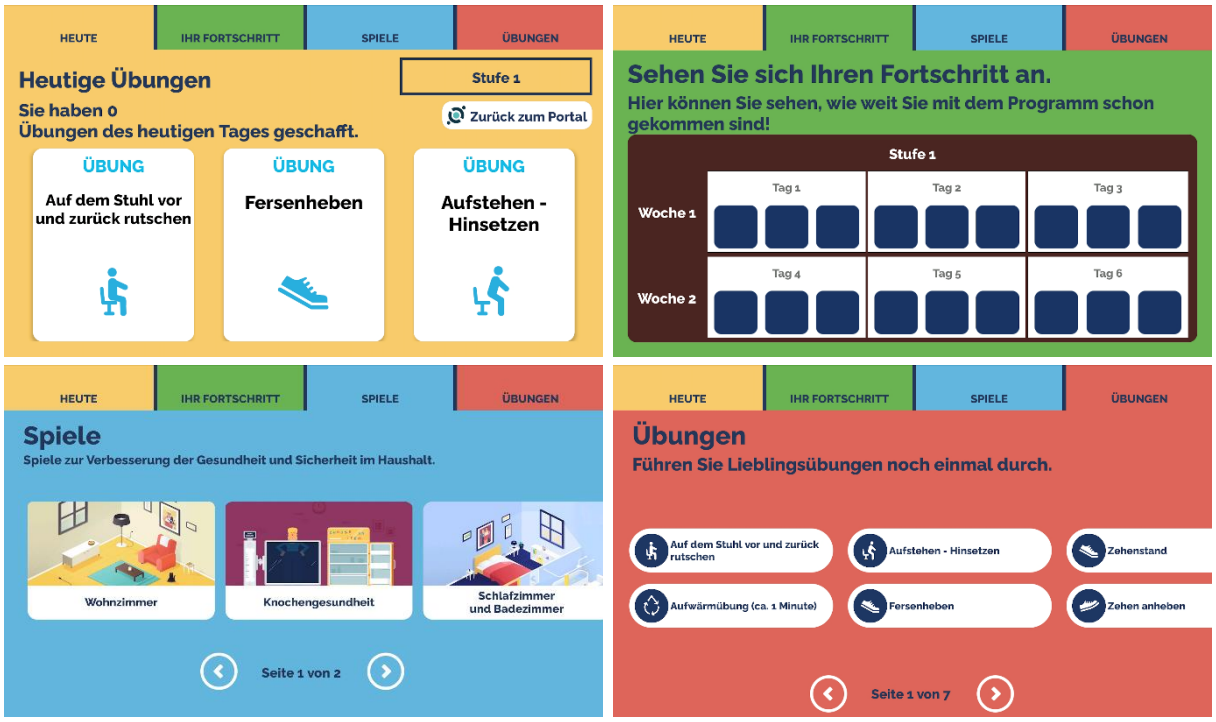
Figure 2.3: Screenshot of start page of SF



2.2.3.3 KOKU

KOKU is a personalized strength and balance training program designed for older adults (Stanmore, 2021). The app offers 26 strength and bodyweight exercises that can be performed while sitting, standing, and walking. These exercises are accompanied by instructional videos and safety tips, enabling users to engage in unsupervised home training. To begin the program, SP must complete an initial assessment, which includes several digitized questionnaires within the app, along with questions about their history of falls. Users advance through six stages, each consisting of two weeks with three training sessions per week. The app recommends three daily exercises, targeting 8-12 repetitions each. SP provide feedback in KOKU on the number of repetitions completed and their subjective sense of safety and well-being. Based on this feedback, the app adjusts the exercise progression. Additionally, KOKU includes four games aimed at promoting health literacy—two focusing on identifying potential hazards in the home (bathroom/bedroom and living room) and two on healthy eating and hydration. The app was translated into German by two native speakers with expertise in physiotherapy and medicine, ensuring that its original structure and functionality were preserved. Figure 2.4 shows screenshots of KOKU.

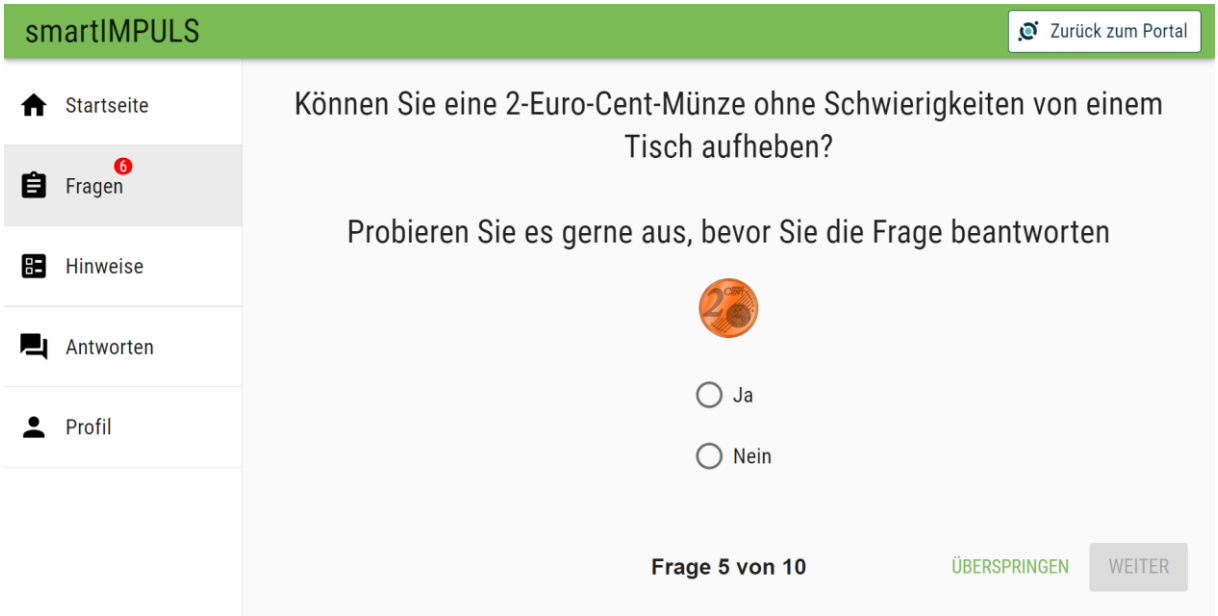
Figure 2.4: Four screenshots of KOKU showing the exercise scheduled for the current day (yellow); showing the SPs' progress (green) indicating how many exercises have been completed so far; showing games (blue) and showing the variety of exercises for users to choose from (red).



2.2.3.4 smartIMPULS (SI)

SI is designed to enhance health awareness in older adults, with a primary focus on health areas essential for maintaining independent living as long as possible, such as mobility, social participation, and nutrition. Each day, SP receive one to four questions that address their current life circumstances, daily functioning, and health status. These questions are repeated after a set period, reflecting the expected timeframe for changes in the relevant health area. If the SP doesn't use the app for one or more days, the questions accumulate, though they may be skipped if desired. The app analyzes the responses of the SP using predefined calculation rules to assess whether there is a need to raise the SP' awareness of a particular health issue. If an area of concern is identified, the app sends a notification with a suggestion. This suggestion might encourage the SP to consider addressing the issue during their next doctor's visit, or with a social worker at an appropriate counseling center, or it may provide additional information about relevant options within or outside of SMART-AGE's offerings. Ultimately, it is up to the SP to decide whether to act on the suggestion. Figure 2.5 shows a screenshot of SI.

Figure 2.5: Screenshot of SI showing a question that is asked to the SP. On the left sidebar the options "Home", "Questions", "Recommendations", "Answers" and "Profile" can be selected.



PART II

PROBLEM INVESTIGATION

Chapter 3

State of the art and practice: Collection of feedback over platforms

A Systematic Mapping Study

This Chapter contributes to the knowledge goal 1 of the thesis: *Understand the current state of the art of collecting feedback over platforms*. It presents a systematic mapping study contributing an overview of feedback platforms, which are online tools specifically designed to facilitate the collection of feedback from multiple users and allow users to interact with feedback of other users. The overview of the platforms is the basis for the treatment design described in Part III of the thesis. Section 3.1 describes the research questions (RQ). Section 3.2 describes the methodology of the publication search. Section 3.3 presents the results of the systematic mapping study. Section 3.4 gives a discussion and Section 3.5 discusses the threats to validity.

3.1 Research questions

This mapping study aims to investigate the main RQ "How is the collection of user feedback supported by platforms". This main RQ is chosen because the results can help us design a process and platform to collect user feedback in SMART-AGE. The knowledge goal 1 is refined into four RQs, which are broken down into sub-questions for more detailed investigation. The RQs and their sub-questions are listed in Table 3.1.

Table 3.1: Research questions

RQ	Description
RQ1	**What feedback is collected how and when?**
RQ1.1	What feedback is collected?
RQ1.2	How is the feedback collected?
RQ1.3	How is the collection influenced by the REengs?
RQ2	**What is the context of the feedback collection?**
RQ2.1	In what environment is the feedback collected?
RQ2.2	How many users is feedback collected from?
RQ2.3	How long is feedback collected?

RQ3	**How are key aspects evaluated and what are the results?**
RQ3.1	How is the user participation evaluated and what are the results?
RQ3.2	How is the feedback evaluated and what are the results?
RQ3.3	How is the platform evaluated and what are the results?
RQ3.4	How do the findings of the feedback affect the software?
RQ4	**How does the platform work?**
RQ4.1	What functionalities to collect feedback does the platform offer?
RQ4.2	What additional functionalities does the platform offer?
RQ4.3	Is the platform accessible publicly?

We use RQ1, RQ2, RQ3, and RQ4 because they provide a comprehensive understanding of how feedback can be collected through platforms, making it easier for other researchers to compare their platforms to existing ones.

RQ1 What feedback is collected how and when?

RQ1 explores what feedback is being gathered (*RQ1.1*), how it is collected (*RQ1.2*) and how the collection of feedback is influenced by the REengs (*RQ1.3*). *RQ1.1* is important because feedback could be collected in different forms, e.g. as freetext or in a structured form. R*Q1.2* is relevant because there could be different methods to collect feedback, such as relying on messages from users or asking users questions. Further, feedback could be collected either once or multiple times. *RQ1.3* is important because influence of REengs could have any forms (e.g. interacting with users or moderating discussions) which could impact the feedback collected or the user participation.

RQ2 What is the context of the feedback collection?

The second research question, *RQ2*, inspects the context of feedback collection. It analyzes in which environment the feedback is collected (*RQ2.1*), how many users give feedback (*RQ2.2*) and how long the feedback is collected (*RQ2.3*). Regarding *RQ2.1* the environment refers to the institutional or organizational setting in which feedback is collected, the product about which the feedback is collected and the users that use the product. This research question is important because the environment can affect how feedback is collected. In an industrial context, the goal is often to generate more profit, while in research, the focus is freer from capitalist goals, allowing for more flexibility in exploring feedback collection. *RQ2.2* is important to assess whether the feedback collection process is practical and scalable for a large number of users. *RQ2.3* is important because it shows whether users are willing to give feedback over a longer period of time.

RQ3 How are key aspects evaluated and what are the results?

RQ3 investigates the evaluation of key aspects and results. The key aspects are the user participation (*RQ3.1*), the feedback received (*RQ3.2*) the platform itself (*RQ3.3*), as well as the findings of the feedback that affect the software (*RQ3.4)*. *RQ3.1* is important because it could be that a lot of users use the platform, but only few give feedback.

RQ3.2 is important because the feedback needs to be in some form helpful for the REengs and it is therefore interesting to know which aspects are evaluated and what the results are. *RQ3.3* gives insights how a platform can be evaluated. This is important as user satisfaction with the platform can influence the feedback and user participation. *RQ3.4* is important because incorporating the findings into the software is necessary to improve it.

RQ4 How does the platform work?

RQ4 investigates the platform regarding the functionalities that the platform offers to collect feedback (*RQ4.1*), the functionalities beyond feedback collection (*RQ4.2*) and regarding whether the platform is publicly accessible (*RQ4.3)*. RQ4.1 is important because there could be different ways to submit feedback (e.g. through comments or posts) and there could be ways that facilitate submitting feedback (e.g. using speech messages instead of text). *RQ4.2* is important because there could be functionalities that are used to motivate users to give feedback or other functionalities that facilitate the use of the platform for the user (e.g. providing an overview of feedback or allowing for export of the feedback). *RQ4.3* is interesting because it means that everyone could use the platform to collect feedback.

3.2 Methodology

This Section outlines the steps of our mapping study and describes our inclusion criteria. To address the research questions listed in Section 3.1, we conducted a systematic mapping study, which adheres to the guidelines proposed by (Kitchenham & Charters, 2007). These guidelines entail having:

- C1 - a well-defined search strategy
- C2 - a search string with alternative terms combined by ANDs and ORs
- C3 - a broad range of search sources
- C4 - explicit inclusion or exclusion criteria
- C5 - strict documentation of the search

Our systematic mapping study follows all the mentioned guidelines (C1 – C5). To understand when we follow which guideline, we denote the guidelines with "Cx" in the following.

We reviewed the papers according to a structured, defined search strategy (C1). The search strategy consists of the definition of the search string (C2), the identification of search sources (C3), the definition of the inclusion criteria (C4), the conduction of a term-based search and the conduction of forward and backward snowballing. We document the conduction of the search in detail by describing every step and result in form of text, tables and graphics (C5).

3.2.1 Generation of the search string

We first construct a prototypical search string (C2) based on the main RQ "How is the collection of user feedback supported by platforms" (Section 3.2.1.1). We then expand this prototypical search string by identifying alternatives for the search terms to also find relevant papers that do not use the exact same words (Section 3.2.1.2). Finally, we check the recall of the resulting search string and adapt it iteratively, so that the number of results is manageable (Section 3.2.1.3).

3.2.1.1 Constructing a prototypical search string

To construct a prototypical search string, we split the search question "How is the collection of user feedback supported by platforms" into four main *root search terms*: **collect**, **user**, **feedback**, and **platform**. Our first prototypical search string in Table 3.2 is generated by concatenating these root terms with the AND operator.

Table 3.2: Prototypical search string

	AND	**AND**	**AND**
collect	**user**	**feedback**	**platform**

3.2.1.2 Identifying alternatives for the root search terms

We try to increase the likelihood to find other relevant papers that do not use the exact same words but alternative words with the same meaning (C2). We take advantage of the fact that we already know of 7 relevant articles. These *known relevant articles* are all feedback platforms described in paper "Crowd-based requirements elicitation via pull feedback: method and case studies" (Wouters et al., 2022).

We identify alternative words for the root search terms in two ways. The first way is to search for the most frequent words of the known relevant articles and check whether these words can be used as alternatives for the root search terms. The second way is to search for alternative terms for the root search terms inside the known relevant articles with a lexical similarity measure.

Identification of most frequent words in the known relevant articles

We analyze the 20 most frequent words of each known relevant article and then check manually, whether these words can be used as alternatives for one of the root search terms. The frequency of words was determined by reading in the PDF file of the article and counting identical words independently of plural or singular with the help of the python NLTK library[1]. The detailed result along with reasons why certain words were used as alternatives to root search terms can be found in appendix (Table A.1.2)**.** The

[1] https://www.nltk.org/

resulting search string can be seen in Table 3.3. Note that we summarized the words "crowdsourced" and "crowdsourcing" to the wildcard version "crowdsourc*" to include different variations of these words. Also "negotiat*" is used to include both "negotiation" and "negotiate". For "community" the plural version "communities" is also added.

Table 3.3: Search string after identifying alternatives for the root search terms (green background) by analyzing the 20 most frequent words of the known relevant articles.

		AND	**AND**	**AND**
	collect	**user**	**feedback**	**platform**
OR	crowdsourc*	stakeholder	requirement	software
OR	negotiat*	crowd	idea	tool
OR	request	participant	post	
OR	elicit	visitor		
OR		employee		
OR		community		
OR		communities		

Identification of similar words to the root search terms in the known relevant articles

As a second way of identifying alternatives for the root search terms, we use the lexical database WordNet[2] and python to identify words inside the known relevant articles that are similar to the root search terms. We search for similar words inside the known relevant articles for each root search term. We compare words using path similarity with a threshold of 0.3, which means that words in the known relevant articles must share at least 30% similarity with the corresponding root search term. We use this threshold because it provides a balanced value that captures a wider range of semantically related terms without being overly restrictive, ensuring that also not perfectly matching terms are included in our analysis. We search for alternative terms inside the known relevant articles and not in the internet, to ensure that the alternative terms are contextually appropriate and belong to the domain-specific language. The detailed result along with reasons why certain similar words were used as alternatives to root search terms can be found in Table A.1.3. The refined prototypical search string can be found in Table 3.4.

[2] https://wordnet.princeton.edu/

Table 3.4: Refined prototypical search string after identifying similar words to the root search terms (green background) inside the known relevant articles.

		AND	AND	AND
	collect	**user**	**feedback**	**platform**
OR	elicit	stakeholder	requirement	software
OR	crowdsourc*	crowd	idea	tool
OR	negotiat*	participant	post	application
OR	request	visitor	review	
OR	gather	employee	answer	
OR	ask	community	comment	
OR		communities	rating	
OR		client		
OR		person		
OR		individual		
OR		customer		

Setting the scope of the search string

We decide to not limit our search to the title, because some titles of our known relevant articles do not contain the root search terms or their alternatives. For example, "CrowdRE in a Governmental Setting: Lessons from Two Case Studies" (Wouters et al., 2021) only mentions "crowd". A mapping to the root search terms "collect", "feedback" or "platform" cannot be established. Furthermore, searching the full text is not possible for search sources like WebOfScience[3]. Therefore, we decide to set the scope of our search string to the abstract of possible relevant papers. To be sure that our known relevant articles would be found when searching for their abstracts, we manually checked all their abstracts and compared them to the search string.

3.2.1.3 Adapting the prototypical search string to allow for a manageable amount of search results

After constructing the search string and before starting the actual search, we conduct an exploratory search on IEEE to test whether our search string yields a manageable number of search results. We do it on IEEE, because it provides the most flexibility regarding the search scope. We test whether our search string yields not more than 500 results, as this is our limit for a manageable amount of search results per search source. If the number of search results is higher than 500, we have to make the search term more concrete or change the scope of the search string (e.g. scoping some terms to the title only instead of the abstract).

[3] https://webofscience.com/

As a first step to check whether our search string yields a manageable amount of search results, we convert the search string in Table 3.4 to the IEEE command format[4]. The resulting search term is listed in Listing 1.

Listing 1: Refined prototypical search term in IEEE command search format

```
("Abstract": collect OR "Abstract": elicit OR "Abstract": crowdsourc* OR
"Abstract": negotiat* OR "Abstract": request OR "Abstract": gather OR
"Abstract": ask)

AND ("Abstract": user OR "Abstract": stakeholder OR "Abstract": crowd OR
"Abstract": participant OR "Abstract": visitor OR "Abstract": employee OR
"Abstract": community OR "Abstract": communities OR "Abstract": client OR
"Abstract": person OR "Abstract": individual OR "Abstract": customer)

AND ("Abstract": feedback OR "Abstract": requirement OR "Abstract": idea OR
"Abstract": post OR "Abstract": review OR "Abstract": answer OR "Abstract":
comment OR "Abstract": rating)

AND ("Abstract": platform OR "Abstract": software OR "Abstract": tool OR
"Abstract": application)
```

With this search term we execute a search on IEEE on 01.06.2023. The number of results was 9465. As this is higher than 500, we subsequently try to limit the number of search results by analyzing false positive search hits and adapting the search terms accordingly. When examining the search results, we noticed that words like "ask" or "answer" produce too many irrelevant hits. Furthermore, words like "post" or "rating" are found in other words, such as "poster", "post-hoc" or "incorporating" and thus produce false hits. Also, the word "application" is ambiguous and occurs in "application field", for example. All of the mentioned words were therefore removed from the search term. There were also articles that described platforms in the context of blockchains, articles that investigate fake reviews and code reviews, articles that analyze training platforms in sports and furthermore articles that focus on machine learning, deep learning, sentiment analysis or classification. These articles were filtered by blacklisting words either in the title or abstract. We blacklist in title and abstract, because it could be that these words in either one of these fields when searching other sources.. Finally, the word "idea" also appeared in phrase "the idea of". This phrase was also removed by blacklisting. In Table 3.5 the refined prototypical search string after the elimination of the mentioned ambiguous or general words is shown.

Table 3.5: Refined prototypical search string after eliminating ambiguous or too general words (red font) and after blacklisting specific words or phrases (green background).

		AND	AND	AND	AND NOT
	collect	**user**	**feedback**	**platform**	Title: blockchain
OR	crowdsourc*	stakeholder	requirement	software	Title: machine learning

[4] https://ieeexplore.ieee.org/Xplorehelp/searching-ieee-xplore/command-search

OR	negotiat*	crowd	idea	tool	Title: deep learning
OR	request	participant	~~post~~	~~application~~	Title: classification
OR	elicit	visitor	review		Title: sentiment analysis
OR	gather	employee	~~answer~~		fake review
OR	~~ask~~	community	comment		code review
OR		communities	~~rating~~		training platform
OR		client			the idea of
OR		person			
OR		individual			
OR		customer			

The elimination of ambiguous and too general words resulted in 3690 search results. The amount of search results could thus be reduced by 5775 hits. Nevertheless, the amount of search results can not yet be examined manually because it is still too large. We couldn't identify any other ambiguous words and continuing to blacklist would only remove a handful of articles. Therefore, we decided to limit the search on the research area "requirements engineering". To achieve that, the term "requirement* engineering" is added. The asterix is used, because some people may use the terminology "requirement engineering" instead of "requirements engineering". The resulting final search term is shown in Table 3.6 and in IEEE format in Listing A.1.1.

Table 3.6: Final search term after limiting research field to "requirement* engineering" (green background).

		AND	**AND**	**AND**	**AND NOT**	**AND**
	collect	**user**	**feedback**	**platform**	Title: blockchain	requirement* engineering
OR	crowdsourc*	stakeholder	requirement	software	Title: machine learning	
OR	negotiat*	crowd	idea	tool	Title: deep learning	
OR	request	participant	review		Title: classification	
OR	elicit	visitor	comment		Title: sentiment analysis	
OR	gather	employee			fake review	
OR		community			code review	
OR		communities			training platform	
OR		client			the idea of	
OR		person				
OR		individual				
OR		customer				

The final search term produced 498 search results. This amount of search results can be checked manually, so this search term will be used for the mapping study.

3.2.2 Specification of the search sources

To have a high coverage of the research field we use four search sources (C3). We use the scientific associations IEEE[5], ACM[6], SpringerLink[7] and WebOfScience[8] as search sources, because a large part of scientific literature of IT can be found there. We cannot search on Scopus[9], because Heidelberg University has no active access to it. SpringerLink does not allow to search in the abstract only. It always searches in title, abstract and full text. We used the search source nevertheless and check only the first 500 results of the ~12.000 search results.

3.2.3 Definition of the inclusion criteria

Following the recommendations of (Kitchenham & Charters, 2007) we define seven explicit inclusion criteria (C4). These inclusion criteria are listed in Table 3.7 All of the known relevant articles (see Table A.1.1) fulfill *I1 - I7*.

Table 3.7: Inclusion criteria

Nr.	Inclusion criterium
I1	Title suggests relevance to main RQ
I2	Abstract suggests relevance to main RQ
I3	Article is available online
I4	Article is not older than 16 years (2008 - 2024)
I5	Article is written in German or English
I6	Article describes how the collection of user feedback is supported by a platform
I7	Article describes how the exchange of feedback among the users is supported by the platform

I1 and *I2* ensure that the title and abstract of the article indicate relevance to the main RQ. *I3* checks whether the article is available online. *I4* limits the time of the publication date to the last 16 years. The original timeframe was 15 years, but the search was repeated a year later, extending the timeframe by an additional year. *I5* is important because the author only understands articles that are written in German or English. *I6* and *I7* are necessary to only include articles that comply to our definition of a feedback platform. With exchange of feedback (I7) we mean that the platform allows users to interact with the feedback of other users.

[5] https://ieeexplore.ieee.org/

[6] https://dl.acm.org/

[7] https://link.springer.com/

[8] https://www.webofscience.com/

[9] https://www.scopus.com/

3.2.4 Conduction of the term-based search

We conduct a term-based search with the search term in Table 3.6 on 07.08.2024 at the specified search sources. We then apply the inclusion criteria to the found articles.

3.2.5 Conduction of forward and backward snowballing

The known relevant articles and the results of the term-based search are used to conduct forward and backward snowballing with the same inclusion criteria to expand the number of search results and find even more relevant articles.

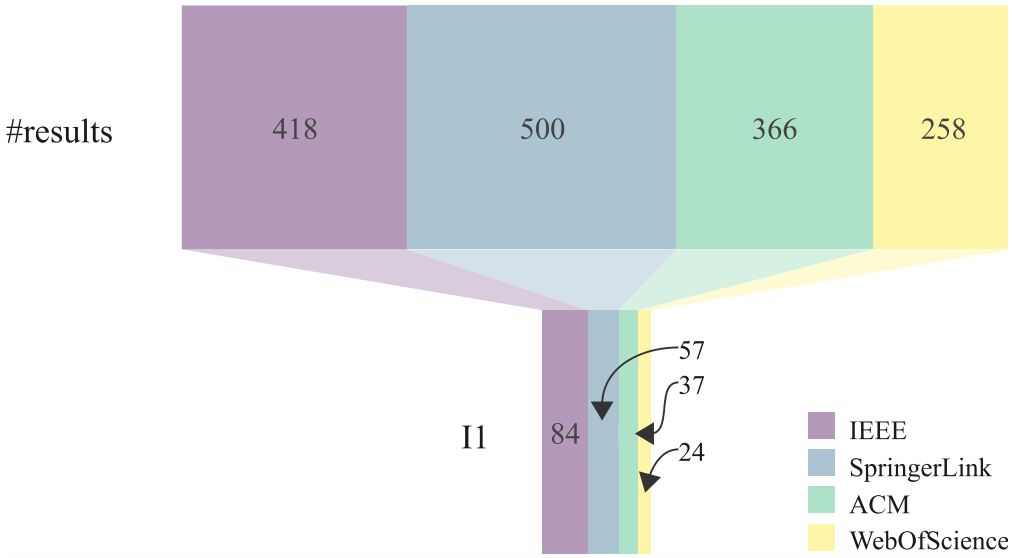
3.3 Results

In Section 3.3.1 the results of the term-based search are presented. In Section 3.3.2 the results of the forward and backward snowballing are presented. Section 3.3.3 list all the relevant articles that result from the search. Section 3.3.4 gives a literature overview over the relevant articles. Section 3.3.5 summarizes the findings of the relevant articles in the form of a synthesis and gives answers to the research questions.

3.3.1 Term-based search

Figure 3.1 shows how many articles of the term-based search were included and excluded based on their title (I1).

Figure 3.1: Included and excluded articles by I1 (term-based search)



■ **IEEE** There were 418 search results, of which 334 articles were excluded, because their titles did not suggest relevance. 84 articles were included (*I1*).

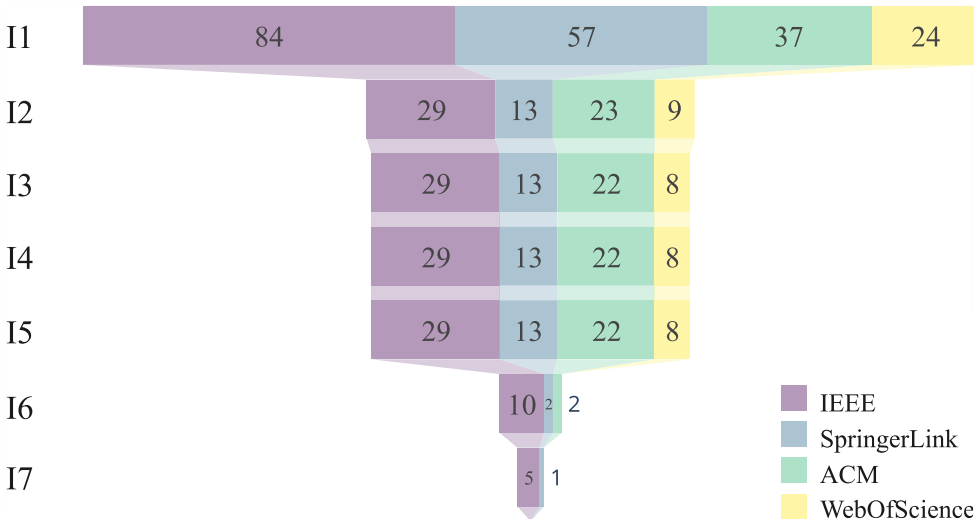
■ **SpringerLink** The first 500 search results were checked. 443 articles could not be included, based on their title. 57 were included.

■ **ACM** There were 366 search results. 329 articles could not be included, because of their title. 37 articles could be included.

■ **WebOfScience** There were 258 results. 234 articles did not fulfill *I1*, 24 did.

In total 1542 articles were checked based on their title. Figure 3.2 gives an overview of the application of all inclusion criteria *I1* – *I7* on the search results.

Figure 3.2: Included articles by I1 - I7 (term-based search)



■ **IEEE** The abstracts of 29 of the 84 included papers suggested relevance (*I2*). All of the 29 papers were available (*I3*), not older than 16 years (*I4*) and written in English (*I5*). 10 articles describe how the collection of user feedback is supported by a platform (*I6*). 5 articles out of the 10 also describe how the exchange of feedback is supported by the platform (*I7*). These 5 articles are the known relevant articles (Renzel et al., 2013; Snijders et al., 2015; Sharma and Sureka, 2018; Menkveld et al., 2019; Wouters et al., 2021)**.** The articles that pass *I6* but not *I7* are (Seyff et al., 2010; Vijayan et al., 2017; Stade et al., 2017; Oriol et al., 2018; Saphira and Rusli, 2019)**.** These do not support the exchange about feedback (*I7*), because users can always see only their own feedback. The articles are listed in Table 3.8 together with their titles.

■ **SpringerLink** 13 of the already included 57 articles were further included, because of their abstract (*I2*). All of the 13 papers passed the inclusion criteria *I3* - *I5*. The 2 articles (Kolpondinos and Glinz, 2020) and (Wüest et al., 2019) were included, because of *I6*. (Wüest et al., 2019) does not pass *I7*, because it only collects feedback, but it does not support the exchange of the feedback with other users. (Kolpondinos and Glinz, 2020) passes *I7*, but it is a known relevant article.

■ **ACM** 23 articles could further be included, because of their abstract (*I2*). From these 23 articles, 22 articles passed *I3* - *I5*. 2 articles fulfilled *I6*. One article (Seyff et al., 2010)

was already found through IEEE. The other article is (Wehrmaker et al., 2012). This article does not pass *I7*, because it does not support the exchange about feedback (*I7*).

WebOfScience 9 of the included articles fulfilled *I2*. 8 articles fulfilled *I3*, *I4* and *I5*. No article fulfilled *I6* or *I7*.

No new relevant articles that pass *I1 - I7* were identified. The identified 6 articles that pass *I1 - I7* were known relevant articles. Table 3.8 shows all 7 identified articles that pass *I6* but not *I7*.

Table 3.8: Identified articles through term-based search that pass *I6* and not *I7*.

Source	Ref.	Title	*I6*	*I7*
IEEE	(Saphira and Rusli, 2019)	Towards a gamified support tool for requirements gathering in Bahasa Indonesia	✓	X
IEEE	(Oriol et al., 2018)	FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring	✓	X
IEEE	(Vijayan et al., 2017)	Collaborative requirements elicitation using elicitation tool for small projects	✓	X
IEEE	(Stade et al., 2017)	Providing a user forum is not enough: First experiences of a software company with CrowdRE	✓	X
IEEE	(Seyff et al., 2010)	End-user requirements blogging with iRequire	✓	X
Springer-Link	(Wüest et al., 2019)	Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use	✓	X
ACM	(Wehrmaker et al., 2012)	ConTexter feedback system	✓	X

3.3.2 Snowballing

Our term-based search yielded only known relevant articles. To expand our set of relevant articles, we performed backward and forward snowballing based on the known relevant articles. Figure A.1.1 in the appendix shows in detail how many articles are found through backward and forward snowballing per known relevant article and also how many articles pass the inclusion criteria.

(Wouters et al., 2021)

Backwards: We found 32 references, from which 9 passed *I1*, and 4 passed *I2 – I7*. All 4 articles are known relevant articles (Renzel et al., 2013; Snijders et al., 2015; Menkveld et al., 2019; Kolpondinos and Glinz, 2020).

Forwards: When forward snowballing the article, 6 articles were found, of which 1 passed *I1*, but not *I2*.

(Kolpondinos & Glinz, 2020)

Backwards: We found 64 references of which 20 passed I1 and 10 passed *I2 – I5*. 4 articles passed *I6*. 2 of the 4 articles are known relevant articles (Fernandes et al., 2012; Snijders et al., 2015). One article was already identified over the term-based search (Oriol et al., 2018). One new relevant article passed *I7* (Lohmann et al., 2009). The new relevant articles are listed in Table 3.10.

Forwards: Forward snowballing resulted in 44 articles, of which 7 passed *I1*, 3 passed *I2* and 2 passed *I3 – I7*. These were known relevant articles (Menkveld et al., 2019; Wouters et al., 2021).

(Fernandes et al., 2012)

Backwards: We found 37 references. 8 passed *I1* and 7 passed *I2 – I5*. 3 passed *I6*. 1 of the 3 was already found in the term-based search (Seyff et al., 2010). The other two articles also passed *I7* and are new relevant articles (Yang et al., 2008; Laporti et al., 2009).

Forwards: We found 221 results. 25 articles passed *I1* and 11 passed *I2 – I5*. 3 passed *I6* and *I7*. 2 of the 3 are known relevant articles (Snijders et al., 2015; Kolpondinos and Glinz, 2020). One article (Vogel et al., 2020) is a new relevant article.

(Snijders et al., 2015)

Backwards: There were 26 results through backward snowballing. 11 articles passed *I1*. 4 articles passed *I2 – I5*. 2 articles passed I6 and I7. Both articles are known relevant articles (Fernandes et al., 2012; Renzel et al., 2013).

Forwards: Forward snowballing resulted in 106 results. 24 articles passed *I1* and 10 passed *I2 – I5*. 7 articles pass *I6* and 4 *I7*. One of the 4 is already found through snowballing (Vogel et al., 2020b). Three are known relevant articles (Menkveld et al., 2019; Kolpondinos and Glinz, 2020; Wouters et al., 2021).

(Renzel et al., 2013)

Backwards: There were 6 results. None of the results passed *I1*.

Forwards: We found 84 articles, of which 16 passed *I1* and 6 passed *I2 – I5*. 4 passed *I6* and *I7*. All articles are known relevant articles (Snijders et al., 2015; Sharma and Sureka, 2018; Menkveld et al., 2019; Wouters et al., 2021).

(Menkveld et al., 2019)

Backwards: We found 22 articles. 8 of the articles passed *I1* and 4 articles passed *I2* – *I5*. 3 passed *I6* and *I7*. The 3 articles are known relevant articles (Renzel et al., 2013; Snijders et al., 2015; Kolpondinos and Glinz, 2020).

Forwards: Forward snowballing resulted in 12 articles found. 4 articles passed *I1* and 1 article passed *I2* – *I7*. This article is also a known relevant article (Wouters et al., 2021).

(Sharma & Sureka, 2018)

Backwards: We found 28 articles. 10 passed *I1* and 4 articles passed *I2* – *I5*. 1 article passed *I6* and *I7*. This article is a known relevant article (Renzel et al., 2013).

Forwards: We found 19 articles, of which 8 passed *I1* and 2 passed *I2* – *I5*. 1 passed *I6*. This article was already found through snowballing (Rietz, 2019).

In total, we checked 707 articles for relevance during snowballing. Table 3.9 shows the newly identified articles that passed *I6*, but not *I7*. Table 3.10 gives an overview over all new relevant articles that were identified through snowballing.

Table 3.9: Identified new articles through snowballing (passing *I1-I6*, but not *I7*). Excluding duplicates.

Source	Ref.	Title	*I6*	*I7*
(Snijders et al., 2015)	(Rietz, 2019)	Designing a conversational requirements elicitation system for end-users	✓	X
	(Haug et al., 2023)	Scalable Design Evaluation for Everyone! Designing Configuration Systems for Crowd-Feedback Request Generation	✓	X

Table 3.10: Identified relevant articles through snowballing (passing *I1-I7*). Excluding known relevant articles and duplicates.

Source	Ref.	Title	*I6*	*I7*
(Kolpondinos & Glinz, 2020)	(Lohmann et al., 2009)	A Web Platform for Social Requirements Engineering	✓	✓
(Fernandes et al., 2012)	(Yang et al., 2008)	WikiWinWin: A Wiki based system for collaborative requirements negotiation	✓	✓
	(Laporti et al., 2009)	Athena: A collaborative approach to requirements elicitation	✓	✓
	(Vogel et al., 2020b)	Leveraging the internal crowd for continuous requirements engineering - Lessons learned from a design science research project	✓	✓

3.3.3 Relevant articles

After checking 2249 articles for relevance through a term-based search and snowballing, we could identify 4 new relevant platforms (see Table 3.10). Together with the 7 known relevant articles (see Table A.1.1), we have 11 relevant articles in total. We list all relevant articles for simpler reading in the joined Table 3.11 along with their platform names. These relevant articles will be the base for the literature overview in Section 3.3.4 and the synthesis in Section 3.3.5.

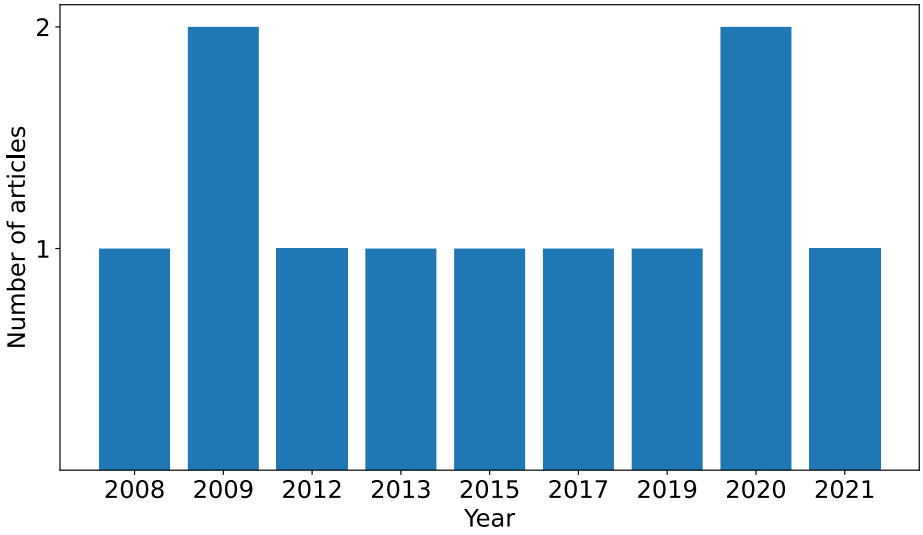
Table 3.11: All relevant articles

Ref.	Title	Platform
(Wouters et al., 2021)	CrowdRE in a Governmental Setting: Lessons from Two Case Studies	KMar-Crowd
(Kolpondinos and Glinz, 2020)	GARUSO: a gamification approach for involving stakeholders outside organizational reach in requirements engineering	GARUSO
(Menkveld et al., 2019)	User story writing in crowd requirements engineering: The case of a web application for sports tournament planning	Tournify
(Sharma & Sureka, 2018)	CRUISE: A platform for crowdsourcing Requirements Elicitation and evolution	CRUISE
(Snijders et al., 2015)	REfine: A gamified platform for participatory requirements engineering	REfine
(Renzel et al., 2013)	Requirements Bazaar: Social requirements engineering for community-driven innovation	Bazaar
(Fernandes et al., 2012)	iThink: A game-based approach towards improving collaboration and participation in requirement elicitation	iThink
(Lohmann et al., 2009)	A Web Platform for Social Requirements Engineering	WPFSRE
(Yang et al., 2008)	WikiWinWin: A Wiki based system for collaborative requirements negotiation	WikiWinWin
(Laporti et al., 2009)	Athena: A collaborative approach to requirements elicitation	Athena
(Vogel et al., 2020b)	Leveraging the internal crowd for continuous requirements engineering - Lessons learned from a design science research project	CrowdCore

Figure 3.3 shows the distribution of publication years for the relevant articles. The publication rate for relevant articles was relatively constant throughout the years 2008 – 2021 with 1 publication per year. The years 2009 and 2020 have 2 publications. It is

notable that no relevant articles were found for the years 2022, 2023 and 2024 even though we did not exclude these years in the term-based search or during the snowballing.

Figure 3.3: Distribution of publication years of relevant articles



3.3.4 Literature overview

Table A.2.1 presents the literature overview of the relevant articles in a tabular format. In the literature overview, a row is created for each relevant article. Each cell in the row contains aspects regarding the content of the publication. It includes the *background, motivation*, *research questions* and *problems*, *principal idea*, as well as the *contribution* of the article. The context and motivation provides background information and describes the motivation of the research direction. The research questions and problems column explains what will be answered or solved in the article, as well as problems that occurred during the research. To reduce the size of the literature overview table, we only present results of the articles in the synthesis under RQ3. The principal idea column contains a description of the research process and the contribution column describes how the article helps others in their work.

3.3.5 Synthesis

The synthesis presents the findings of the individual relevant articles. The complete synthesis matrix is in the appendix and it is split into Table A.2.2 and Table A.2.3. Table A.2.2 addresses RQ1 and RQ2, whereas Table A.2.3 addresses RQ3 and RQ4. We have created a condensed synthesis matrix with Table 3.12 In the following, we present results to the research questions by explaining the condensed synthesis matrix and enriching the explanation with important details of the complete synthesis matrix in the appendix (Table A.2.2 and Table A.2.3). We also summarize each research question and provide key takeaways for the sub-research questions when the text is more extensive.

Table 3.12: Condensed synthesis matrix.
US=User story, US*=US, User scenario, Use case

Platform	RQ1.1		RQ1.2		RQ1.3	RQ2.1	RQ2.2			RQ2.3	RQ3.1	RQ3.2	RQ3.3	RQ4.1				RQ4.2		RQ4.3
	Main feedback: Free text	Main feedback: Template	Feedback collection in multiple phases	Feedback collection in one phase	REengs influence feedback collection	G=Government, R=Research, C=Commercial	Invited users	Accessing users	Contributing users	Feedback collection duration in days	Number of main feedback	Evaluation of feedback	Evaluation of the platform	Submitting main feedback	Commenting	Voting/Scoring/ Rating	Other	Gamification	Other	Accessible for public
S-Sys KMar-Crowd		US	X		X	G	478	135	60	33	32	X	X	X	X	X		X	X	
V-Sys KMar-Crowd		US	X		X	G	2393	385	130	56	78	X	X	X	X	X		X	X	
GARUSO		US		X	X	R		726	32	92	56		X	X	X	X	X	X	X	
Tournify		US		X	X	C	337	157	39	35	57	X	X	X	X	X			X	
CRUISE	X			X		R	37	18	18			X	X	X	X	X			X	
REfine	X			X	X	C		19	19	35	21		X	X	X	X		X	X	
Bazaar		US	X		X									X	X	X			X	X
(1) iThink	X			X		C		7	7		10		X	X	X	X		X	X	
(2) iThink	X			X		C		17	17		22		X	X	X	X		X	X	
WPFSRE	X			X	X									X	X	X	X		X	
Athena		US*	X			R		6	6	1/6				X	X				X	
WikiWinWin	X			X		R		6	6	1/6	62		X	X	X	X			X	
CrowdCore	X		X		X	C							X	X	X	X			X	

The condensed synthesis matrix includes the platforms in the left column. If a platform appears twice, it indicates that the platform was evaluated in two different studies. In the columns to the right, the research questions are listed, and crosses are marked if aspects relevant to answering the research questions apply.

——————— RQ1 What feedback is collected how and when? ———————

——————————— RQ1.1 What feedback is collected? ———————————

We distinguish between main feedback and meta-feedback in feedback collection. Main feedback is feedback that does not reference existing feedback, while meta-feedback is feedback that references existing feedback (e.g. a comment). The main feedback is always either collected in an unstructured form as freetext or in a structured form as a template. A template means, that the main feedback content is split into parts (e.g. the role, action and reason of a user story are split into three different input fields). 6 platforms collect feedback as freetext (CRUISE, REfine, iThink, WPFSRE, WikiWinWin and CrowdCore). The other 5 platforms collect main feedback as a template. Templated main feedback is always a user story except for Athena where main feedback is collected as a combination of user scenarios and use cases. The meta-feedback is either collected through comments or through a form of voting, scoring or rating (see Table 3.12: RQ4.1). Three platforms have special forms of meta-feedback. GARUSO allows to submit meta-feedback in the same form as main feedback underneath another main feedback. REfine allows the users to create branches of the main feedback, so that different aspects of the main feedback can be isolated. WPFSRE allows the users to mark relations between different main feedback.

Key takeaway: Feedback can be distinguished into main and meta-feedback, which is meta-feedback references main feedback. Platforms collect main feedback either in an unstructured form as freetext or in a templated form. Meta-feedback is collected via comments, voting, scoring, rating, or specialized methods such as branching.

——————————— RQ1.2 How is the feedback collected? ———————————

All platforms collect their main and meta-feedback as push feedback only. None of the platforms uses questions to ask for feedback (pull feedback). During feedback collection it can be distinguished whether feedback is collected in multiple phases or only in a single phase. Four platforms (KMar-Crowd, Bazaar, Athena and CrowdCore) collect feedback over multiple phases, while the other platforms collect feedback in a single phase. KMar-Crowd starts with collecting main feedback in a first phase. Then, a summary about the collected feedback is provided by the REengs, and subsequently the main feedback is commented upon or voted on by the users. For Bazaar there are also multiple phases. There is an initial phase where main feedback is collected. Then, this main feedback is discussed through commenting and voting, and subsequently, it is again refined through commenting and voting. For Athena, main feedback is collected first as user stories. Then these stories are converted by the users into

scenarios, and subsequently, the scenarios are converted by the users into use cases. Comments can be made at each phase of this process. For CrowdCore, main feedback is also collected first. Then the product owner decides which main feedback progresses to the voting phase. In the voting phase, the users vote on feedback. After the voting phase there is a decision phase, where the product owner selects main feedback and the users can comment on it.

Key takeaway: All platforms rely on push feedback to collect both main and meta-feedback, with no use of direct questioning. While the most platforms collect feedback in a single phase, few platforms employ a multi-phase approach for feedback collection.

——————— RQ1.3 How is the collection influenced by the REengs? ———————

The REengs influence the collection of feedback on seven platforms (KMar-Crowd, GARUSO, Tournify, Refine, Bazaar, WPFSRE and CrowdCore). Regarding KMar-Crowd and GARUSO the REengs write summaries and present them to the users. Regarding Tournify, the REengs comment on some of the main feedback and also label main feedback as in development or done. Regarding REfine, the REengs provision guidelines for feedback collection, they delete irrelevant needs and also sent weekly updates to improve the activity of users. Regarding Bazaar, the REengs comment on main feedback. Regarding WPFSRE, the REengs supervise and moderate discussions and regarding CrowdCore, the product owner motivates the users to interact and participate by providing incentives such as praise and encouragement. Even though the requirements engineers influence feedback collection in various ways, no conclusions can be drawn about the impact of their influence, as this has not been evaluated on any platform.

Key takeaway: All platforms involve REengs influencing feedback collection in some way, whether through summarizing, commenting, creating guidelines, moderating or motivating users.

Answer to RQ1: Feedback collected on platforms can be categorized into main feedback and meta-feedback, where meta-feedback references main feedback. Main feedback is submitted either unstructured or structured, while meta-feedback is given through comments, voting or special forms (RQ1.1). Platforms rely on push feedback instead of asking questions to the user, and while most collect feedback in a single phase, some collect feedback through multiple phases for refinement (RQ1.2). REengs influence the collection by summarizing, commenting, providing guidelines, moderating discussions or motivating users (RQ1.3).

—————— RQ2 What is the context of the feedback collection? ——————

———————— RQ2.1 In what environment is the feedback collected? ————————

There are three different environments in which feedback is collected: government, research, and commercial. One platform (KMar-Crowd) describes the feedback collection in the government sector, four platforms (GARUSO, CRUISE, Athena, WikiWinWin) describe feedback collection in the research sector, and four platforms (Tournify, REfine, iThink and CrowdCore) describe feedback collection in the commercial sector. Two platforms do not mention the environment for feedback collection. Feedback was collected for a range of products, including operational systems, smart living applications, tournament management tools, compliance platforms, and specialized databases. Users varied widely, including employees, students, clients, and online participants, though some products and users were not described in detail.

Key takeaway: Feedback is collected across three environments - government, research, and commercial - targeting a variety of products with different user groups.

———————— RQ2.2 How many users is feedback collected from? ————————

To answer this research question, we present the number of users who are invited to the platform, the number of users who access the platform, as well as their proportion to the invited users, as well as the number of users that contribute main feedback or meta-feedback and their proportion to the number of invited users. We mention the platforms descending by the number of contributing users. In the V-Sys study of KMar-Crowd, out of 2392 invited users, 385 users are accessing (16%) and 130 users are contributing (5%). In the S-Sys study of KMar-Crowd, out of 478 invited users, 135 are accessing (28%) and 60 contributing (13%). On Tournify, out of 337 invited users, 157 are accessing users (47%), and 39 users contribute feedback (12%). On GARUSO, out of 726 accessing users, 32 users contribute feedback. The number of invited users is unknown. On Refine, 37 users were invited and 19 users (51%) access and contribute feedback. On CRUISE, all 18 users contribute feedback. In the second study of iThink all 17 users contribute feedback. In the S-Sys study of iThink all 7 users contribute feedback. On Athena, all 6 users contribute feedback. Lastly, on WikiWinWin, all 6 users provide feedback. The other platforms do not mention the number of users.

Key takeaway: The proportion of contributing users among accessing users varies across platforms, with some where all accessing users are also contributing and others where only very few contributing users exist among the accessing users.

———————————— RQ2.3 How long is feedback collected? ————————————

GARUSO collects feedback for the longest with a study duration of 92 days. The V-Sys study of KMar-Crowd collects feedback for 56 days. Tournify and REfine both collect feedback for 35 days. The S-Sys study of KMar-Crowd collects feedback for 33 days.

Lastly, Athena and WikiWinWin only collect feedback for 4 hours. The other platforms do not mention how long feedback is collected.

Key takeaway: Feedback collection durations vary widely across platforms, ranging from 92 days to just 4 hours, with some platforms not reporting their collection periods.

Answer to RQ2: Feedback is collected in three main environments—government, research and commercial (RQ2.1)—targeting a variety of products with different user groups. The number of contributing users relative to accessing users varies strongly, with some where all accessing users are also contributing and others where only very few contributing users exist among the accessing users (RQ2.2). Feedback collection durations also differ widely, ranging from several months to just a few hours, while some platforms do not report the duration at all (RQ2.3).

——————— RQ3 What is evaluated and what are the results? ———————

—— RQ3.1 How is the user participation evaluated and what are the results? ———

Platforms measure user participation by the number of interactions, where an interaction is either the submission of main feedback or the submission of meta-feedback. Additionally, users are divided into invited, accessing and contributing, as described in RQ2.2. The number of votes tends to be higher than the number of comments. In terms of total main feedback, the V-Sys study of KMar-Crowd collected 78 overall, averaging 0.6 per contributing user. WikiWinWin collected 62 in total, Tournify 57 with 1.5 per user, GARUSO 56 with 1.75 per user, the S-Sys study of KMar-Crowd 32 with 0.5 per user, REfine 21 with 1.1 per user, while iThink collected 22 in the second study and 10 in the first. The number of meta-feedback can be found in the complete synthesis matrix (Table A.2.3) in the appendix. Regarding the number of meta-feedback, the tendency is that the number of comments is higher than the number of main feedback.

Key takeaway: Platforms measure participation through interactions, including main and meta-feedback submissions, with votes typically outnumbering comments and comments tending to exceed the amount of main feedback. Main feedback contributions vary widely, both in absolute numbers, ranging from 10 to 78 and average submissions per contributing user, which range from less than 0.5 to nearly 2.

————— RQ3.2 How is the feedback evaluated and what are the results? —————

Three platforms evaluated the main feedback that is collected. The authors of KMar-Crowd classified their main feedback (user stories) into three categories of the KANO model (Berger and Blauth, 1993). The three categories were: “must-be implemented”, “one-dimensional” (detrimental if not implemented, useful when implemented) and “attractive qualities” (i.e. delighters). In the S-Sys study of KMar-Crowd 13 user stories fell into the category “must-be implemented”, 10 were “one-dimensional” and 12 were

“attractive”. In the V-Sys study 50,6% of the user stories were “must-be implemented”, 36,7% was “one-dimensional” and 12,7% was “attractive”. In the S-Sys study KMar-Crowd also classified whether user stories were gathered earlier through other methods than the platform. 19 times they were gathered before completely, 6 times partly and 5 times not at all. It was also classified in the S-Sys study whether the user stories were complete enough for the development teams to implement. 11 times the user stories were complete enough and 19 times they were not. In the V-Sys study the user stories were classified regarding whether they were suitable for an MVP (59,5%) or whether they were enough for a product (27,8%). Furthermore, also in the V-Sys study the granularity of the user stories was analyzed. 40,5% of the user stories represented the granularity of an epic, 54,5% the granularity of an actual user story and for 5,1% of the user stories the classification was not applicable. Tournify evaluated their main feedback (user stories) based on a quality framework for user stories. 52% of the user stories met all quality aspects and 48% of the user stories contained one or more easily preventable error(s). They also analyzed the amount of work needed to implement the user storis. Therefore, nine out of ten user stories can be developed within one workday. One user story could not be estimated, because it was formulated too vaguely. Lastly, iThink evaluated the sentiment of their meta-feedback (comments). There were 6 positive comments in the first study and 48 in the second study. 3 comments of the first study and 32 of the second study were neutral. 6 comments of the first and second study were negative. CRUISE also evaluated whether their collected main feedback is comparable to feedback from interviews. The result was that the feedback is comparable.

Key takeaway: Platforms evaluated main feedback using various criteria, such as classification after KANO, where feedback was categorized into essential, useful, or delightful qualities. Main feedback was assessed for its suitability for development, with most being adequate for MVPs and some for full products. Granularity was also assessed ranging from epics to user stories. Quality evaluations highlighted either no or preventable errors in some user stories. Analysis for implementation effort showed most main feedback could be quickly addressed. Sentiment analysis of meta-feedback revealed more positive or neutral comments than negative comments.

——— RQ3.3 How is the platform evaluated and what are the results? ———

Seven articles evaluated their platform. The user acceptance of the platform was evaluated five times either through a questionnaire or through conversations with experts. KMar-Crowd used a questionnaire to ask the users questions about how they liked the way of working with the platform. In both the S-Sys and the V-Sys study the results were positive. Tournify also used a questionnaire to ask about the perceived usefulness of the platform. 10 users answered the questionnaire and found that the platform is very useful. Regarding REfine, 17 users answered a questionnaire and found the process as difficult, more useful and more engaging compared with previous feedback experiences. The users especially liked voting and commenting. The

results of an interview with experts were that the platform is useful for requirements elicitation, negotiation and specification. iThink also evaluated the platform through a questionnaire and the answers indicated a high level of acceptance for both the first and second study. CrowdCore asked experts about their opinion regarding the platform. The approach was found effective in involving users in the requirements engineering process. Concerns were raised about the applicability of the approach for all software types and the need for users to trust product owners to implement prioritized requirements. One platform (WikiWinWin) also compared their features to other platforms (EasyWinWin and SOP-Wiki). The results were that WikiWinWins main strengths are the exchange of ideas and knowledge, the content editing and versioning. Main weaknesses are its lacking automated consistency checking and problems with conflicts during editing conflicts. Lastly, KMar-Crowd especially evaluated the gamification features in the V-Sys study through a questionnaire. The results were that they did not increase motivation.

Key takeaway: Authors evaluated their platforms by measuring user acceptance, perceived usefulness, by asking experts for feedback, by comparing their platform features to other platforms and by assessing gamification features. Results showed positive user acceptance, with users appreciating features like voting and commenting and finding platforms suitable for requirements engineering. However, some challenges were noted, such as the applicability of the platform to other software and the need to trust product owners to implement requirements. Additionally, gamification features were found to have no impact on user motivation in one study.

——— RQ3.4 How do the findings of the feedback affect the software? ———

The findings of the feedback (see RQ3.2) influence the software by providing actionable insights for development. Classification using models like KANO helps prioritize user stories based on their importance. Assessing feedback suitability for MVPs or full products ensures development aligns with user needs. Quality evaluations improve user stories errors and thus influence further development of software positively.

Answer to RQ3: Platforms evaluate user participation by measuring main and meta-feedback submissions, with participation varying widely between the platforms. Feedback (RQ3.2) is evaluated using criteria like KANO classification, granularity and quality, providing actionable insights for development. Platforms themselves (RQ3.3) are assessed for user acceptance and usefulness with positive results overall and few challenges. The findings of the feedback (RQ3.4) influence software development by improving the prioritization and quality of user stories.

——————————— RQ4 How does the platform work? ———————————

——— RQ4.1 What functionalities to collect feedback does the platform offer? ——

All platforms offer functionalities to collect main feedback and meta-feedback. Regarding meta-feedback all platforms offer the possibility to comment on other users' main feedback and all platforms except Athena offer the ability to do either some form of voting, scoring or rating. As mentioned in RQ1.1 three platforms have special forms of meta-feedback. GARUSO offers functionality to submit meta-feedback in the same form as main feedback, REfine offers functionality to create branches of the main feedback and WPFSRE allows the users to mark relations between main feedback.

————— RQ4.2 What additional functionalities does the platform offer? ————

Platforms offer a variety of additional functionalities beyond feedback collection, including gamification elements like points, badges, leaderboards and challenges to enhance engagement. Other features support organization and collaboration, such as main feedback overviews, user profiles, tagging, filtering, revision history, synchronization with issue trackers and dashboards. Advanced functionalities include importing/exporting feedback, controlled user registration, role and user management, status tracking and effort estimation.

——————————— RQ4.3 Is the platform accessible? ———————————

Only Bazaar is accessible publicly through a website, but it requires an account to use.

Answer to RQ4: Platforms provide various functionalities to support feedback collection (RQ4.1), including the ability to collect main feedback and meta-feedback through unstructured and structured forms, commenting, voting or scoring, with some platforms offering advanced meta-feedback features like branching or marking relations. Additional functionalities (RQ4.2) extend beyond feedback collection, using gamification elements like points and leaderboards and supporting collaboration with features such as user profiles, tagging, filtering, revision history and synchronization with issue trackers. Accessibility (RQ4.3) is limited, with only one platform accessible publicly, but needing to register a user account for access.

3.4 Discussion

In the following the results of the research questions are discussed, based on the authors' interpretations.

[RQ1.1] The platforms collect their main feedback almost equally as freetext or in the form of templates. Collecting main feedback as freetext allows the users to express their thoughts without constraints which could lower the barrier for providing feedback, as users do not need to worry about fitting their feedback into a predetermined template. On the other hand, templated feedback makes analyzing the feedback easier.

[RQ1.2] All platforms use push feedback, meaning that the users do not get any questions and provide the feedback autonomously. The feedback collection methodology of the platforms also differs in the number of phases at which feedback is collected. Using multiple phases could have an impact on the resulting quality of the feedback, but requires more time and effort from the users.

[RQ1.3] The REengs of most platforms influence the feedback collection either by summarizing feedback, commenting and setting guidelines. Summarization and commenting can enhance the motivation of the users to participate and shows the users that their feedback is received and valued. However, influencing the collection of feedback might introduce bias. For example, by summarizing and commenting feedback, the REengs might inadvertently highlight certain aspects while ignoring others.

[RQ2.1] Feedback is collected in government, research and commercial environments. Only one platform (KMar-Crowd) collects feedback in a governmental environment. The remaining platforms equally collect feedback in either a research or commercial environment. Platforms in the research environment (GARUSO, CRUISE, Athena and WikiWinWin) could profit from a potentially less restrictive settings regarding feedback collection methods. Platforms in a commercial environment (Tournify, REfine, iThink and CrowdCore) might be more influenced by business and developer goals than other platforms.

[RQ2.2] The number of accessing and contributing users varies strongly across platforms. The number of contributing users is often times much lower than the number of accessing users. Furthermore, the number of contributing users itself in general is not very high, with only one platform reaching more than 100 contributing users (KMar-Crowd). This indicates that while many users may visit a platform, very few are motivated to contribute to the feedback collection. This could be because of a lack of motivation, time or trust.

[RQ2.3] Feedback collection durations range from a few hours to several months. Longer durations can lead to more comprehensive and considered feedback, as users have enough time to engage with a product and the platform. However, long collection

duration can also lead to user fatigue and a drop in participation over time (e.g. in the S-Sys study of KMar-Crowd).

[RQ3.1] The number of collected main feedback varies between the platforms. However, it is unclear how much main feedback each user contributes. It is possible that most of the main feedback is submitted by a few highly motivated contributing users. This however cannot be validated based on the available data of the platforms.

[RQ3.2] As only three platforms analyzed their feedback, it is unclear whether the feedback collected by the other platforms is helpful. Also, it was often times not clear, whether feedback represented opinions, problems or improvement ideas regarding the product. This information would help to be able to assess how much potential the feedback has to derive requirements from it. Opinions without wishes for change are for example not helpful for deriving requirements in our view.

[RQ3.3] Although seven articles evaluated their platform**,** it is noteworthy that no standardized instruments (e.g. the System Usability Scale) were used. The absence of standardized measures makes it difficult to compare the evaluations across different platforms objectively. Furthermore, the evaluations conducted were not very fine-grained. For example, the articles did not evaluate each functional or non-functional requirement of the platforms in detail.

[RQ3.4] Even though some platforms found that the feedback contains actionable insights regarding the further development of the software, none of the platforms described whether the feedback had an actual effect on the software, meaning whether the proposed changes were actually implemented.

[RQ4.1] All platforms allow users to comment on other users' feedback. This implies that user feedback is always visible to other users. By making feedback visible to all users, platforms can create a sense of community and shared purpose as users see their input as part of a larger effort. However, this transparency can also make some users more uncomfortable sharing honest but critical feedback and users' opinions might be influenced by reading the feedback of others.

[RQ4.2] The integration of gamification features has the potential to increase the amount of user feedback, but it could reduce the quality of the feedback, as the users' focus could shift on merely accumulating reward elements. The additional overview and search functionalities could make it easier for the user to find feedback, but if the functionalities are not designed well, it could lead to decreased usability.

[RQ4.3] As only one platform is accessible through the web, the limited accessibility of all platforms makes it hard to validate the insights regarding the platforms.

[Consequences for researchers] The lack of standardized evaluation methods makes it difficult to compare platforms objectively. Established benchmarks for feedback quality or quantity do not exist, making it challenging to assess whether a platform is truly effective in collecting actionable user feedback. This is why researchers should

focus on establishing these evaluation methods and benchmarks in the future to allow for reliable comparisons between platforms.

Gamification elements, such as points, badges, and leaderboards, are implemented in some of the platforms to increase user motivation. While these features could have the potential to motivate users to contribute, it is still unclear, because only one platform evaluated the effect of gamification with the result that it doesn't have an effect on motivation. To address this uncertainty, future research should systematically evaluate the impact of gamification on user motivation regarding the submission of feedback.

[Consequences for practitioners] Unfortunately, only one platform (Bazaar) is available publicly. We suggest trying out this platform to assess whether it has potential to be adapted to specific feedback collection needs in industry.

Furthermore, practitioners can adopt collecting feedback in multiple phases (idea generation, refinement and decision-making) to assess whether this method yields more helpful feedback for them compared to the single-phase methods that are often used in platforms but also in app stores, where feedback collection consists of solely an input form.

3.5 Threats to validity

This Section discusses potential biases of the systematic mapping study.

[Search strategy bias] Our search term could exclude relevant articles because of our specific choice of search terms. To counteract this threat, we identified multiple alternatives to the search terms through analyzing the frequency and similarity of words in known relevant articles. Furthermore, we included wildcards in our search terms to include also slightly different forms of a word.

[Selection bias] One threat is to not find all relevant articles. One problem could be that we couldn't use the Scopus library, because we had no access to it. Another problem could be that we had to limit the number of search results of SpringerLink, because it didn't allow to scope the search terms individually to title, abstract and full text. Furthermore, during the search term construction, we had to limit the results to the research field of requirements engineering, because otherwise the amount of search results would not have been manageable. This means that there could be more relevant articles that are not in the research field of requirements engineering. To counteract this threat, we used a combination of a term-based search across four high quality search sources and then applied forwards and backwards snowballing again.

[Data interpretation] It might be that we interpreted the data of the articles in a different way than the authors, leading to inaccuracies or incorrect conclusions. To counteract this threat, we carefully cross-checked our interpretations with the data presented in the articles, ensuring consistency and alignment with the authors' findings.

PART III

TREATMENT DESIGN

Chapter 4

Process to collect feedback and derive requirements

This Chapter describes the process to collect feedback and derive requirements. The documentation format is described in Section 4.1. Design decisions that span multiple process steps are described in Section 4.2. Section 4.3 gives an overview over the whole process. Section 4.4 explains the process to collect feedback through initial questions (IQ) and Section 4.5 explains the process to derive requirements through follow-up questions (FUQ). We summarize how our process addresses the problems in Section 4.6. We give a conclusion in Section 4.7.

4.1 Explanation of the documentation format

In this Chapter and the following chapter, we explain the process both in general and specifically for the application within SMART-AGE. We call the application of the process in SMART-AGE the *instance* of the process. When describing the process we document decisions. We differentiate between general decisions and instance decisions. General decisions represent decisions which are independent of the instance and instance decisions are dependent on the instance of the process. We document general decisions in a box that looks like this:

General decision
This text describes a general decision.

We document instance decisions like this:

Instance decision
This text describes an instance decision.

We document the description of the instance like this:

Instance description
This text describes the instance and it can contain instance decisions.

Instance decision
This text describes an instance decision inside the instance description.

4.2 Design decisions

We describe design decisions that affect multiple process steps in this Section. We describe further design decisions regarding individual steps in the following Sections.

General decisions
[Designing a new process] All of the platforms that we found through the systematic mapping study in Chapter 3 use push feedback in their process. Using push feedback makes it difficult to address *P2 "Control of timing of feedback collection"*, because the REengs rely on the users to give feedback autonomously. Furthermore, none of the platforms addresses *P1.2 "Feedback can be mapped to requirements"* by collecting feedback together with information to which requirement it is associated. To address these problems we design a new process that uses questions to collect feedback. We describe how our process addresses the problems in Section 4.4, Section 4.5 and Section 4.6 . **[Use multiple phases for feedback collection]** As we identified in the mapping study in Chapter 3, several platforms use multiple phases to collect feedback. We adopt using multiple phases for our process to collect feedback and to derive requirements, because this makes it possible to converge on specific aspects of feedback. This contributes to conquering *P3 "Support of change requests among users"*, because we can use a separate phase to measure the support among users of wishes for change in functionality (change requests). **[No gamification]** We decided to not use gamification, because the gamification features seemed to not have an effect for KMar-Crowd and because according to the meta-review (Sardi et al., 2017), a noticeable short-term effect on the users' motivation and engagement is unlikely to be sustained, as the users' interest and enthusiasm in the game-like features seems to decrease in the long.

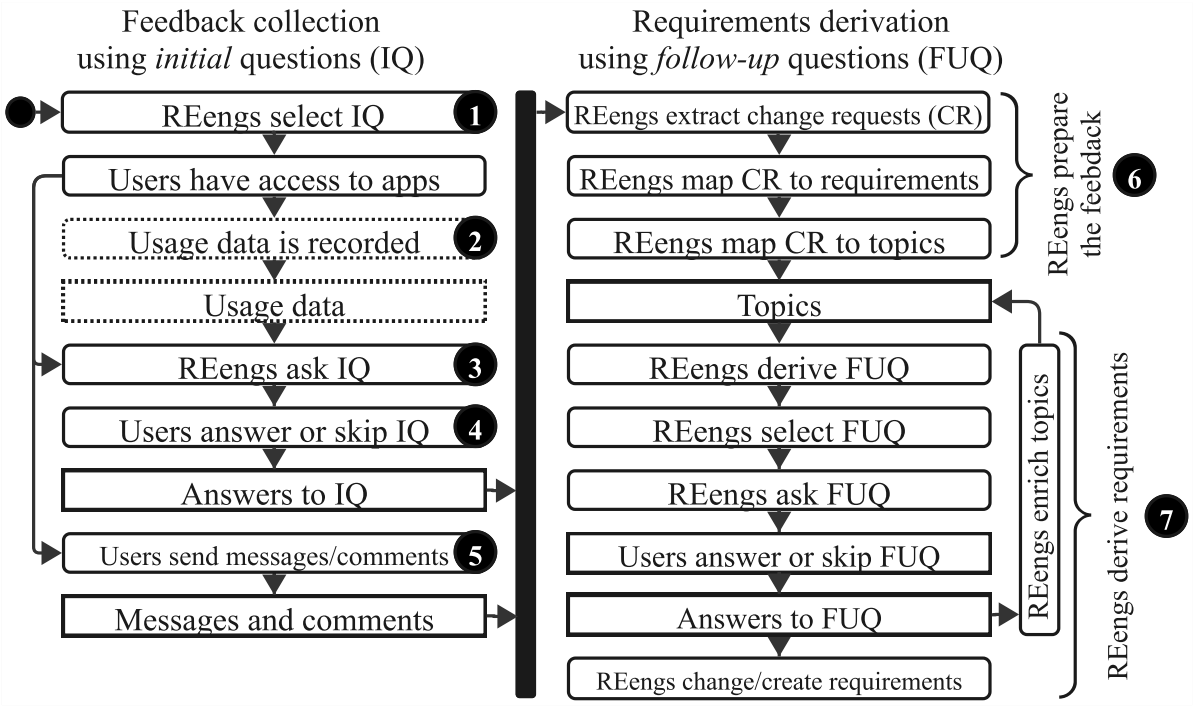
Regarding the application of the process in SMART-AGE, we made the following decision that spans multiple process steps.

Instance decision
[No gamification] (Altmeyer et al., 2018) found that older adults avoid competition and prefer collaboration and caretaking. They consider badges and points as meaningless because they provide a level of visibility that puts older adults under pressure.

4.3 Overview of the process

In the following, we describe our process to collect feedback and derive requirements. The process is supported by the platform SF which is described in Chapter 5. The process is shown in Figure 4.1.

Figure 4.1: Activity diagram representing the process to collect feedback and derive requirements. Numbers (e.g. ❶) indicate individual steps (or groups of steps) of the process.



The process consists of the collection of feedback through initial questions (IQ) and the derivation of requirements through follow-up questions (FUQ). Table 4.1 gives an overview about the process steps and in which Section they are described.

Table 4.1: Process steps description and reasons

Nr.	Step	Section
	Process to collect feedback through IQ	
❶	REengs select IQ	Section 4.4.1
❷	Usage data is recorded (optional)	Section 4.4.2
❸	REengs ask IQ	Section 4.4.2
❹	Users answer or skip IQ	Section 4.4.3
❺	Users send messages and comments	Section 4.4.4
	Process to derive requirements through FUQ	
❻	REengs prepare the feedback	Section 4.5.3
	REengs extract change requests (CR)	Section 4.5.3.1
	REengs map CR to requirements	Section 4.5.3.2
	REengs map CR to topics	Section 4.5.3.3
❼	REengs derive requirements	Section 4.5.4
	REengs derive FUQ	Section 4.5.4.1 Section 4.5.4.3
	REengs select FUQ	Section 4.5.4.2 Section 4.5.4.3
	REengs ask FUQ	Section 4.5.4.3
	Users answer or skip FUQ	Section 4.5.4.3
	REengs enrich topics	Section 4.5.4.3
	REengs change/create requirements	Section 4.5.4.4

In the following we describe the steps briefly including a justification for each step**.** We extend our description in the respective Sections of the process steps.

Process to collect feedback through IQ

REengs select IQ (❶)

The REengs select the IQ that they want to ask the users before the users start the use of the apps.

General decision
A selection of IQ is necessary, because depending on the goals of the REengs, different IQ need to be chosen.

We also propose adaptive IQ, which are personalized questions that are based on the usage data of the user.

General decision
We propose adaptive IQ, because they allow us to better ensure that a question is actually answerable for the user. For example, we can ask why a specific function was not used. This question only makes sense to a user if they really did not use the function.

Usage data is recorded (optional) (❷)

This step is only required, when adaptive IQ are selected in ❶, because adaptive IQ are the only questions that rely on recorded usage data. If no adaptive IQ were chosen this step can be skipped. If adaptive IQ were chosen, all interactions of the users with the apps (e.g. starting/stopping an app, clicking on UI elements) are recorded and saved.

REengs ask IQ (❸)

The REengs ask the selected IQ to the users.

General decision
We ask questions instead of relying solely on push feedback to address the problems *P1.2* and *P2*, as already mentioned above in Section 4.2.

Users answer or skip IQ (❹)

The users can answer and skip the IQ (❹).

General decision
We allow for skipping questions, to allow users the option to not answer a question. In our view this respects the users' autonomy and reduces frustration.

Users send messages and comments (❺)

As an alternative to answering IQ, the user can also send messages and give comments (addressing *P1.1: A lot of feedback can be collected from a lot of users*).

General decision
We allow for submitting messages to give users the possibility to send feedback that is not covered by the IQ. We allow for submitting comments, so that users can add additional information to their answers and messages after submission.

Based on the received feedback, the process to derive requirements through FUQ is conducted.

——————— Process to derive requirements through FUQ ———————

——————————— REengs prepare the feedback (❻) ———————————

The REengs prepare the feedback, which involves extracting change requests (CR) and mapping them to requirements and to topics.

General decision
We extract CR, because they indicate wishes regarding the change of functionality or new functionality and we need them to derive requirements. We map CR to topics because this an established method to process feedback (Li et al., 2024). In our process topics represent aspects of change regarding a requirement. This is why we map CR to requirements as well.

——————————— REengs derive requirements (❼) ———————————

Based on the topics, the REengs iteratively derive, select and ask FUQ.

General decision
We ask FUQ to collect CR that are actionable (ACR), to identify the ACR which are most desired by the users and to validate whether an ACR should be implemented. Collecting ACR is necessary, because some CR are not concrete enough to derive requirements (non-actionable). Identifying the most desired ACR is necessary when there are multiple ACR and it is not clear which ACR should be validated regarding whether it should be implemented. The identification of ACR and the validation of whether an ACR should be implemented address *P3: Support of change requests among users*.

The users answer or skip these FUQ. Based on the answers to the FUQ, the REengs either enrich the topics with the collected ACR, they note in the topics which ACR is

most desired by the users and they change/create requirements based on an ACR that should be implemented. Based on the enriched topics the REengs can conduct a new iteration of deriving, selecting, asking and enriching topics. We explain how this works and why iterations are needed in Section 4.5.4.3.

4.4 Process to collect feedback through initial questions (IQ)

In this Section we describe in detail steps ❶ to ❺ of the process. We describe how REengs select IQ (❶) in Section 4.4.1. We describe how usage data is recorded (❷) and how REengs ask IQ (❸) in Section 4.4.2. We describe how users answer or skip IQ (❹) in Section 4.4.3 and we describe how users send messages and comments (❺) in Section 4.4.4.

4.4.1 REengs select IQ

We offer a variety of IQ with different characteristics that can be used to collect feedback (addressing *P1.1: A lot of feedback can be collected from a lot of users*). We describe the characteristics of these IQ in Section 4.4.1.1 and we describe their structure and give examples in Section 4.4.1.2. We describe how to decide which IQ to select in Section 4.4.1.3.

4.4.1.1 IQ characteristics

Our proposed IQ have the following characteristics: *owner*, *purpose*, *type*, *aspect*, *category* and *app*.

The *owner* of an IQ represents the person that is responsible for that IQ. This could be the REeng, the product owner, the developer or any other person that needs to ask a question to the users.

Instance description
In SMART-AGE we have us REengs and stakeholders of the apps asking IQ to the users.

The *purpose* of an IQ represents the reason for why the IQ is asked. There could be different purposes for asking an IQ to the users. For example, the purpose could be to derive requirements for the app, to evaluate aspects of the app, or to let the users answer a question as an exercise.

General decision
For the conduction of our process to collect feedback and derive requirements, only IQ with the purpose to derive requirements are necessary. Exercise questions

are optional and they can be asked so that the users are familiar with the functionality before answering IQ with the purpose to derive requirements.

Instance description
In SMART-AGE also ask questions to evaluate the usability of the apps.

Instance decision
We ask questions regarding the evaluation of the usability, to use the results in our evaluation.

The *type* of a question can either be scheduled or adaptive. Scheduled means that the question is asked after a fixed number of days relative to the start of using an app (addressing *P2 "Control of timing of feedback collection"*). Adaptive means that the question is asked depending on the usage behavior of the user.

Instance description
In SMART-AGE we ask both scheduled and adaptive IQ. Regarding adaptive IQ, in SMART-AGE we ask why the users didn't use the app or a specific functionality of it for some time and we ask how the functionality can be improved so it is used more often.

Instance decision
We ask adaptive IQ in SMARTAGE, because we think that asking about reasons for inactivity can yield change requests that help us to improve the apps (addressing *P1.3 "Feedback contains change requests"*).

The *category* represents whether the IQ asks about opinions, problems or improvements (OPI) or about reasons and improvements (RI).

General decision
We ask for OPI in isolation to obtain more specific answers. We always ask an opinion question, followed by a problem question, followed by an improvement question. We ask for OPI and RI to address *P1.3 "Feedback contains change requests"*.

The category RI is used for adaptive IQ.

IQ of category 'other' do not follow the OPI structure. In our process, questions of category 'other' are a) asked by the stakeholders (as they are not interested in a

combination of OPI), b) asked for the purpose of evaluation taken from a questionnaire or c) address non-functional aspects that often need to be formulated in a specific way.

The characteristic "aspect" distinguishes whether the IQ refers to the system (that means one of the apps) as a whole or to functional aspects or non-functional aspects.

General decision
We ask questions about specific requirements to reduce effort of mapping feedback to requirements (conquering *P1.2 "Feedback can be mapped to requirements"*).

IQ with aspect 'other' cannot be uniquely mapped to either the system as whole or a specific functional or non-functional requirement. In general IQ with a functional aspect refer to system functions and IQ with a non-functional aspect refer to quality in use and product quality (ISO/IEC 25012:2008).

General decision
We want to have feedback about the following aspects of quality in use: Satisfaction, Effectiveness and Efficiency, because we think these aspects help most in deriving requirements. The aspects regarding satisfaction address how useful the users find the app (Usefulness), how much trust they have in it (Trust), how pleasurable they find it (Pleasure) and how much comfort they have with it (Comfort). Regarding effectiveness and efficiency, we ask about the goals of the users, as well as related problems and improvement proposals. Regarding product quality, we address the following aspects, also because we think these aspects help most in deriving requirements: Compatibility, Usability and Security. Regarding compatibility, we ask about the connection to other apps. Regarding usability, we ask about how learnable the app is (Learnability), how easy to operate it (Operability) is, whether user errors occurred and how these can be prevented (User Error Protection), as well as how accessible the app is for older adults (Accessibility). Regarding security, we ask whether the users have doubts regarding data protection.

The characteristic *app* represents the app which the IQ addresses (SF, SV or SI).

4.4.1.2 IQ structure and examples

We explain in this Section how our proposed IQ are structured and give examples from the instance of our process in SMART-AGE. Table 4.2 gives an overview over the structure of IQ, along with their characteristics and their answer options.

Table 4.2: Structure of IQ and their answer options. O=Owner, P=Purpose, T=Type, A=Aspect, C=Category

ID	IQ and its characteristics	Answer options
1)	*How do you like <function> in <app>? Why?* Characteristics: O: REeng, P: Requirements Derivation, T: Scheduled, A: Functional, C: Opinion, App: SF	Likert scale selection, Freetext
2)	*Are there any problems with <function> in <app>? If yes, which ones?* Characteristics: O: REeng, P: Requirements Derivation, T: Scheduled, A: Functional, C: Problem, App: SF	Yes/No selection Freetext
3)	*Can the <function> in <app> be improved? If yes, how?* Characteristics: O: REeng, P: Requirements Derivation, T: Scheduled, A: Functional, C: Improvement, App: SF	Yes/No selection Freetext
4)	This type of question can be formulated very freely. See our instance examples below for inspiration. Characteristics: O: REeng, P: Requirements Derivation, T: Scheduled, A: Non-Functional, C: Other, App: SV	As desired
5)	*Can <app> be improved to improve <NFR>? If yes, how?* Characteristics: O: REeng, P: Requirements Derivation, T: Scheduled, A: Non-Functional, C: Improvement, App: SF	Yes/No selection Freetext
6)	*What is the reason that you have not used <function> in <time range>? How could <app> be improved so that you use it more often?* Characteristics: O: REeng, P: Requirements Derivation, T: Adaptive, A: Functional, C: RI, App: SF	Freetext Freetext

General decision
We often combine two sub-questions in one IQ, such as "*How do you like <function> in <app>? Why?*". Here the first part of the IQ can be answered by selecting a value from a likert scale and the second part can be answered by freetext. This is because only the selection of a value without reasoning does not allow us to derive requirements. We ask for a likert scale quantitatively, because the distribution of the selection of the likert scale allows for a quick insight into the users opinions, without the need to qualitatively analyze the answers. We ask qualitatively for the reason, because it

is important to understand the selection of the likert scale, because otherwise no change to the application can be derived.

For IQ with category OPI we always use a combination of selection and freetext. In category other, we mix sub-questions with the answer option combinations: freetext – selection, selection – only, freetext – freetext and freetext only. This is because some questions address aspects that need more freedom in the question structure.

Instance description

We give concrete examples for IQ that we use in our process in SMART-AGE in Table 4.3.

Table 4.3: Example IQ and answer options.

Id	Example IQ
1)	How do you like the history function in SF? Why?
2)	Are there any problems with displaying the history in SF? If yes, which ones?
3)	Can the display of the history in SF be improved? If yes, how?
4)	Are you concerned about the security of your data in SV? Why?
5)	Can SV be improved to make it particularly good for users over 67? If yes, how?
6)	What is the reason that you have not looked at a question in SF in the last week? How could the app be improved so that you use it more often?

We provide the complete list of IQ in our repository[10].

4.4.1.3 IQ selection

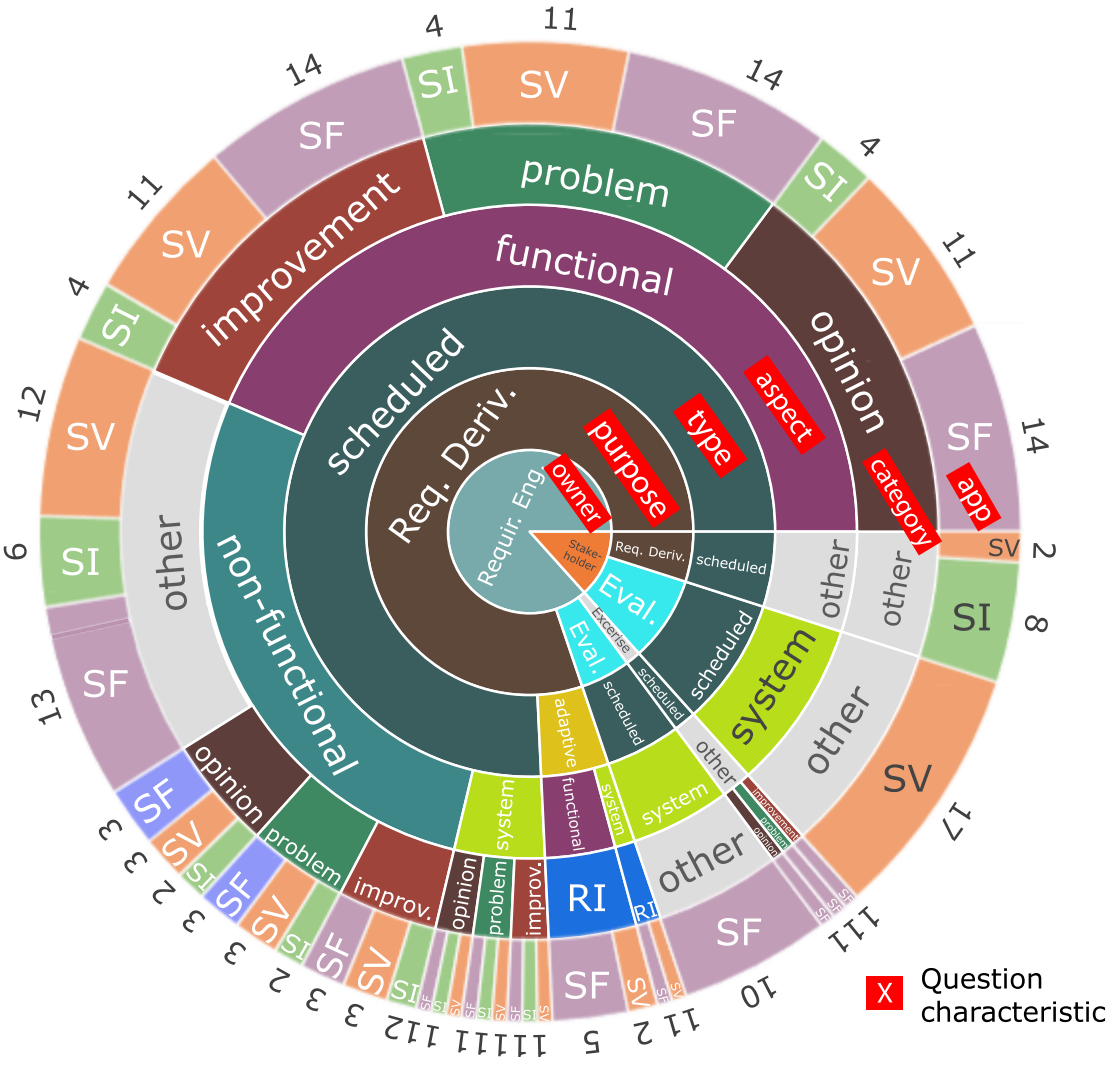
Instance description

Figure 4.2 shows the distribution of IQ that we selected in SMART-AGE. Each characteristic is mapped to a ring starting with *owner* as the innermost ring and *app* with the outermost ring. We provide a repository which includes the detailed plan when which questions are asked[11].

[10] https://github.com/lradeck/dissertation/blob/main/IQ.xlsx

[11] https://github.com/lradeck/dissertation/blob/main/IQ.xlsx

Figure 4.2: Sunburst chart with distribution of IQ and their characteristics



To conduct our process, other REengs don't need to select IQ the same way we do in SMART-AGE. They can select a subset of IQ that fits their individual needs and constraints. For example, when no usage data can be recorded because of data privacy, adaptive IQ can be omitted. When the REengs want to focus on collecting feedback regarding functional requirements, they can just ask IQ with aspect functional. Additionally, we recommend to read Section 10.3.2, which gives insights regarding how effective different IQ were in collecting feedback. These insights also can help when deciding which IQ to select.

4.4.2 REengs ask IQ

We ask no more than five IQ per day and we ask a mixture of different IQ with a specific order.

General decision
To enhance the quality of experience for the users during feedback collection, we do not ask more than five questions per day (Fotrousi et al., 2018) and we mix questions to make answering questions more interesting. When asking for OPI, we ask for the opinion first, then for problems and then improvements. We ask for opinions first, because they are more abstract than answers regarding problems and improvements. We ask questions at specific timepoints to address *P2 "Control of timing of feedback collection"*.

Instance description
In SMART-AGE out of the IQ that we ask each day, three are specifically designed to gather feedback on OPI, while the other two IQ are selected randomly to provide a variation.

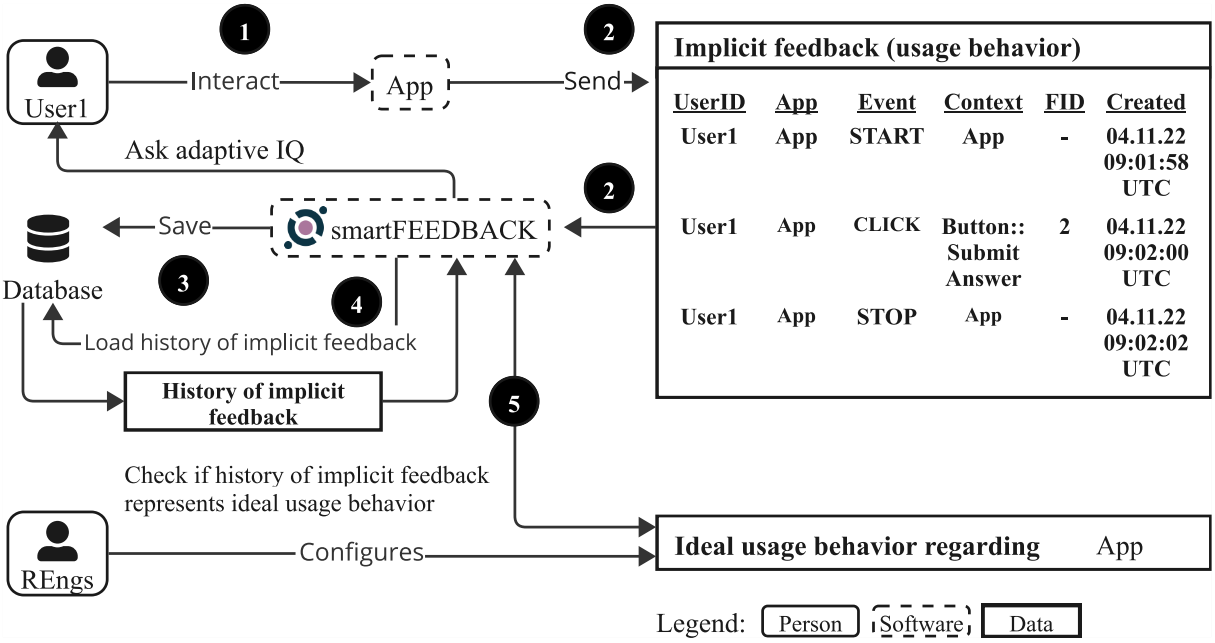
We strategically schedule the IQ regarding functionality and user experience.

General decision
We ask IQ about less prominent system functions later in the process, allowing users enough time to explore and familiarize themselves with all aspects of the app.

IQ that are asked to the users do not expire. If the users don't answer the IQ they receive on one day, they can answer these IQ on the next day together with the new IQ of that day.

The process of asking adaptive IQ is illustrated in Figure 4.3. The user with user id "User1" starts interacting with the app (❶). The resulting implicit feedback is sent to SF (❷). The implicit feedback consists of the ID of the user (*UserID*), the app that was used (*App*), the event that happened (*Event* – e.g. CLICK for clicking on a user interface element or START for starting the app), the context of the event (*Context* – e.g. which user interface element was clicked on), a foreign ID referencing an entity of the app that was used (*FID* – e.g. the ID of a link clicked) and the date at which the event was created (*Created*). The *userID* is necessary to trace back the implicit feedback to the user. The *App* is important, as otherwise implicit feedback of multiple apps couldn't be distinguished.

Figure 4.3: Diagram representing the process of asking an adaptive IQ.



The *context* and *FID* are necessary, because they define an interaction with an app. *Created* is important, because otherwise no calculations regarding a behavior over time can be conducted. SF receives the implicit feedback and saves it to the database (❸). SF now periodically loads the history of the implicit feedback (❹) and checks whether it does not represent the ideal usage behaviour of the app about which feedback is collected (❺). The ideal usage behaviour is configured by the REengs but defined by the stakeholders of the apps. To define ideal usage behavior the stakeholders have to think about how they would like the users to interact with the apps. They then formulate their wishes in form of measurable metrics and communicate them to the REengs. The REengs then configure the metrics in SF.

Instance description
For example, in SMART-AGE all stakeholders want the users to open the apps it at least once a week. This frequency was discussed among the stakeholders and was found to reflect regular engagement with the apps. The SI stakeholders defined ideal usage behavior so that the users should check every recommendation within one day, that they should answer every question within one day and that they should open the link of a given recommendation within three days. The users should also never skip questions. Regarding SV, the users should open the news and a category at least once per week. Regarding SF, once per week the users should open the history, look at least one of their answers, open a question, send a message and use the audio recording function

If the implicit feedback does not represent ideal usage behavior the user receives an adaptive question, which asks for the reason and for improvement ideas.

General decision
The asking of adaptive question is similar to (Wüest et al., 2019) and (Fotrousi and Fricker, 2016). (Wüest et al., 2019) trigger feedback collection based on user goals in the context of a navigation system. (Fotrousi and Fricker, 2016) also collects explicit pull feedback based on implicit feedback. Our collection of implicit feedback adapts aspects from these articles: (Dzvonyar et al., 2016), (Oriol et al., 2018a), (Stade et al., 2017) and (Fotrousi et al., 2018). From (Dzvonyar et al., 2016) we adapt recording the user id to know which user sent the implicit feedback. We adapt recording which application sent which implicit feedback from (Oriol et al., 2018a), to be able to differentiate implicit feedback between the applications. We adapted recording events based on interaction level and their timestamps from (Stade et al., 2017; Fotrousi et al., 2018; Oriol et al., 2018b), because this is necessary to ask adaptive questions. QoE logs the user id, timestamps of events on feature level (e.g. starting or completing a feature) and user interaction level (e.g. user input or an application output) and then triggers a feedback collection form with the option to answer a question about the users satisfaction and the reasons for the user behaviour. Compared to (Wüest et al., 2019) and (Fotrousi and Fricker, 2016) we do not collect pull feedback based on specific activities that are detected in the implicit feedback, but we collect pull feedback through adaptive questions based on the absence of activity to ask for the reasons.

4.4.3 Users answer or skip IQ

The users can answer and skip the IQ. After answering or skipping an IQ either the next IQ is shown (if there is another IQ open to answer) or no IQ is shown anymore.

4.4.4 Users send messages and comments

We allow users to send messages and comments as an alternative to answering IQ.

General decision
We implement commenting on answers and messages like almost all platforms.

Instance description
Due to constraints of our study, users cannot see and comment other feedback of other users. This is because users could influence each other through their feedback, leading to non-independent responses. This would violate the assumption of independent observations, which is critical for many statistical analyses. However, the users can comment their own feedback to extend their answer or message with aspects they did not include initially.

4.5 Process to derive requirements through follow-up questions (FUQ)

In this Section we describe how we derive requirements through follow-up questions (FUQ) based on the feedback to the IQ. Section 4.5.1 presents related work. Section 4.5.2 explains relevant terminology. The following Sections describe the steps of the process to derive requirements through FUQ of Figure 4.1 (❻ and ❼). Section 4.5.3 describes how REengs prepare the feedback (❻) and Section 4.5.4 describes how the REengs derive requirements (❼).

4.5.1 Related work

We conducted a term-based literature search and snowballing in (Scherbatschenko, 2023) to find articles that describe how requirements can be derived from feedback systematically. The search was executed in 2023 and did not yield any results. However, we eventually found two industry studies (Johanssen et al., 2019) and (Li et al., 2024). In (Johanssen et al., 2019), the authors asked practitioners how they capture and utilize user feedback. Feedback is collected explicitly (e.g., surveys) and implicitly (e.g., usage data), analyzed to link it to features or applications, validated to ensure alignment with user needs and prioritized to guide feature development and improvement. These steps correspond well to the insights of (Li et al., 2024). (Li et al., 2024) state that the derivation of requirements from feedback is still an open problem. Even though they do not present a systematic process to derive requirements from feedback, they identify key steps of a life cycle of managing user feedback in organizations to improve their products. This life cycle comprises four essential steps: 1) collection, 2) analysis, 3) validation and 4) prioritization of user feedback. In the collection phase, organizations gather feedback from different sources, including emails, support tickets, online platforms and user usage data. Next, in the analysis phase, feedback is examined to identify common themes. A theme is a set of feedback that addresses the same problem or feature. (Panichella et al., 2015; Guzman et al., 2016) for example, analyze feedback by employing systematic content analysis and machine learning to classify feedback into categories (e.g. praise, bug, complaint, etc.), to understand a vast volume of feedback related to software applications. The validation phase ensures that these themes are accurate and really represent the users' needs. For example, (Lohmann et al., 2009; Laporti et al., 2009; Fernandes et al., 2012; Snijders et al., 2015; Sharma and Sureka, 2018; Menkveld et al., 2019; Kolpondinos and Glinz, 2020; Wouters et al., 2022) validate user feedback through some form of voting, scoring or rating, to assess the support among users. Finally, in the prioritization phase, validated feedback is ranked based on its alignment with organizational goals or potential impact on users. For example, the articles (Gartner and Schneider, 2012; Kifetew et al., 2021; Malgaonkar et al., 2022) describe the automation of how feedback can be prioritised, but according to the industry practitioners in (Li et al., 2024), these tools are rarely used.

4.5.2 Terminology

In this Section we explain the terminology that is relevant for understanding the further Sections.

[Deriving requirements] With *derivation of requirements* we mean changing existing requirements and creating new requirements.

[Follow-up questions (FUQ)] In our process to derive requirements from feedback we use questions. We call these questions *follow-up questions (FUQ),* because they are asked to the users after the user already gave feedback to our IQ We use different types of FUQ which we distinguish with FUQ1, FUQ2 and FUQ3 as explained below.

[Change requests] From all the feedback of our users, we use feedback that represents *change requests (CR)* regarding an app.

General decision
We use only CR, because these indicate wishes regarding the change of functionality or new functionality, and thus change wrt. requirements.

There exist two types of CR, *non-actionable change requests* (*NACR*) and *actionable change requests* (*ACR*).

[Non-actionable change requests (NACR)]. *NACR* are CR that do not contain detailed enough information to allow the direct derivation of a change to a requirement. An example for an NACR would be "I want to find a website link faster" or "In my view, a website link cannot be found fast". These CR are NACR, because "finding a website link faster" can be achieved through a variety of different solutions. A search function could be the solution to find website links faster or there could be a function that allows the user to adjust the UI to his/her individual needs. For the REeng it is not apparent what exactly should be changed.

[Actionable change requests (ACR)] *ACR* in contrast to NACR allow the direct derivation of requirements. An example for an ACR is "I want to have a bigger font size" or "In my view the font size is very small". These CR are ACR, because an existing requirement (e.g. a system function that is responsible for displaying text) can be changed to reflect the wish for bigger font size.

[Topics] Topics represent an aspect of change regarding a requirement and both NACR and ACR are mapped to them.

4.5.3 REengs prepare the feedback

In this Section we describe the preparation of the feedback that we received through our IQ. The result of the preparation are CR that are mapped to requirements and topics. The steps for the preparation of the feedback are: ❶) Extracting CR from the feedback ❷) Mapping the CR to requirements and ❸) Mapping the CR to topics.

4.5.3.1 REengs extract change requests (CR)

To extract CR we first check whether the feedback is comprehensible. Feedback is comprehensible when the REeng can understand the feedback in terms of grammar and spelling. When feedback is comprehensible, we extract the CR (e.g. in contrast to bug reports). When feedback contains multiple CR or when it contains a CR and also information that is not a CR, we split the feedback into parts, so that each part represents either a CR or not. We call these parts "statements". When the feedback is not split, we call the whole feedback a statement. Examples and more details regarding the extraction of CR can be found in the appendix B.1.

4.5.3.2 REengs map CR to requirements

After extracting CR, we map the CR to requirements.

Instance description
Our requirements are specified using TORE (Paech and Kohler, 2004) which distinguishes user tasks, subtasks, system functions and workspaces. The latter bundles data and functions presented together to the user. We map a CR always to the most technical requirement possible. System functions and workspaces give more technical details than subtasks and subtasks more details than user tasks. We map a CR to a system function when it addresses mainly functional aspects. We map a CR to a workspace when it addresses mainly aspects about the user interface. We map a CR to a subtask if it mainly addresses aspects about the subtask and we map it to a user task, if it addresses aspects about the app or app context which cannot be associated with an existing subtask. Further details regarding the mapping of CR to requirements can be found in the appendix B.1. The requirements for the apps are listed in in the appendix in Table B.1.5, Table B.1.6, Table B.1.7 and Table B.1.8.

Using TORE is not necessary for the process to work. If you are using user stories, just try to identify which user story fits best to the CR. Keep in mind that later based on the CR changes to the associated requirement will be derived.

4.5.3.3 REengs map CR to topics

We map CR to topics, which address aspects of change regarding a requirement. We give examples in the following which should make it clear in general how mapping CR to topics work.

Instance description

Table 4.4 gives examples for topics which address different aspects regarding the system function "SF: Display Question", which is responsible for displaying the questions in SF. C1, C5 and C6 have their own topic, because they address different aspects related to the system function "SF: Display Question". C2, C3 and C4 have the same topic, because they all represent CR regarding the understandability of the questions. C2 wishes for better understandability of questions in general and C3 and C4 provide concrete proposals for enhancing the understandability.

Table 4.4: CR and their topics. C=Id of CR, T=Id of topic,

C	Change request	Reason for ACR/NACR	Topic	T
C1	NACR: "When I answer a question, I want to have more answer options to select."	It is unclear which answer options the user wants.	Answer options	T1
C2	NACR: "I want to have questions that are easier to understand."	It is unclear how we can make the questions easier to understand	Understandability	T2
C3	ACR: "I want you to explain the terms in a question better."	We can explain the terms of a question in more detail		
C4	ACR: "I want you to explain better what aspect of the app a question addresses."	We can explain in more detail what aspect of the app is addressed by the question		
C5	ACR: "I want an overview over all open questions."	We can provide an overview over all open questions	Overview	T3
C6	ACR: "I don't want to be asked about my inactivity."	We can ask no more questions regarding inactivity	Inactivity	T4

4.5.4 REengs derive requirements

In this Section we describe how the REEngs can derive requirements. The process of deriving requirements consists of the steps of deriving FUQ, selecting FUQ, asking FUQ, enriching topics and changing/creating requirements.

4.5.4.1 REengs derive FUQ

Based on the topics we can derive FUQ. Depending on which types of CR are mapped to a topic, we derive different types of FUQ.

We use three types of FUQ: FUQ1, FUQ2 and FUQ3.

General decisions
We use these three types of FUQ, because they allow us to achieve the goals in Table 4.5.

Table 4.5 gives an overview over the different FUQ, their conditions, goals and from which topics they are derived.

Table 4.5: Different types of FUQ along with their conditions, goals and the topic id (T) of Table 4.4 for which they are used

Type	Condition	Goal	Style	Enables	T
FUQ1	Topic contains only NACR	Collect ACR	Divergent	FUQ2/3	T1
FUQ2	Topic contains multiple ACRs	Identify most voted ACR	Convergent	FUQ3	T2
FUQ3	Topic has only one ACR or the most supported ACR is already identified	Validate whether ACR should be implemented.	Convergent	Deriving requirements	T3, T4

If a topic contains only NACR, then we derive a FUQ1 which has the goal to collect an ACR for that topic. This question is a divergent question (Glinz et al., 2020), as it encourages brainstorming and exploration of solution. If a topic contains multiple ACRs, we derive and ask a FUQ2 and let the users decide which ACR is most supported. This question is a convergent question (Glinz et al., 2020), as it chooses one specific solution out of many. If a topic has only one ACR or when the topic has multiple ACR, but the most supported ACR is already identified, then we derive a FUQ3. This question is also convergent, because it validates whether the specific ACR should be accepted for implementation or not. Table 4.5 shows in column "Enables" that the answers of asking FUQ1 enable asking FUQ2 or FUQ3, that the answers of a FUQ2 enable asking FUQ3 and that based on the answers to FUQ3 we can derive requirements.

Instance decision
It would have been possible to ask multiple FUQ3 for a topic that has multiple ACRs, but we decided to let the users vote for the most desired ACR of a topic to reduce the amount of FUQ needed for requirements derivation.

If the number of questions is not relevant, one can ask a FUQ3 for every ACR in a topic.

Table 4.6 shows the elements of FUQ1 and examples.

Table 4.6: FUQ1 elements and examples regarding topic id T1 from Table 4.4

ID	Element	Example
E1.1	Description of the topic as text	"We have received the request to provide more options for answering questions. However, we are not exactly sure how to implement this request."
E1.2	Optional description and screenshot of the related functionality in the app to give context	Description: "In the image below, you can see the current view of a question and its answer options." Image: [for an example see Figure B.2.42 in the appendix]
E1.3	Question whether change regarding topic is desired in general and the answer options:	Instance example: "Do you even want there to be more options for answering questions?"
	Answer options:	Selection: 1) Yes 2) No 3) I don't care 4) I find the question not comprehensible 5) I cannot answer this question
E1.4	Question asking for ACRs	"In case change regarding topic is desired:" Instance example: "What answer options do you want there to be?"
	Answer option:	Freetext

General decisions
E1.1 explains the topic briefly so users can understand why they receive a question. E1.2 provides context, because it could be that users don't understand which functionality in the app is meant. E1.3 lets users state if they desire change in general, which is a precondition for answering E1.4. E1.3 also allows the users to indicate that they find the question not comprehensible or that they cannot answer the question. These answer options are important to improve construct

validity of the FUQ. E1.4 is important to give users a possibility to specify how the change should look like.

Table 4.7 shows the elements of FUQ2 and examples.

Table 4.7: FUQ2 elements and examples regarding topic id T2 of Table 4.4

ID	Element	Example
E2.1	Description of the topic as text	Instance example: "We have received the request to make our questions easier to understand." "However, we are not exactly sure how to implement this request."
E2.2	Optional screenshot of the related functionality in the app to give context	A screenshot could be given here to remind the users about the view of a question, but it is not obligatory. Description: "In the image below, you can see the current view of a question." Image: [for an example see Figure B.2.43]
E2.3	Description of the ACRs as text	"There are the following proposed changes: Instance examples: • More detailed explanation of the terms in a question • More detailed explanation of what aspect of the app the question addresses"
E2.4	Question whether change regarding topic is desired in general	Instance example: "Do you even want our questions to be easier to understand?"
	Answer options:	Selection: 1) Yes 2) No 3) I don't care 4) I find the question or the ACRs not comprehensible 5) I cannot answer this question
E2.5	Question asking which of the ACR is wanted.	"If yes, please decide for a change: Instance examples: • More detailed explanation of the terms in a question • More detailed explanation of what aspect of the app the question addresses"
	Answer options:	All ACRs of the topic (the users can choose one ACR) along with the answer option "No ACR is suitable"

General decisions
E2.3 is important as it shows the users all the possible changes from which they can choose one in E2.5. We let the users select only one option in E2.5, so that they have to think about which change the desire most. The other elements are important because of the same reasons as we explained before.

Table 4.8 shows the elements of FUQ3 and examples.

Table 4.8: FUQ3 elements and examples regarding topic id T3 and T4 of Table 4.4

ID	Element	Example
E3.1	Description of the ACR as text	*[If mockup]:* [T3]: T3 needs solution proposed through mockup, because it is important to validate the UI of the mockup. We write Instance example: "We have received the request to provide an overview over all open questions. "In the image below, you can see our mockup outlined in red." [Mockup] *[if no mockup]:* [T4]: For T4, the solution is clear (not ask any questions regarding inactivity anymore). We write Instance example: "We have received the request to not ask questions about your inactivity anymore."
E3.2	[If mockup]: Mockup is shown here	Mockup: [for an example see Figure B.2.54]
E3.3	[if no mockup]: Optional screenshot of related functionality is shown here	Screenshot: [for an example see Figure B.2.53]
E3.4	Question whether ACR should be accepted	Instance examples: [T3]: "Do you want an overview of the questions?" [T4]: "Do you want us to not ask about your inactivity anymore?"
	Answer options:	1) Yes 2) No 3) I don't care 4) I find the question or the ACR not comprehensible 5) I cannot answer this question

E3.5	[if mockup]: Question whether ACR should be accepted	Instance example: [T3]: "If yes: Do you want us to implement our solution proposal?"
	Answer options:	1) Yes 2) No 3) I don't care
E3.6	[if solution is proposed]: Freetext field where users can enter notes.	"If you have notes: Which notes do you have?"

General decision
E3.1 is important as it provides a description of the ACR to ensure stakeholders understand the proposed change. In E3.2 we provide a mockup, which is necessary if the proposed solution involves UI changes. We use mockups to propose solutions, because explaining user interface changes through text is difficult to understand. In E3.5 we offer the possibility to choose "Yes, but I have notes" and to give notes as freetext in E3.6, because users could have remarks regarding our proposals. The other elements are important because of the same reasons as we explained before.

4.5.4.2 REengs select FUQ

Based on the derived FUQ the REengs select specific FUQ that they want to ask to the users.

General decision
Selecting specific FUQ is necessary if the amount of derived FUQ is too high to be asked to the users. Furthermore, depending on the feedback and the goals of the REengs it makes sense to prefer asking specific FUQ. If the feedback contains a lot of NACRs but few ACRs and the goal is to derive as many requirements as possible, than it makes sense to ask a lot of FUQ1 initially to collect more ACRs. If the goal of the REengs is to change or create requirements very quickly, but it is not necessarily important that a lot of requirements are changed or created, then only FUQ2 and FUQ3 could be asked. It is even possible to skip the voting of ACRs regarding FUQ2 and ask FUQ3 only in case the REengs are sure that the ACR is desired by the community or when there are no alternatives to the ACR. Furthermore, the selection of specific FUQ can be conducted based on characteristics of the FUQ. We explain the characteristics and the reasons why selecting FUQ based on the characteristics is helpful for the REengs below.

The characteristics of the FUQ are listed and explained in Table 4.9 along with examples.

Table 4.9: Characteristics of FUQ and examples

Innovativity (all FUQ have this characteristic)

Definition: A FUQ is considered innovative when the proposed change(s) enable users to do more or less, or to receive more or less amounts of information

Examples for innovative FUQs:

Do more: The FUQ asks whether the users want to change the order of categories or links in an app

Do less: The FUQ asks whether the filter function in an app should be removed

Get more information: The FUQ asks whether the users want to see characteristic tags for the news in a news app

Get less information: The FUQ asks whether the users want to have access to less links in an app

Examples for not innovative FUQs:

The FUQ asks whether the users want to have a larger font regarding the news in a news app

General decision
Ask innovative FUQ to identify changes to requirements or new requirements that influence what users can do or what information they see. Ask not innovative FUQ to identify changes to requirements or new requirements that influence only how users see the information that is displayed by the system.

Percentage of associated users (all FUQ have this characteristic)

Definition: The percentage of users that submitted an ACR or NACR that is associated with the topic of the FUQ divided by all users that submitted an ACR or NACR.

Examples: Assuming the ACRs and the NACR of topic with T2 of Table 4.3 were extracted from the feedback of three different users, the number of associated users would be 3. If there would be 100 users that submitted ACRs or NACRs, the percentage would be 3%.

General decision
Ask FUQ that have a high percentage of associated users to identify changes to requirements or new requirements that are already recognized as important by relatively many users. This can increase the chance of the FUQ getting answered by the users. Ask FUQ that have a low percentage of associated users to identify changes to requirements or new requirements that are recognized as important by relatively few users yet. This can be still be helpful when the proposed change(s) of the FUQ are seen as very important by the REengs.

Cognitive effort (FUQ1 and FUQ2 have this characteristic)

Definition: The required level of creativity (**low, medium, high**) needed to answer a FUQ1 or FUQ2.

Low cognitive effort: In the case of a FUQ1, it is easy for the user to come up with a desired ACR, or in the case of a FUQ2, the range of possible ACRs is highly covered by the provided ACRs.

Example: A FUQ2 asks whether the users don't want to get asked about feature A and/or feature B anymore. The FUQ2 offers the ACRs "Don't ask about feature A anymore", "Don't ask about feature B anymore" and "Don't ask about feature A and feature B anymore". There are no other ACRs imaginable for the user and the user can just choose one of the offered ACRs to answer the question.

Medium cognitive effort: In the case of a FUQ1, it is moderately difficult for the user to come up with a desired ACR, or in the case of a FUQ2, the range of possible ACRs is moderately covered by the provided ACRs.

Example: A FUQ1 asks whether an app should update their links more often and in the case this is desired, it asks how often the links should be updated. The range of possible ACRs is moderate (e.g. once a week/month/year etc.). For the user it is moderately difficult to come up with an ACR. A FUQ2 asks how the design of links could be enhanced. The ACRs offered are "Don't shorten the length of links" and "Make links colored". There are a few other ACRs imaginable for the user (e.g. "Give links clearer names" or "Differentiate links to apps and links to websites in the design", but overall the range of possible ACRs is moderately covered by the offered ACRs.

High: In the case of a FUQ1, it is difficult for the user to come up with a desired ACR, or in the case of a FUQ2, the range of possible ACRs is poorly covered by the provided ACRs.

Example: A FUQ1 asks whether the users want to find content in an app more quickly and if yes how this can be achieved. The range of possible ACRs is high (finding content more quickly can be achieved through a lot of ways such as a search function, tagging, layout changes, etc.). For the user it is difficult to come up with a desired

ACR. A FUQ2 with high cognitive effort would for example ask whether the design of an app should be changed and offer ACRs like “Make symbols bigger” and “Make font bigger”. However, there are very many possibilities how the design of an app and its features could be changed, so the range of possible ACRs is only poorly covered by the two offered ACRs.

General decision
Asking FUQ with higher cognitive effort can yield more creative change requests, but it could also decrease the chance of the FUQ getting answered.

Visualization type (FUQ3 has this characteristic)

Whether or not the FUQ3 requires a mockup to describe the proposal

Mockup: A mockup is required to describe the solution proposal for an ACR

Example: see Table 4.8

No mockup: There is no need for a solution proposal and it is just asked whether the ACR should be implemented or not

Example: see Table 4.8

General decision
Asking FUQ3 with a mockup takes more effort for the REengs, as they have to create the mockup first. However, asking FUQ3 with mockups is needed when changes to requirements or new requirements need to be identified which affect the user interface of the system.

Instance decision
In SMART-AGE we try to ask FUQ which cover all characteristics. We describe our decision in the appendix in B.2.

4.5.4.3 REengs iteratively derive, select and ask FUQ

We derive, select and ask FUQ iteratively, because we can only derive requirements from answers to FUQ3. To ask a FUQ3 it is necessary that either the topic only has one ACR or the most supported ACR is already identified. The identification of the most supported ACR requires to ask a FUQ2. A FUQ2 can only be asked when the topic contains multiple ACR. For topics with only NACR it is thus necessary to ask FUQ1 to identify ACR. We explain the iterative deriving, selecting and asking of FUQ in Figure 4.4. In the following we explain terminology that is necessary to understand before describing Figure 4.4.

[Votes] We define votes as answers that represent an opinion regarding a change ("Yes", "I don't care", "No" and a selection of proposed ACR). A FUQ can receive a maximum of one vote. If for example a user answers the FUQ3 regarding topic T3 with "Yes" for E3.4 and with "Yes" for E3.5, then this is counted as one vote and not two. There is no vote, when the answer doesn't represent an opinion (answering "I find the question (or the ACR) not comprehensible" or "I cannot answer this question") or when the question is skipped.

General decision
Like most other platforms which we identified in Chapter 3, we implement voting. We use voting during requirements derivation to measure whether change requests should be implemented or not. We don't make the results of the voting public during the voting phase, so that the users are not influenced by the intermediate results. Voting contributes to conquering *P3 " Support of change requests among users"*, because we can measure the support of the change requests among the users.

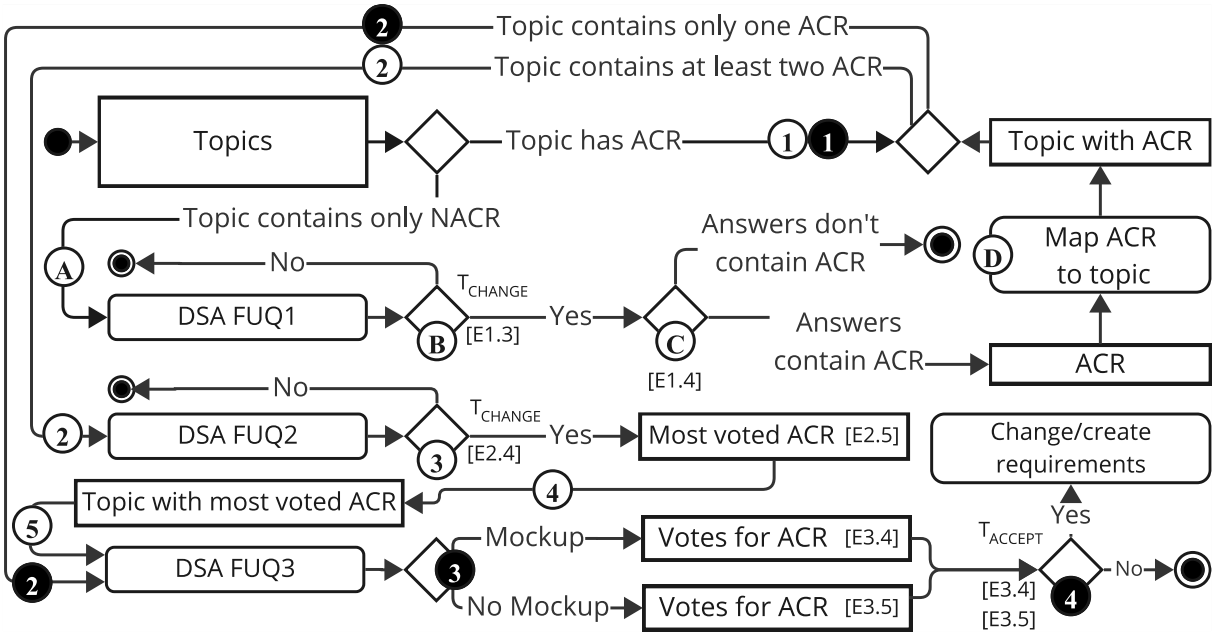
[Thresholds] Change desired in general (T_{CHANGE}). As outlined in Table 4.5, based on the responses to a FUQ1, we can proceed to ask either a FUQ2 or FUQ3. Similarly, based on the responses to a FUQ2, we can ask a FUQ3. We only proceed asking further FUQ when users desire change in general, because it makes no sense to refine or validate a change that is not desired by the users.

General decision
We define T_{CHANGE} for FUQ1 when at least 70% of the votes to E1.3 in Table 4.6 are not against change. For FUQ2 we use the same calculation but for votes regarding E2.4 of Table 4.7. If 70% of votes are not against the change, it makes sense to refine or validate it, as this shows most users are open for the change.

Acceptance of an ACR (T_{ACCEPT}). We accept an ACR based on the answers of the users to a FUQ3. We define an acceptance threshold T_{ACCEPT} with the same reason as for T_{CHANGE}. An ACR is accepted when at least 70% of the votes to E3.4 or E3.5 in Table 4.8 are not against change. After accepting an ACR, we derive changes to existing requirements or create new requirements (see Section 4.5.4.4).

Figure 4.4 shows how we derive, select and ask FUQ iteratively. The start of Figure 4.4 are the topics that are extracted in Section 4.5.3.

Figure 4.4: UML activity diagram representing the process to iteratively derive, select and ask FUQ. DSA=Derive, select and ask



FUQ1. We derive and ask a FUQ1, if only NACR are mapped to the topic Ⓐ. After asking the FUQ1 we check whether T_{CHANGE} is fulfilled Ⓑ. If it is fulfilled, we check whether the given freetext to E1.4 in Table 4.6 contains ACR Ⓒ. We cannot ask any more FUQ if the freetext to E1.4 does not contain ACR. If it does contain ACR, we map the ACR to the topic Ⓓ. When the topic contains at least two ACR, it is now the basis for asking a FUQ2. If it contains only one ACR, it is the basis for asking a FUQ3.

FUQ2. We derive and ask a FUQ2, if at least two ACR are mapped to the topic ①②. After asking the FUQ2, we check whether T_{CHANGE} is fulfilled ③. If it is fulfilled, we analyze which ACR is the most voted one ④ by checking which ACR of E2.4 of Table 4.7 is selected most often by the users. As it is now clear which ACR is most voted, the topic is now the basis for asking FUQ3 ⑤.

FUQ3. We derive and ask FUQ3 either if only one ACR is mapped to the topic ❶❷ or if the most voted ACR of the topic is already identified ⑤. If the FUQ3 uses a mockup ❸, we analyze the votes of the ACR regarding E3.4 of Table 4.8. If the FUQ3 does not use a mockup ❸, we analyze the votes of the ACR regarding E3.5 of Table 4.8. Only if T_{ACCEPT} is fulfilled ❹, we derive requirements.

Instance decision
In SMART-AGE we conduct two rounds of asking FUQ, each round containing a first and second iteration. We use two rounds to spread out the effort of answering questions for users over time.

4.5.4.4 REengs change or create requirements

The REengs change or create a requirement based on a validated ACR of a FUQ3. For example, when a FUQ3 validated the ACR "I don't want to see advertisements when opening a link in SV", then we would adjust the description of the corresponding requirement, in this case *SF: displayLink*, to include the constraint that advertisements should be blocked when opening a link. When a FUQ3 validated the ACR "I want to edit my answer after I submitted it in SF", then we would create a new requirement *SF: editSubmittedAnswer (U)* that allows to edit an already submitted answer.

4.6 Addressing the problems

We summarize in Table 4.10 how we address the problems *P1* (*P1.1*, *P1.2*, *P1.3*), *P2* and *P3*.

Table 4.10: Addressing the problems

P1.1 A lot of feedback can be collected from a lot of users
Addressed by: • Asking all users a variety of IQ at multiple times • Letting users submit messages and comments
P1.2 Feedback can be mapped to requirements effortlessly
Addressed by: • Asking IQ that are associated with requirements
P1.3 Feedback contains change requests
Addressed by: • Asking all users a variety of IQ, especially IQ that ask improvements about the system and its functional and non-functional requirements, as well as adaptive IQ that ask about reasons for inactivity and proposals for improvements.
P2 Control of timing of feedback collection
Addressed by: • Asking the IQ at specific defined times
P3 Support of change requests among users
Addressed by: • Asking three different types of FUQ

4.7 Conclusion

To summarize, the process to collect feedback and derive requirements is divided into two parts: the process to collect feedback through IQ and the process to derive

requirements through FUQ. In the process to collect feedback through IQ, the REengs start by selecting IQ from our proposed IQ. If adaptive IQ are selected, usage data needs to be recorded. After the users have access to the app(s), REengs ask IQ and users can answer or skip these IQ. Users may also send messages and comments. Based on the collected feedback, the process to derive requirements through FUQ can be conducted. REengs prepare the feedback by extracting change requests, mapping these change requests to requirements and topics. Building on this prepared feedback, REengs derive requirements by iteratively deriving, selecting and asking FUQ, while users can answer or skip these FUQ. Lastly, the REengs are able to change and create requirements based on the answers to the FUQ.

Chapter 5

smartFEEDBACK (SF) - Platform that supports the process

This Chapter describes the requirements of SF in Section 5.1 and the design and implementation of SF in Section 5.2. The quality assurance of SF is described in Section 5.3. We also created a handbook that explains the complete user interface of SF with screenshots and descriptions in B.5 in the appendix.

5.1 Requirements

We use Task and Object-oriented Requirements Engineering (TORE) by (Paech and Kohler, 2004) to define the requirements. TORE is a method to unify requirements engineering and object-oriented software development into a single conceptual model. TORE identifies 16 distinct types of decisions that are organized into four levels of abstraction. The levels along with their explanation and artefacts are presented in Table 5.1.

Table 5.1: TORE levels

TORE Level	Explanation	Artefacts	Described in
Task	Describes the reasons why users are motivated to use the software.	• User Roles • User Tasks • Subtasks	Section 5.1.1
Domain	Outlines the activities users need to perform as part of their work.	• Domain data model	Section 5.1.2
Interaction	Specifies how users should interact with the system to complete their tasks.	• System Functions • Workspaces	Section 5.1.3

System	Details the decisions related to the internal workings of the application and the user interface.	• Virtual Windows	Section 5.1.4

For the task level we present user roles, user tasks and subtasks in Section 5.1.1. For the domain level we describe the domain data model in Section 5.1.2. For the interaction level we present the system functions and workspaces in Section 5.1.3. For the system level we show virtual windows in Section 5.1.4.

5.1.1 Task level

When designing a new system, it's important to first understand the users who will interact with the software. This understanding helps create a system that meets their needs, which is necessary for its success. The insights about the users are captured in user role models. We present the user roles, along with their user tasks, success criteria, knowledge/experience/capabilities and communication partners in Table 5.2. The task information outlines what the users aim to achieve by using the software. The success criteria define the conditions that must be fulfilled by these tasks for the software to be deemed successful. The knowledge/experience/capabilities Section offers more information regarding the context of the users. The communication partners Section describes who the users communicate with.

Table 5.2: User roles with User Tasks

Title	User Tasks (UT)	Success criteria	Knowledge Experience Capabilities	Communi-cation partners
REeng	UT1(R): Collect feedback about apps and derive new requirements or (-changes)	New requirements or -changes can be derived from the feedback	High experience with technology	App stakeholders
Platform User	UT1(U): Submit feedback in the form of answers to questions, messages and comments to improve the application	Little effort is required to give feedback and feedback impact is reflected to users	Little to high experience with technology	REeng

There are two user roles, the REeng and the platform user. The task of the REeng is to collect feedback about apps and to derive new requirements or -changes. The task is successful when new requirements or -changes can be derived from the feedback. The REeng has high experience with technology and communicates on a regular basis with

the stakeholders of the apps. The task of the platform user is to submit feedback in the form of answers to questions, messages and comments to improve the application. This task is successful when little effort is required from the user to give feedback and the impact of the feedback is reflected to the user. The user has either little or high experience with technology and communicates with the REeng by sending feedback.

Descriptions for the subtasks are created to further elaborate on the user tasks. The subtasks for the platform user are shown in Table 5.3 and for the REeng in Table 5.4. Each subtask is given a name and a detailed description that provides further clarification. Example solutions are suggested to help support the subtask, outlining specific ways to implement it in software. These example solutions reference the specific system functions that are explained in Section 5.1.3.2 in detail. The variants **Section** specifies any variations of the subtasks that might differ slightly. The problem Section describes problems that may arise during the execution of the subtasks, with example solutions for these problems also provided.

Table 5.3: Subtasks for UT1(U)

Role: Platform User (abbreviated as "user" in the following)

Task: UT1(U)

Step (ST) / Problem (PB)	Example solution
UT1S1(U): Submit feedback	
UT1S1(U)_ST1: The user can send an answer, message or comment. **Necessary for process steps:** • Users answer or skip IQ • Users send messages and comments • Users answer or skip FUQ	The user can submit an answer, message and comment. This is supported through: • *SF: displayQuestion (U)* • *SF: submitAnswer (U)* • *SF: submitMessage (U)* • *SF: submitComment (U)*
UT1S1(U)_PB1.1: The users forget to give feedback	The users are reminded to give feedback through *SF: remindUser(U)*
UT1S1(U)_PB1.2: The users may not understand the question or may not like to give an answer to the question	The users can skip a question through *SF: skipAnswer (U)*
UT1S1(U)_PB1.3: The users may not like to use the keyboard to submit a longer message	The users can add an audio recording to a message through *SF: addAudioRecording (U)*

UT1S1(U)_PB1.4: The user forgets how to submit feedback	The user can read a brief explanation of SF through *SF: displayInstructions (U)*
UT1S2(U): Manage existing feedback	
UT1S2(U)_ST1: The user can manage answers grouped by question, messages and comments. **Necessary for process steps:** • Users send comments (This requires to see submitted answers or messages) *Indirectly*: This decreases the chance that users send duplicate feedback. Furthermore, having an overview is helpful for the user to keep track of his/her feedback in general.	The presentation of answers, messages and comments is supported through: • *SF: displayQuestionsWithAnswers(U)* • *SF: displayAnswersForQuestion (U)* • *SF: displayMessages (U)* • *SF: displayComments (U)* The presentation of the details of an answer and message is supported through: • *SF: displayAnswerDetails (U)* • *SF: displayMessageDetails (U)* The user can sort feedback: • *SF: sortQuestions (U)* • *SF: sortAnswers (U)* • *SF: sortMessages (U)* The user can filter feedback: • *SF: filterAnswersOfQuestion (U)* • *SF: filterOnlyMessagesOrAnswers(U)* • *SF: filterMessagesOrQuestions (U)*
UT1S2(U)_PB1.1: When a large amount of answers or comments is submitted, it can be difficult for the user to keep an overview over the feedback.	Feedback is presented in batches and the user can load a new batch of feedback. This is supported through: • *SF: displayMoreQuestionsWithAnswers (U)* • *SF: displayMoreAnswersForQuestion (U)* • *SF: displayMoreComments (U)*
UT1S2(U)_PB1.2: The user forgets how to manage answers grouped by question, messages and comments.	The user can read a brief explanation of SF through *SF: displayInstructions (U)*
UT1S2(U)_ST2: The user can see a history of all his/her answers and messages **Necessary for process:** *Indirectly*: This prevents sending duplicate feedback. Furthermore,	The presentation of answers and messages is supported through: • *SF: displayAnswersInHistory (U)* • *SF: displayMessagesInHistory (U)*

having an overview about when which feedback was sent helps the user to keep track of his/her feedback	The presentation of the details of an answer and message is supported through: • *SF: displayAnswerDetailsInHistory (U)* • *SF: displayMessageDetailsInHistory (U)* The user can filter answers and messages: • *SF: filterAnswersAndMessagesInHistory (U)*
UT1S2(U)_PB2.1: When a large amount of answers or messages is submitted, it can be difficult for the user to keep an overview over the history of feedback.	Feedback is presented in batches and the user can load a new batch of feedback. This is supported through: • *SF: displayMoreAnswersInHistory (U)* • *SF: displayMoreMessagesInHistory (U)*
UT1S2(U)_PB2.2: The user forgets how to check the history of all his/her answers and messages	The user can read a brief explanation of SF through *SF: displayInstructions (U)*

The system functions are designed to address various user needs and technical constraints to enhance the overall user experience. Due to space limitations, we cannot display detailed information for multiple answers, messages or comments on a single screen. Therefore, we provide a condensed view through functions like *displayAnswersForQuestion(U)*, *displayAnswersInHistory(U)*, *displayComments (U,R)*, *displayMessages(U)* and *displayMessagesInHistory(U)***.** For users who want to know more detailed information, we use functions such as *displayAnswerDetails(U)***,** *displayAnswerDetailsInHistory(U)***,** *displayMessageDetails(U)***,** and *displayMessage-DetailsInHistory(U)*. Additionally, to manage screen space more effectively, we allow content to be loaded incrementally, reducing the number of elements displayed simultaneously. We also offer audio recording options to support older adults who may face motor or vision challenges, as speaking is often easier than typing. The ability to skip questions accommodates users who may not understand certain questions or who may not have previously used a particular functionality. To further support our users, the reminder functionality assists those who wish to provide feedback but might occasionally forget. Finally, by differentiating between answers, messages, and comments, we address both reactive users who prefer responding to prompts and proactive users who wish to initiate communication.

Table 5.4: Subtasks for UT1(R)

Role: REeng

Task: UT1(R)

Step (ST) / Problem (PB)	Example solution
UT1S1(R): Manage questions	

UT1S1(R)_ST1: The REEng manages all existing questions **Necessary for process steps:** • REengs ask IQ • REengs ask FUQ	The REeng can: • View questions: *SF: displayQuestions (R)* • Sort questions: *SF: sortQuestions (R)* • Search questions *SF: searchQuestions (R)* • Edit a question: *SF: editQuestion (R)* • Schedule a question: *SF: scheduleQuestion (R)* • Set metric for an adaptive question: *SF: configureAdaptiveQuestion (R)* • Enable a question: *SF: enableQuestion (R)* • Disable a question: *SF: disableQuestion (R)*
UT1S1(R)_ST2: The REEng creates a new question **Necessary for process steps:** • REengs ask IQ • REengs ask FUQ	The REeng can create a new question. This is supported through *SF: createQuestion (R)*
UT1S2(R): Manage given feedback	
UT1S1(R)_ST1: The REeng can manage the answers and messages given by the users **Necessary for process steps:** • REengs derive FUQ	The REeng can: • View all answers of all users: *SF: displayAnswers (R)* • View the details of an answer: *SF: displayAnswerDetails (R)* • Sort the answers of all users based on aspects specified in *SF: sortAnswers (R)* • Filter answers of all users for a question based on aspects specified in *SF: filterAnswersOfQuestion (R)* • View all messages of all users: *SF: displayMessages (R)* • View the details of a message: *SF: displayMessageDetails (R)* • Sort the messages of all users based on aspects specified in *SF: sortMessages (R)* • Filter messages of all users based on aspects specified in *SF: filterMessages (R)* • Filter questions and messages based on aspected specified in *SF: filterQuestionsAndMessages (R)*
UT1S1(R)_PB1.1: When a large amount of feedback is submitted, it can be difficult for the	The REeng can lose the overview when all feedback is presented at once. This is why feedback is presented in batches and the REeng can load a new batch of feedback. This is supported through: • *SF: displayMoreAnswers (R)*

REeng to keep an overview over the feedback.	• *SF: displayMoreMessages (R)* • *SF: displayMoreComments (R).*
UT1S1(R)_ST2: The REeng can manage the comments on given feedback **Necessary for process steps:** • REengs derive FUQ	The REeng can: • View all comments of an answer or message: *SF: displayComments (R)* • Add a comment to an answer or message: *SF: submitComment (R)*
UT1S3(R): View study partners	
UT1S1(R)_ST1: The REeng can manage the users **Necessary:** To track how many users use SF	The REeng can: • view all users that use SF: *SF: displayUsers (R)* • filter the users as specified in *SF: filterUsers (R)*

We display feedback in batches for the REeng as well through *displayMoreAnswers (R)*, *displayMoreMessages (R)*, and *displayMoreComments (R)*, preventing information overload for the REeng. We provide functions like *createQuestion (R)*, *editQuestion (R)*, *scheduleQuestion (R)* or *configureAdaptiveQuestion (R)* which are essential to support the process defined in Section 4.4. Furthermore, we offer viewing and filtering of users through *displayUsers (R)* and *filterUsers (R)* which helps in identifying problems during the study.

5.1.2 Domain level

We present the domain data model in Figure B.3.1. The domain data model defines the key entities from the task descriptions and their relationships, without using any implementation or solution-specific language. The entities are represented through rectangles with the entity's name displayed at the top. The model is further refined with associations between entities.

5.1.3 Interaction level

We present the workspaces and the UI structure diagrams in Section 5.1.3.1. The system functions are presented in Section 5.1.3.2.

5.1.3.1 Workspaces

Workspaces provide a way for documenting which data and functions are used within a specific context and are accordingly organized on the user interface. We describe the data and the system functions for each workspace of the user in Table 5.5 and for the REeng in Table 5.6.

Table 5.5: Workspaces for the user

Data	Description of functionality
W: QuestionView (U)	
+ questions: List<Question>	The user displays a question (*displayQuestion (U)*). The user can answer (*submitAnswer (U)*) and skip open questions (*skipAnswer (U)*).
W: MessageView (U)	
+ message: Message	The user can submit a message (*submitMessage (U)*). The message can contain an audio recording (*addAudioRecording (U)*) and a file (*addFile (U)*).
W: SentView (U)	
+ questions: List<Question> + messages: List<Message>	The user can see his/her answered questions (*displayQuestionsWithAnswers (U)*) and their submitted messages (*displayMessages (U)*). The user can load more answered questions (*displayQuestionsWithAnswers (U)*) and more messages (*displayMoreMessages (U)*). The user can sort the questions (*sortQuestions (U)*) and the messages (*sortMessages (U)*). The questions and messages can also be filtered (*filterMessagesOrQuestions (U)*). The user can navigate to the answers of a question (*navigateToAnswersForQuestion (U)*) and the user can navigate to a message (*navigateToMessageDetails (U)*).
W: DetailedQuestionView (U)	
+ question: Question + answers: List<Answer>	The user can see the question and its answers (*displayAnswersForQuestion (U)*). The user can load more answers for that question (*displayMoreAnswersForQuestion (U)*). The user can sort (*sortAnswersOfQuestion (U)*) and filter (*filterAnswersOfQuestion (U)*) the answers of the question. The user can also navigate to the answer (*navigateToAnswerDetails (U)*).
W: DetailedAnswerView (U)	
+ answer: Answer + comments: List<Comment>	The user can see his/her answer (*displayAnswerDetails (U)*) and its comments (*displayComments (U)*). The user can load more comments *displayMoreComments (U)*) and can navigate to a

	comment (*navigateToComment (U)*). The user can also sort comments (*sortComments (U)*).
W: DetailedMessageView (U)	
+ message: Message + comments: List<Comment>	The user can see his/her message (*displayMessageDetails (U)*) and its comments (*displayComments (U)*). The user can load more comments (*displayMoreComments (U)*) and can navigate to a comment (*navigateToComment (U)*). The user can also sort comments (*sortComments (U)*).
W: CommentView (U)	
+ comment: Comment	The user can add a new comment (*submitComment (U)*). The comment can contain an audio recording (*addAudioRecording (U)*).
W: HistoryView (U)	
+ answers: List<Answer> + messages: List<Message>	The user can see his/her answers (*displayAnswersInHistory (U)*) and his/her submitted messages (*displayMessagesInHistory (U)*). The user can load more answers (*displayMoreAnswersInHistory (U)*) and more messages (*displayMoreMessagesInHistory (U)*). The user can filter the answers and messages (f*ilterAnswersAndMessages (U)*). The user can navigate to an answer (*navigateToAnswerDetailsInHistory (U)*) and to a message (*navigateToMessageDetailsInHistory (U)*).
W: DetailedAnswerHistoryView (U)	
+ answer: Answer	The user can see his/her message (*displayAnswerDetailsInHistory (U)*).
W: DetailedMessageHistoryView (U)	
+ message: Message	The user can see his/her message (*displayMessageDetailsInHistory (U)*).
W: InstructionView (U)	
+ text: String	The user can see an explanation for the app (*displayInstructions (U)*).
W: SidebarView (U)	
+ links: List<String>	The user can navigate over links to the workspaces W: *QuestionView (U)*, *W: MessageView (U)*, *W: SentView (U)*, *W: HistoryView (U)* and *W: InstructionView(U).*
W: HeaderView (U)	
+ link: String	The user can navigate to the SMART-AGE portal (*navigateToPortal (U)*).

Table 5.6: Workspaces for the REeng

Data	Description of functionality
	W: QuestionView (R)
+ questions: List<Question>	The REeng can see all questions that exist (*displayQuestions (R)*). The REeng can load more questions (*displayMoreQuestions (R)*). The REeng can create a new question (*createQuestion (R)*). The REeng can filter (*filterQuestions (R)*), sort (*sortQuestions (R)*) and search questions (*searchQuestions (R)*). The REeng can enable (*enableQuestion (R)*) and disable (*disableQuestion (R)*) questions. The REeng can edit a question (*navigateToEditQuestion (R)*) and the REeng can schedule a question (*navigateToScheduleQuestion (R)*).
	W: ScheduleQuestionView (R)
+ question: Question	The REeng can schedule a question (*scheduleQuestion (R)*).
	W: CreateQuestionView (R)
+ question: Question	The REeng can create (*createQuestion (R)*) and edit (*editQuestion (R)*) a question
	W: ResultsView (R)
+ questions: List<Question> + messages: List<Message>	The REeng can see all questions that contain at least one answer (*displayQuestionsWithAnswers (R)*) and all submitted messages (*displayMessages (R)*). The REeng can load more questions (*displayMoreQuestionsWithAnswers (R)*) and more messages (*displayMoreMessages (R)*). The REeng can sort questions (*sortQuestions (R)*) and messages (*sortMessages (R)*). The REeng can filter questions and messages (*filterQuestionsAndMessages (R)*). The REeng can navigate to the answer for a question (*navigateToAnswersForQuestion (R)*) and to a message (*navigateToMessageDetails (R)*).
	W: DetailedQuestionView (R)
+ question: Question + answers: List<Answer>	The REeng can see the question and all of its answers from all users (*displayAnswersForQuestion (R)*). The REeng can load more answers for that question (*displayMoreAnswersForQuestion (R)).* The REeng can sort (*sortAnswersOfQuestion (R))* and filter (*filterAnswers-OfQuestion (R))* the answers of the question. The REeng can also navigate to the answer *(navigateToAnswerDetails (R)*).
	W: DetailedAnswerView (R)
+ answer: Answer	The REeng can see an answer (*displayAnswerDetails (R)*) and its comments (*displayComments (R)*). The REeng can load more comments (*displayMoreComments (R)*) and can navigate to a

+ comments: List<Comment>	comment (*navigateToComment (R)*). The REe*ng* can also sort comments (*sortComments (R)*).
W: DetailedMessageView (R)	
+ message: Message + comments: List<Comment>	The REeng can see a message (*displayMessageDetails (R)*) and its comments (*displayComments (R)*). The REeng can load more comments (*displayMoreComments (R)*) and can navigate to a comment (*navigateToComment (R)*). The REeng can also sort comments (*sortComments (R)*).
W: CommentView (R)	
+ comment: Comment	The REeng can add a new comment (*submitComment (R)*). The comment can contain an audio recording (*addAudioRecording (R)*).
W: UserView (R)	
+ users: List<User>	The REeng can see all users (*displayUsers (R)*). The REeng can load more users (*displayMoreUsers (R)*) and can filter users (*filterUsers (R)*).
W: SidebarView (R)	
+ links: List<String>	The REeng can navigate over links to the workspaces *W: QuestionView (R)*, *W: ResultsView (R)*, *W: UserView (R)*.

General decision
We designed the workspaces so that the user or REeng is not overloaded with information during their tasks. For example, we separate information regarding questions, answers and comments into multiple workspaces instead of showing the whole information in one workspace. We also allow quick navigation between the workspaces to support the user and REeng in their tasks. For this we introduce the *W: SidebarView (U, R)* which contains links to important workspaces.

The UI structure diagram for the user is shown in Figure 5.1 and for the researcher in Figure 5.2. The system functions that allow navigation between workspaces are underlined.

Figure 5.1: UI structure diagram for the user

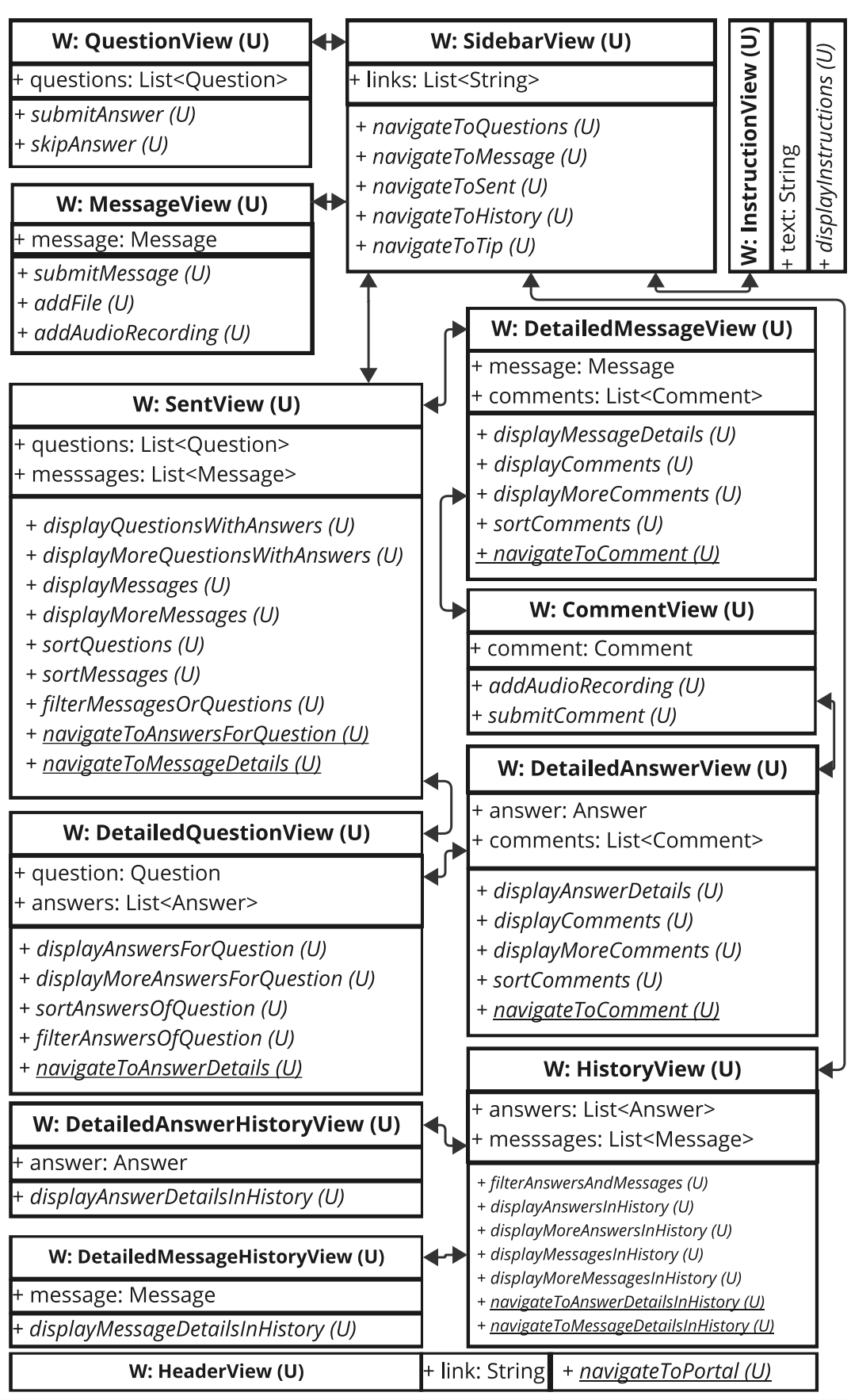
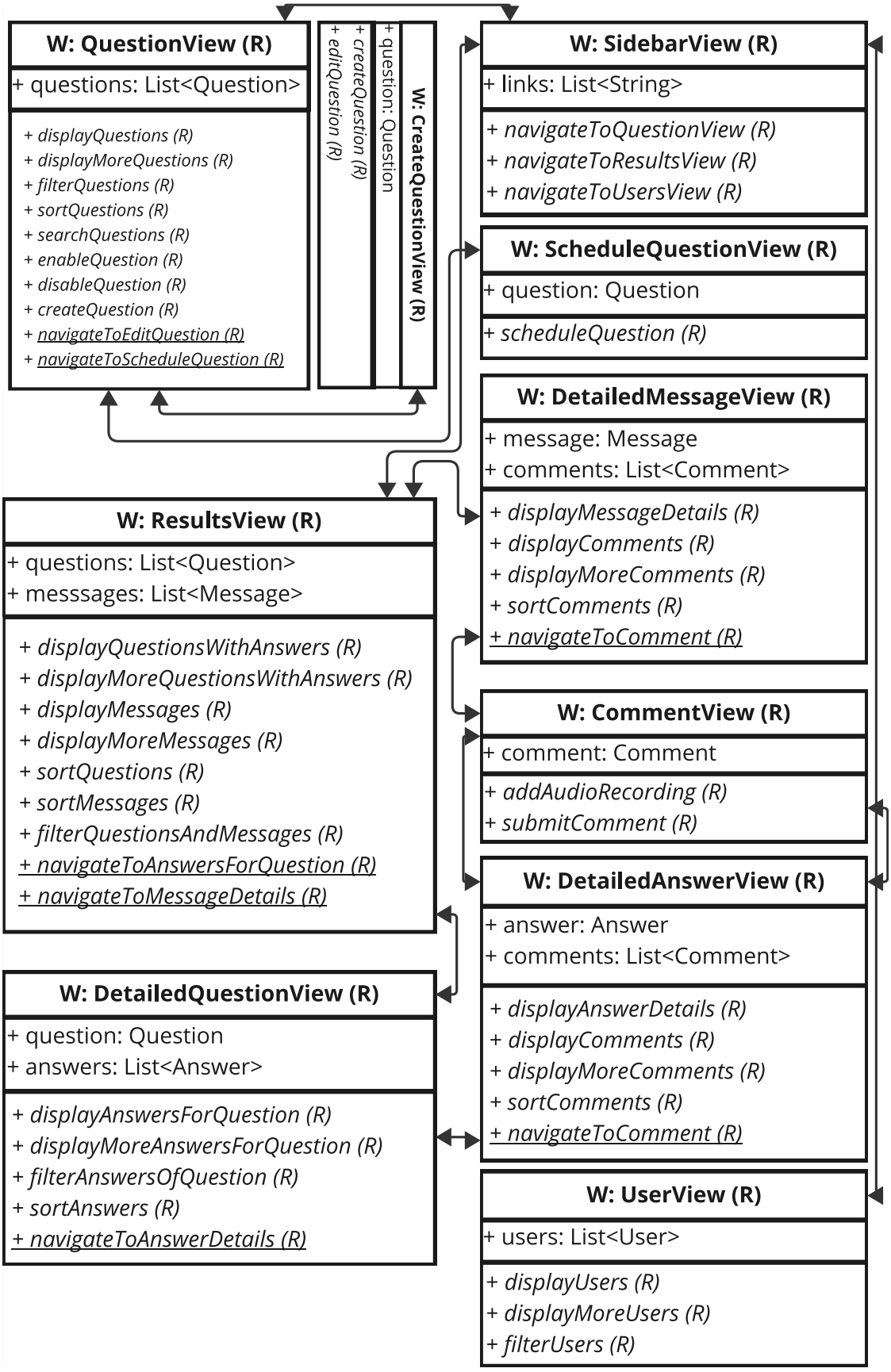


Figure 5.2: UI structure diagram for the REeng



5.1.3.2 System functions

The system functions are listed for the user in Table B.3.1 and for the REeng in Table B.3.2 in the appendix. They are described along with their name, their description, their pre- and post-conditions, their input and output, as well as exceptions that occur and rules which define how the function should operate.

5.1.4 System level

Based on the workspaces that are mapped to system functions, we can create virtual windows (mockups) to describe how the user interface of SF should look like. We give an example for a virtual window for the workspace W: DetailedQuestionView (U) in Figure 5.3.

Figure 5.3: Virtual window for *W: DetailedQuestionView (U)* including *W: HeaderView (U)* and *W: SidebarView (U)*

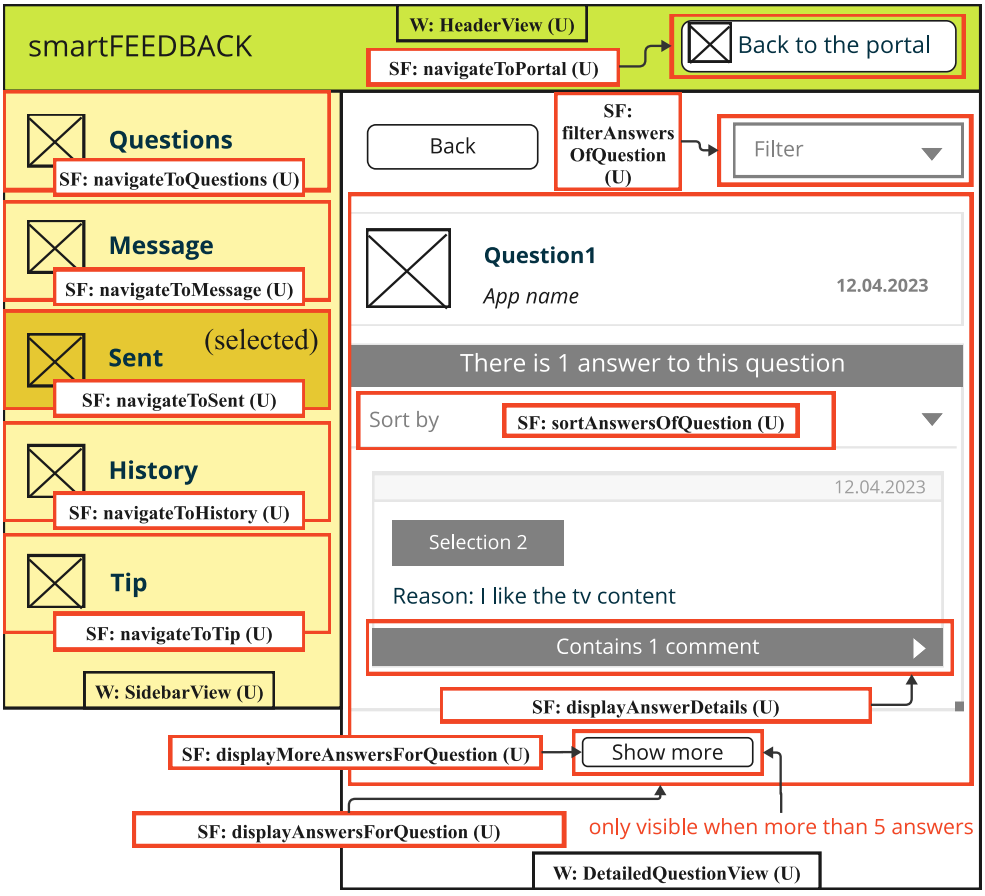


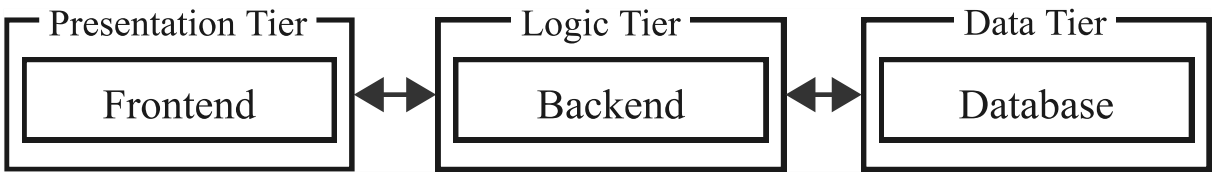
Figure 5.3 shows the virtual window *W: DetailedQuestionView (U)* along with the virtual windows of *W: SidebarView (U)* and *W: HeaderView (U)*. We decided to show the two virtual windows for the workspaces *W: SidebarView (U)* and *HeaderView (U)* always to the user, because *SidebarView (U)* allows navigation to other workspaces and *HeaderView (U)* allows navigation to the SMART-AGE portal, which are navigations

that should be easily accessible as they are done frequently. Figure 5.3 also shows the mapping between the user interface and the system functions (red). We don't describe the virtual windows in detail here, because we describe the implemented virtual windows (screenshots) for all workspaces in detail in B.5 in the appendix. The screenshots do not show meaningful differences to the virtual windows, so an extra explanation is not necessary. The only exception is the virtual window of *CreateQuestionView (R)* in Figure B.2.13 in the appendix. This virtual window we didn't implement exactly how we planned, because we could use an existing user interface from a library (see Section 5.2.2.3). We list all virtual windows for the user in Figure B.2.1 - Figure B.2.11 and for the REeng in Figure B.2.12 - Figure B.2.20 in the appendix. In general, we designed the virtual windows with respect to our target group of older adults. We use large fonts and interaction elements to make interaction easier despite potential vision or motor difficulties.

5.2 Design and implementation

The design of SF follows the widely adopted three-tier architecture used in software engineering. The three tiers are typically referred to as the presentation tier, logic tier and data tier. Figure 5.4 gives an overview over the architecture. The presentation tier contains the frontend of the application, the logic tier contains the backend (business logic) and the data tier contains the database. The frontend communicates with the backend in both directions. The frontend receives data from the backend to show it to the user and the backend receives data from the frontend to do calculations with it and to store it in the database. The backend also receives data from the database to do calculations with it. The database receives the data from the backend and stores it. For the presentation tier and the logic tier we describe in Sections 5.2.1 and 5.2.2 what technology we use, what our design is and what the implementation is. We explain the technology selection before the design, because the design is dependent on the chosen technology. We don't describe the details of the data tier, because the design of the database is automatically derived from the entities of the backend, which we explain in detail in Section 5.2.2.2. The only relevant information is that we use a PostgreSQL[12] database and that we tune its settings through PGTune[13] to enhance its performance.

Figure 5.4: Overview over the three-tier architecture of SF. Arrows represent communication direction.



[12] https://www.postgresql.org/

[13] https://pgtune.leopard.in.ua/

5.2.1 Presentation tier

We describe the presentation tier in this Section. The presentation tier represents the frontend of SF. Section 5.2.1.1 describes the technology selection, Section 5.2.1.2 the design and Section 5.2.1.3 the implementation.

5.2.1.1 Technology selection

The author has previous experience with the frontend technologies Vue.js[14] and Google Web Toolkit (GWT)[15]. The decision which frontend technology to use is guided by the consideration of their advantages and disadvantages which are shown in Table 5.7.

Table 5.7: Advantages and disadvantages between Vue.js and GWT

Advantages	Disadvantages
Vue.js	
Reactive data binding (UI reflects data changes automatically)	No type safety
Strong community support (Survey framework SurveyJS[16] available)	
Performance (Fast loading times)	
GWT	
Type safety	Slower development cycle (Java to Javascript compilation adds overhead)
Rich set of widgets (prebuilt components)	Outdated (decreased relevance and little community support)

Vue.js offers the advantage of reactive data binding, which makes it possible for the UI to automatically update when data changes occur. Furthermore, it has a strong community support with extensive resources and plugins. Specifically, it offers support for the open source javascript library SurveyJS, which allows to create question forms in a flexible way. Lastly, Vue.js allows for fast loading times. Compared to Java, which is used in GWT it does not offer type safety. Next to type safety, GWT offers a rich set of widgets (prebuilt components) which can be used to build the UI. However GWT faces important disadvantages. It has a slower development cycle compared to Vue.js, as Java needs to be compiled to Javascript. Furthermore, GWT is increasingly seen as outdated technology, having less relevance and little community support.

[14] https://vuejs.org/

[15] https://www.gwtproject.org/

[16] https://surveyjs.io/

General decision
We chose Vue.js over GWT mainly because the reactive data binding eliminates the need to refresh the UI when needed, which is very convenient. Furthermore, the development time is reduced by relying on SurveyJS for creating question forms and by not needing to wait for the compilation of Java code to Javascript code, that is necessary for GWT.

5.2.1.2 Design

Vue.js uses components as building blocks of an application. A component in Vue.js is a reusable and isolated unit of code that contains the *structure of the user interface* expressed as HTML (Hyper Text Markup Language), its *style* expressed in CSS (Cascading Style Sheets) and its *data and behavior* expressed in JavaScript. The data of a component can be changed dynamically depending on the application state. We derive components for the design of the user interface based on the virtual windows of Section 5.1.4.

Figure 5.5 shows that only three components are needed to represent the virtual windows for the workspaces ❶ W: DetailedQuestionView (R) and ❷ DetailedQuestionView (U). The components (❸) are HeaderView, SidebarView and DetailedQuestionView. We can create reusable components for parts of the virtual windows that always have the same structure. For example, the sidebar always lists links in the form of a list. Only the links (data) changes for example depending on whether the application is used as a user or as a REeng. The components HeaderView and SidebarView are used throughout all virtual windows. Only the component DetailedQuestionView is switched out for other components depending on the application state (e.g. when the user clicks a link on the sidebar). The user interface of SF can be thus generalized to the structure in Figure 5.6.

Figure 5.5: Representation of the virtual windows of the workspaces W: DetailedQuestionView (R) and W: DetailedQuestionView (U) through components.

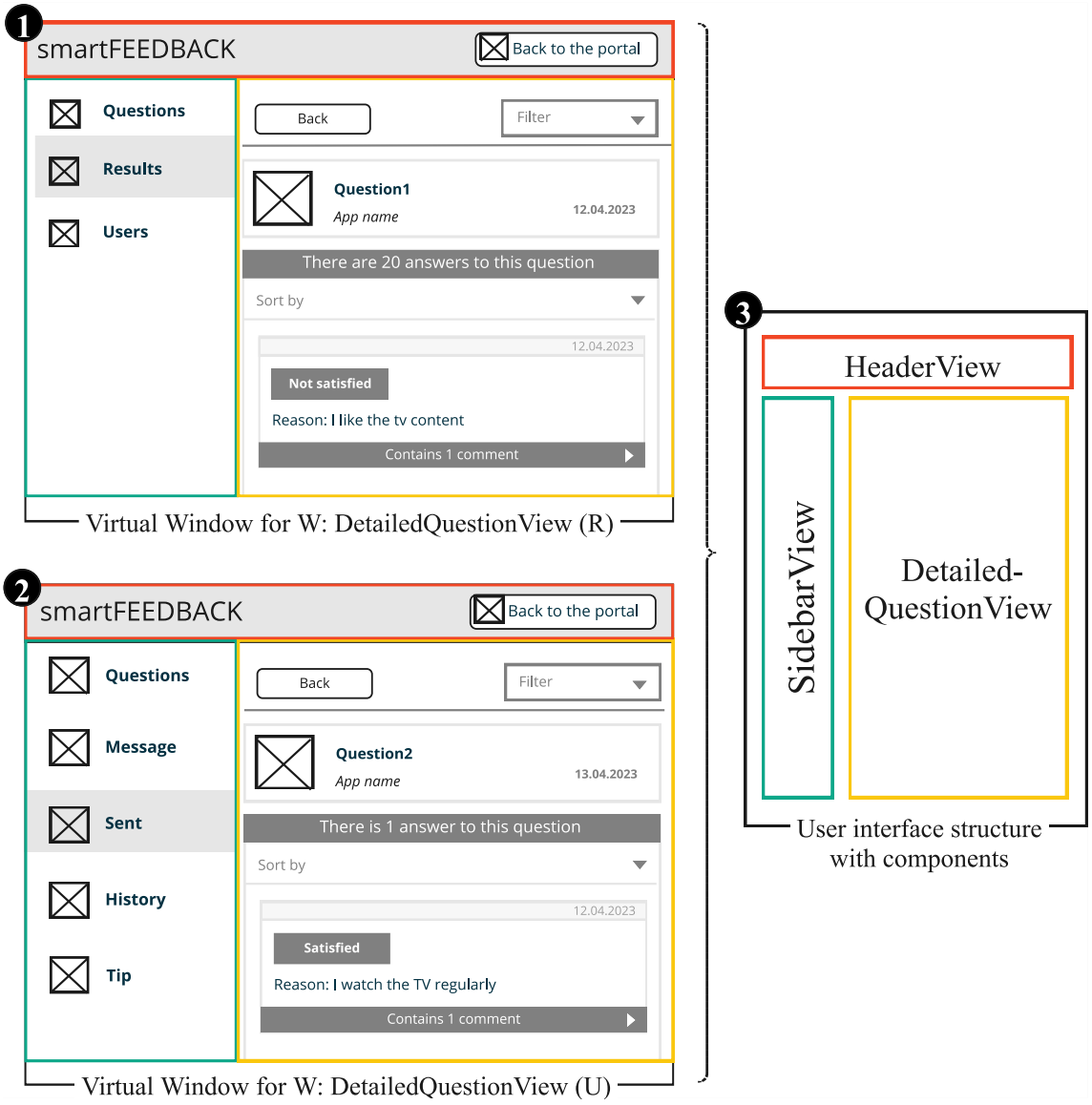


Figure 5.6: Structure of the user interface of SF. Dashed line means that this part of the user interface is represented by different components.

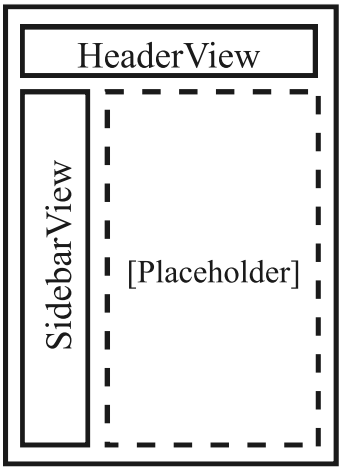


Table 5.8 shows for all virtual windows which component is representing the placeholder part of the user interface.

Table 5.8: Virtual windows for workspaces and their component that is used instead of the placeholder part of the user interface.

Virtual window for workspace	Component that is used instead of placeholder
W: QuestionView (U)	QuestionView
W: MessageView (U)	MessageView
W: SentView (U) **W: ResultsView (R)**	FeedbackMainView
W: DetailedQuestionView (U) **W: DetailedQuestionView (R)**	DetailedQuestionView
W: DetailedAnswerView (U) **W: DetailedAnswerView (R)**	DetailedAnswerView
W: DetailedMessageView (U) **W: DetailedMessageView (R)**	DetailedMessageView
W: CommentView (U) **W: CommentView (R)**	CreateCommentView
W: HistoryView (U)	HistoryView
W: DetailedAnswerHistoryView (U)	DetailedFeedbackHistoryView
W: DetailedMessageHistoryView (U)	DetailedMessageHistoryView
W: InstructionView (U)	InstructionView
W: QuestionView (R)	ManageQuestionsView
W: ScheduleQuestionView (R)	ScheduleQuestionView
W: CreateQuestionView (R)	CreateQuestionView
W: UserView (R)	UserView

In total, we need 17 components to represent the virtual windows, including SidebarView and HeaderView.

5.2.1.3 Implementation

In this Section we present a mockup of an implemented virtual window and we explain how we use the frameworks Vuetify[17], Survey.js[18], VueApollo[19], VueRouter[20] and Axios[21] to implement the components. We list all implemented virtual windows in the appendix (Figure B.2.21 - Figure B.2.40). These implemented virtual windows represent screenshots of SF. As screenshots are often pixelated and font can hardly be read, we vectorized the screenshots, so that also smaller font can be read through zooming in.

[Reusing existing user interface elements with Vuetify]

To explain how we use the framework Vuetify to reuse existent user interface elements for the implementation of our components, Figure 5.7 shows a mockup of the implemented virtual window *W: HistoryView (U)*. The colored areas highlight the specific Vuetify user interface elements that we have reused. The figure shows the components HeaderView on the top, SidebarView on the left and HistoryView on the right. The HeaderView component contains an app-bar element[22] to show the SF text and a button element[23] for the navigation button to the portal. The SidebarView is using a navigation drawer[24] element for the navigation links and multiple icon elements[25] that decorate the navigation links. The HistoryView contains two data tables[26] to display the answers to questions and messages. The data table elements come with built-in functionalities, such as displaying a limited number of items at a time and offering pagination options. Other used Vuetify elements that we reuse in our components are selection menus[27] that we use for filtering, as well as spinners[28] to show progress during data loading. We don't list all Vuetify elements here, because it is easy to identify them into code. The vuetify components are all marked with the HTML tag <v>. All implemented components can be found in the folder *ui/app/src/components* folder in the repository of smartFEEDBACK[29] on branch dissertation. The repository of smartFEEDBACK will be open source after SMART-AGE study ends in Q2/Q3 of 2025.

[17] https://vuetifyjs.com/en/

[18] https://surveyjs.io/

[19] https://apollo.vuejs.org/

[20] https://router.vuejs.org/

[21] https://axios-http.com/docs/intro

[22] https://vuetifyjs.com/en/components/app-bars/

[23] https://vuetifyjs.com/en/components/buttons/

[24] https://vuetifyjs.com/en/components/navigation-drawers/

[25] https://vuetifyjs.com/en/components/icons/

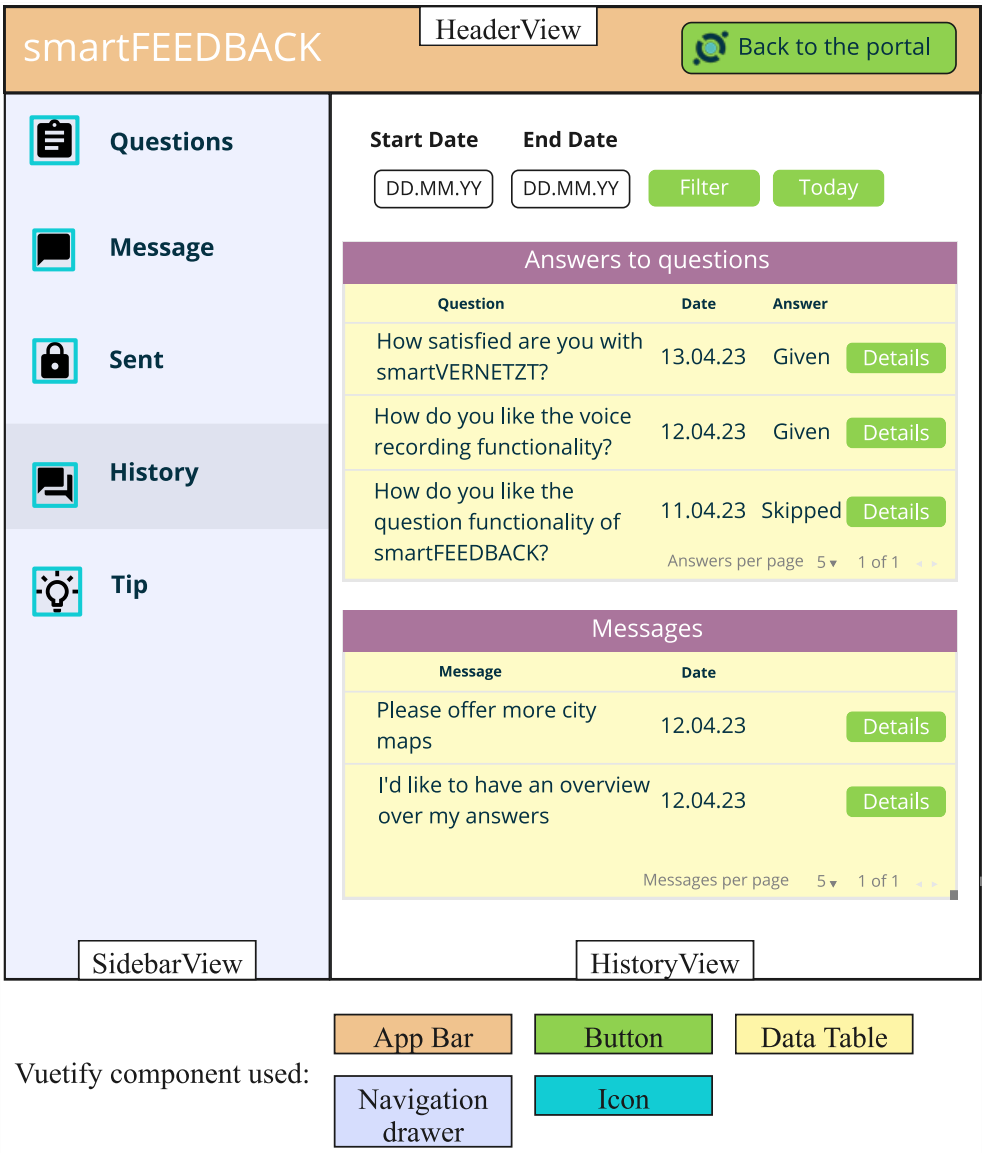
[26] https://vuetifyjs.com/en/components/data-tables/

[27] https://vuetifyjs.com/en/components/selects/

[28] https://vuetifyjs.com/en/components/progress-circular/

[29] https://github.com/SMARTAGE21/smartage-feedback-app

Figure 5.7: Mockup of the implementation of the virtual window *W: HistoryView (U)* along with colored areas to show what Vuetify components are used.



[Creating questions with Survey.js]
We use the framework Survey.js in SF.

General decision
We decided to use Survey.js to allow the REeng to easily set up and customize questions with a wide variety of question elements, such as text input, selection and, html content.

Using Survey.js as a framework to integrate questions into SF offers flexibility when it comes to extending the application in the future. If additional question types or more

complex question logic are required later on, these can be easily implemented without manually coding all functionalities or change the existing functionality. Therefore using Survey.js not only simplifies the initial development but also makes the application more adaptable in the long term.

[Communication with the backend through VueApollo]
VueApollo is a state management library for Vue.js applications that allows the communication from frontend to backend through a GraphQL API (Application Programming Interface). GraphQL is a query language for APIs that allows clients to request exactly the data they need from a server with one endpoint. GraphQL uses queries to fetch data and mutations to modify data, reducing the number of requests and the amount of data transferred, ensuring a fast communication between client and server.

We use queries to fetch data from the backend through GraphQL. We define the GraphQL queries in each component, which allows the components to request exactly the data they need. The component then automatically receives and displays the data, and if the data changes on the backend, the component updates to reflect those changes. For example, if a user opens the "Questions" screen in the sidebar, the corresponding Vue component (QuestionView) has a query that receives all open questions for that user from the backend.

We use mutations to send a request to change, add, or delete data on the backend. This is similar to how a query works, but instead of just fetching data, a mutation actively alters the data stored on the backend. We define the GraphQL mutations also directly within each Vue component, specifying the data to be modified and the new values to be applied. For example, if a user answers a question, the corresponding Vue component (QuestionView) has a mutation that sends the answer to the backend.

We describe the design and implementation of the queries and mutations in the in Section 5.2.2.2 of the logic tier, because the backend is responsible for the actual calculation that results from the queries and mutations.

[Sending monitoring data]
When using SF, we record each interaction that the user does with SF components as implicit feedback. To send implicit feedback we use Axios.

General decision
We chose Axios, because it is a popular and established popular JavaScript library used for making HTTP requests from the browser.

We first create an instance of Axios (httpClient) with a base URL for the API. This instance is used to manage HTTP requests throughout the application. The monitoringService object defines a method createEvent, which is responsible for sending implicit feedback to a central monitoring service that is responsible for receiving the implicit feedback from all apps used in SMARTAGE.

General decision
We use a central httpClient instance for SF encapsulates the logic for sending monitoring data in a reusable service, making it easy to send monitoring events from different components whenever necessary.

The monitoring service can be found in the folder *ui/app/src/services* in the repository[30] on branch dissertation.

[Navigation with VueRouter]
We use the library VueRouter for navigation.

General decision
We use VueRouter, because it is is the official routing library for Vue.js applications, enabling navigation between different screens without reloading the entire page.

We use VueRouter every time the component in the placeholder of the user interface switches (see Figure 5.6 in Section 5.2.1.2). For that we define routes, which are mappings between URL paths and each Vue component. For example the route for the overview of questions for the user is called with the URL path "/questions" in the browser. The vue component QuestionView is shown when the user navigates through the press on a user interface element to this path. The user interface elements that allow navigation through routes (e.g. buttons) use router links, which function like regular hyperlinks but allow for navigation without triggering a page reload. VueRouter intercepts the navigation and updates the browsers' URL while swapping out the old placeholder component for the new one. VueRouter also keeps track of the applications' history, allowing for backward and forward navigation.

The Vue router configuration along with all routes can be found in the folder *ui/app/src/router* in the repository[31] on branch dissertation.

[30] https://github.com/SMARTAGE21/smartage-feedback-app
[31] https://github.com/SMARTAGE21/smartage-feedback-app

5.2.2 Logic tier

We describe the logic tier in this Section. The logic tier represents the backend of SF. Section 5.2.2.1 describes the technology selection, Section 5.2.2.2 the design and Section 5.2.2.3 the implementation.

5.2.2.1 Technology selection

The author has previous experience with the backend technologies JavaEE (Java Platform, Enterprise Edition) and Java Spring Boot. The author has more experience with Java EE than Java Spring Boot. The decision on which backend technology to use is guided by the consideration of their advantages and disadvantages, which are listed in Table 5.9 and explained below.

Table 5.9: Advantages and disadvantages between Java EE[32] and Java Spring Boot[33]

Advantages	Disadvantages
JavaEE	
Well-established with robust tools and libraries	High amount of boilerplate code
	Requires external application server
Java Spring Boot	
Litte amount of boilerplate code	High learning curve
External application server not required	

Java EE is a widely used technology for building enterprise-level applications. One of its main advantages is that it is a well-established and uses robust tools and libraries. However, Java EE requires a high amount of boilerplate code, which makes the development process slower and more complex. Additionally, it requires an external application server, adding extra configuration and deployment effort.

Java Spring Boot, on the other hand, offers several benefits that make it an attractive alternative to Java EE. One big advantage is that it requires little boilerplate code, which simplifies the development process and makes it faster to build and deploy applications. Spring Boot also does not require an external application server, as it can run an embedded server, which further saves effort during development. However, Spring Boot has a high learning curve, especially for the author, because he has little previous experience with it.

[32] https://www.oracle.com/de/java/technologies/java-ee-glance.html

[33] https://spring.io/projects/spring-boot

General decision
We chose Java Spring Boot over Java EE because development effort is decreased a lot through the minimal boilerplate code and the advantage to not need an external server (-configuration). Although Java Spring Boot has a higher learning curve, the benefits of faster development cycles and easier deployment outweighed the downsides, making it the preferred choice for our backend technology.

5.2.2.2 Design

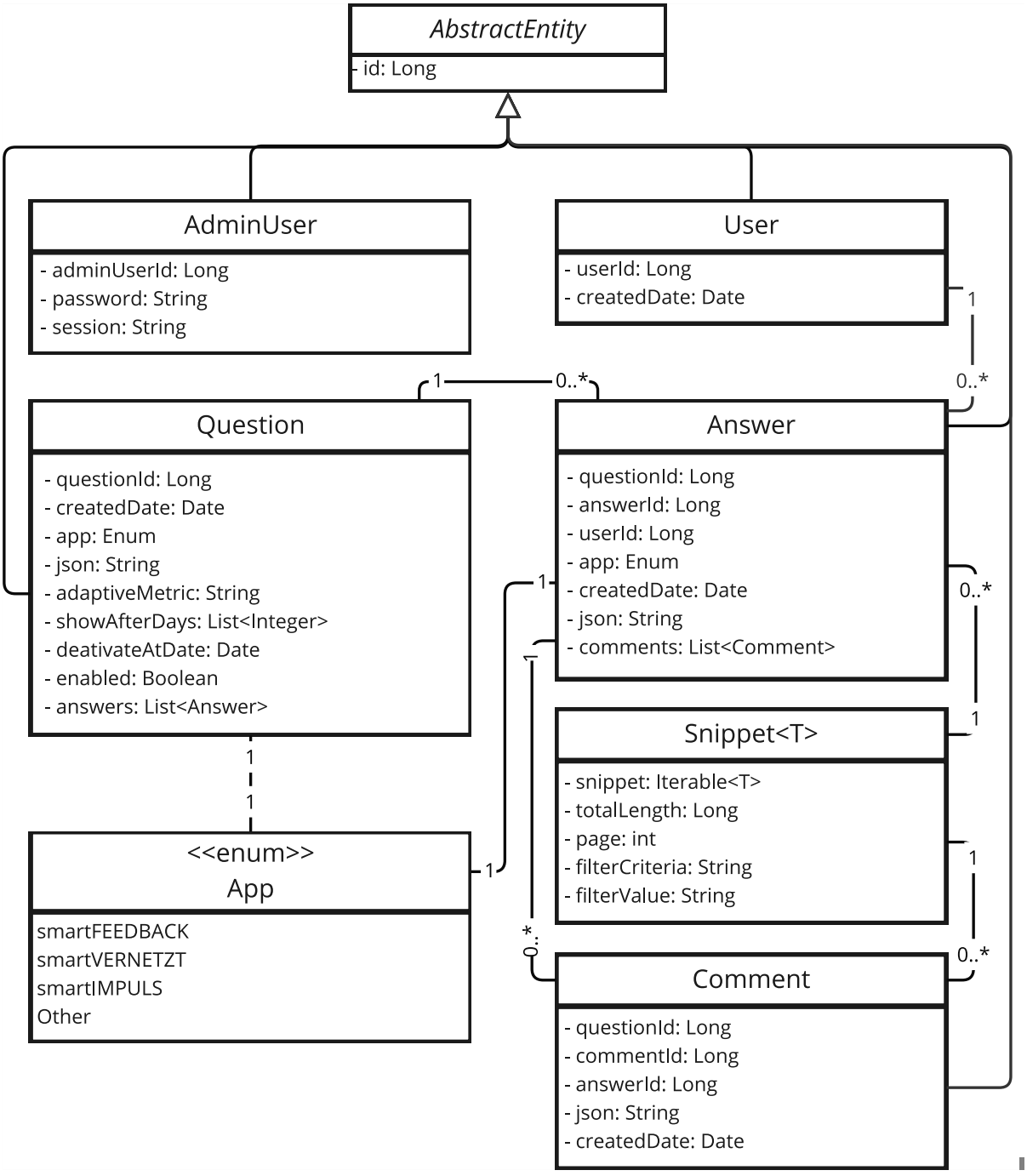
In this section, the design of the entities of the Java Spring Boot backend is described, which are the core data structures. We also describe the design of the GraphQL queries and mutations that are offered by the backend and accessed by the frontend. We also describe how we design the authentication of devices.

[Entities]
The class diagram shown in Figure 5.8 represents the entities for a Java Spring Boot backend and their relationships. The entities include *AbstractEntity*, *AdminUser*, *User*, *Question*, *AnswerOrMessage*, *App*, *Snippet* and *Comment*.

At the core is *AbstractEntity*, a base class that provides a common attribute, id, serving as a unique identifier for all entities inheriting from it. This abstract entity is extended by all other entities except *App* which represents an enum. The entity *AdminUser* represents the REeng user in the system and contains attributes such as *userId*, *password*, and *session* to manage user credentials and the user sessions. The entity *User*, also extends *AbstractEntity* and represents regular users of the system, with attributes like *userId* and *createdDate* to track the identity of the user and the date when the user started using SF.

The *Question* entity, inheriting from *AbstractEntity*, represents questions within the application and includes attributes such as *questionId*, *createdDate* (date of the creation of the *Question*), *app* (associated app of the question), *json* (content of the *Question* including the subquestions and answer options), *adaptiveMetric* (the metric that is used to determine the receivers of the question in case the question is adaptive), *showAfterDays* (list of days when the *Question* should appear), *deactivateAtDate* (date when question should be deactivated), *enabled* (determining whether the question is enabled or disabled) and *answers* (containing all answers for the question).

Figure 5.8: Class diagram for entities



The *Answer* entity, inheriting from *AbstractEntity*, represents an answer to a question. We also use the *Answer* entity to model messages. This way we can make use of the Survey.js framework functionality which allows us to configure the message structure through the form of a question. Concretely this means, that we can easily extend our message structure in the future, if we for example want to add new input fields or change the appearance of the message. *Answer* includes the attribute *questionId*. When *Answer* represents a message, this *questionId* is fixed, representing the fixed question that defines the structure of the message input form. When the entity represents an

answer, the *questionId* is the id of the corresponding question, which can be different depending on the answer. *Answer* also includes the attributes *userId* (representing the user that created the *Answer*), *app*, *creationDate* (representing the date when the *Answer* was created), *json* (content of the *Answer* including all values that were input for the subquestions of the question with id *questionId*) and *comments* (a list of all comments for the *Answer*).

We also model the entity *Comment* in a compatible way with Survey.js. This means that we can also flexibly change and extend the input field of a comment in the future.
The *Comment* entity has a *questionId* (representing the fixed question which defines the structure of the comment), *commentId* (the id of the comment), an *answerId* (representing the id of the associated *Answer*), json (content of the comment including all values that were input for the subquestions of the question with id *questionId*), as well as *createdDate* (representing the date of creation of the comment).

The *Snippet<T>* entity represents a generic, paginated collection of *Answer* or *Comment* entities. The *Snippet<T>* includes the actual entities in the attribute *snippet<T>*. Further attributes which are relevant for pagination are *totalLength* (needed to show the total number of entities for pagination) and *page* (indicating the current page that the entities in *Snippet<T>* belong to). The attributes relevant for filtering are *filterCriteria* and *filterValue.* An example for *filterCriteria* would be "creationDate" and an example for the filterValue would be "last_week", to filter entities which were created in the last week.

In terms of relationships, the diagram shows also that each *AdminUser* can manage multiple *Question* entities, while each *Question* is associated with exactly one *AdminUser*. Similarly, each User can be linked to multiple *Answer* entities, and each *Answer* is associated with one *User*. *Question* entities can have multiple related *Answer* entities, and each *Answer* can have multiple *Comment* entities.

[Queries and mutations]
For the user and the REeng to see data on the frontend, the data needs to be provided from the backend. This happens through queries as explained in Section 5.2.1.3. Table 5.10 presents an overview of the queries that are required to fetch different data in different workspaces used by the REeng or the user. The queries fetch entities that we describe in this Section under the caption "Entities". The queries are categorized by entity type (question (Q), answer (A), and comment (C)) and by the caller (REeng or user). Each entry in the table specifies also the query name and the reason why this query must exist.

Table 5.10: Queries regarding different entities
Entity legend: Q = Question, A=Answer, C=Comment

Entity	Query name	Reasons
		Used by REeng
Q	ADMIN_SNIPPET_ OF_QUESTIONS	Needed to fetch a paginated, filtered and sorted list of existent questions in *W: QuestionView (R)*
	ADMIN_QUESTION	Needed to fetch the details of a question in *W: DetailedQuestionView (R)*
	ADMIN_SNIPPET_OF_ QUESTIONS_WITH _ANSWERS	Needed to fetch a paginated, filtered and sorted list of questions in *W: ResultsView (R)*
A	ADMIN_ANSWER	Needed to fetch the details of an answer in *W: DetailedAnswerView (R)* and a message in *W: DetailedMessageView (R).*
	ADMIN_SNIPPET_ OF_ANSWERS	Needed to fetch a paginated, filtered and sorted list of answers. *W: DetailedQuestionView (R)*
C	ADMIN_SNIPPET_ OF_COMMENTS	Needed to fetch a paginated sorted list of comments in *W: DetailedAnswerView (R)* and *W: DetailedMessageView (R).*
		Used by user
Q	QUESTIONS	Needed to fetch a list of all open questions for the user in *W: QuestionView (U)*.
	QUESTION	Needed to fetch the details of a question in *W: DetailedQuestionView (U)*
	MESSAGE_ QUESTION	Needed to fetch the structure of the message input form in *W: MessageView (U)*.
A	ANSWER	Needed to fetch the details of an answer in *W: DetailedAnswerView (U)* and W: *DetailedAnswerHistoryView (U)*, as well as a message in *W: DetailedMessageView (U)* and *DetailedMessageHistoryView (U).*
	SNIPPET_OF_ ANSWERS	Needed to fetch a paginated, filtered and sorted list of answers or messages in *W: DetailedQuestionView (U)* and *W: HistoryView (U)*
C	SNIPPET_OF_ COMMENTS	Needed to fetch a paginated and sorted list of comments in *DetailedAnswerView (U)* and *DetailedMessageView (U)*

Table 5.11 gives an overview of the mutations that are needed to change or update the entities of the backend. The table uses the same structure and categorization as Table 5.10.

Table 5.11: Mutations regarding different entities
Entity legend: Q = Question, A=Answer, C=Comment

Entity	Query	Reason
	Used by REeng	
Q	ADMIN_CREATE_ QUESTION	Needed to create a question in *W: CreateQuestionView (R).*
	ADMIN_UPDATE_ QUESTION	Needed to update a question in *W: CreateQuestionView (R)*
	ADMIN_DELETE_ QUESTION	Needed to delete a question in *W: CreateQuestionView (R)*
	ADMIN_UPDATE_ SCHEDULE_QUESTION	Needed to schedule a question in *W: ScheduleQuestionView (R)*
C	ADMIN_CREATE_ COMMENT	Needed to create a comment in *W: CommentView (R).*
	Used by user	
A	CREATE_ ANSWER	Needed to create an answer in *W: QuestionView (U).*
	CREATE_ANSWER_ SKIPPED	Needed to mark an answer as skipped in *W: QuestionView (U).*
C	CREATE_ COMMENT	Needed to create a comment in *W: CommentView (U).*

[Authentication]

As the feedback of the users involves sensitive data, we need to design our authentication mechanism secure. This involves limiting communication with the backend for users of SMART-AGE and making sure that a user has no access to the feedback of other users.

General decision
We decide to not use a traditional authentication method that includes a combination of username and password, because our users might forget the credentials and could have issues with the process of recovering them. For example, when they forget the username, they would need to communicate with the study personal through emails or through phone. Communication through

email could be a problem for some of the users and our hotline phone is only available at certain days and hours. Because we want to avoid any issues with authentication having impacts on the app usage, we design the authentication so that the users don't even notice it. As each user has a unique user id, we decide that the user id needs to be contained in each request to the backend, to achieve that the backend accepts the request. As the user id might be known by other users, we furthermore decide to encrypt the user id with a key that is safely contained on the device of the users.

This key is known to the backend and it is used by the frontend to encrypt the user id. When a request with an encrypted user id is received by the backend. The backend checks whether the user id can be decrypted successfully. If it can be decrypted, the backend accepts the request.

5.2.2.3 Implementation

In this section, the implementation of the entities of the Java Spring Boot backend and the implementation of the GraphQL queries is explained. Furthermore, we briefly explain how the authentication works and how many lines of code were produced.

[Entites]

For the implementation of the entities in Section 5.2.2.2, we use JPA (Java Persistence API[34]) annotations to define how they relate to other entities and how they are stored in the database.

General decision
We use JPA, because it is already integrated in Java Spring Boot and represents an established API.

The *AbstractEntity* provides a common identifier (id) for each entity. The id attribute is annotated with @Id and @GeneratedValue(strategy = GenerationType.IDENTITY), indicating the it represents the primary for the entity and the its value is automatically generated by the database every time a new entity is persisted. The *User* entity uses the annotation @Temporal(TemporalType.TIMESTAMP) for its attribute createdDate to ensure correct formatting of the date. The *Question* entity uses the @OneToMany(fetch = FetchType.LAZY) annotation on the answers attribute. This represents a one-to-many relationship between *Question* and *Answer*. The FetchType.LAZY part is important for optimizing the performance, as it ensures to load the answers only when needed from the database. For example, the answers are not loaded when only the json attribute of *Question* is needed for calculation. The

[34] https://spring.io/projects/spring-data-jpa

Answer entity uses the @Column(columnDefinition = "TEXT") annotation for the attribute json, because this field may contain more text that is allowed by the default database text type VARCHAR. The @OneToMany(fetch = FetchType.LAZY) annotation on the attribute comments defines again a one-to-many relationship with the entity *Comment* while using lazy loading. The *Comment* entity uses the annotation @ManyToOne for the attribute answer, indicating that multiple comments are associated with a single answer. The *AdminUser* entity does not need any specific JPA annotations for its attributes. The *Snippet<T>* only serves as an entity to wrap a collection of other entities for pagination and it is not saved in the database. All implemented entities along with their attributes and their JPA annotations can be found in the folder *api/src/main/java/de/se/ifi/uniheidelberg/domain/route* folder in the repository[35] on branch dissertation.

To save the entities in the database, we use Hibernate[36], an object-relational mapper (ORM), because the object structure must be mapped to tables.

General decision
We use Hibernate[37] as an ORM, because it is the standard OR mapper of Java Spring Boot and it is an established technology.

Hibernate creates the database structure containing the tables and the columns of the tables along with their types automatically based on the attributes of the entities and their JPA annotations.

[Queries and mutations]

In this Section we show how we implement the queries and mutations of Section 5.2.2.2. For the implementation we use query/mutation resolvers[38] and repositories[39]. We use resolvers to handle incoming queries and mutations. The resolvers map the different queries and mutations to specific methods which perform calculations including fetching and writing data to the database. To fetch and write data from and to the database, repositories are used. Repositories perform CRUD (Create, Read, Update, Delete) on the entities in the database. There exists a repository for each entity. Figure 5.9 shows a flow diagram that represents the steps required to give a response to the frontend which sends a query or mutation.

[35] https://github.com/SMARTAGE21/smartage-feedback-app

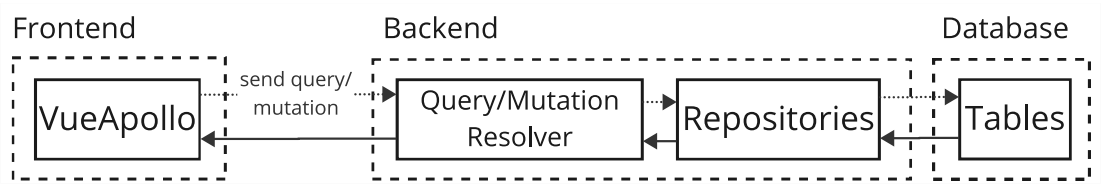
[36] https://hibernate.org/

[37] https://hibernate.org/

[38] https://docs.spring.io/spring-graphql/reference/request-execution.html#execution.graphqlsource.default-type-resolver

[39] https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html

Figure 5.9: Flow diagram for steps necessary to execute queries and mutations. Unnamed dashed arrows represent "accesses" and solid arrows represent data.



The frontend GraphQL client (VueApollo) sends a query or mutation to the query/mutation resolver in the backend. This resolver maps the query or mutation internally to a method which accesses repositories to fetch and write data from and to the database tables. Repositories can automatically write SQL statement for fetching and writing data. They do this by analyzing a declared methods signature. For example, when declaring the method "findByUserId(String id)" in the repository AnswerRepository, the SQL statement "SELECT * from answers where userId = id" is generated automatically. In more complicated cases, the SQL has to be written manually. The resolvers can be found in the folder *api/src/main/java/de/se/ifi/uniheidelberg/domain/route/resolver* folder in the repository[40] on branch dissertation.

[Authentication]

To implement the authentication design described in Section 5.2.2.2, we adjust the resolvers to include a parameter called dataFetchingEnvironment. This parameter contains details about each request that the backend receives, including the requests headers. Inside the headers, the encrypted user id is contained. The encrypted user id is read and decrypted through the file AESHelper. If the decryption is successful the request is allowed to proceed. If the decryption fails, an exception is thrown, blocking the request and preventing unauthorized access. The AESHelper can be found in the folder *api/src/main/java/de/se/ifi/uniheidelberg/domain* folder in the repository on branch dissertation.

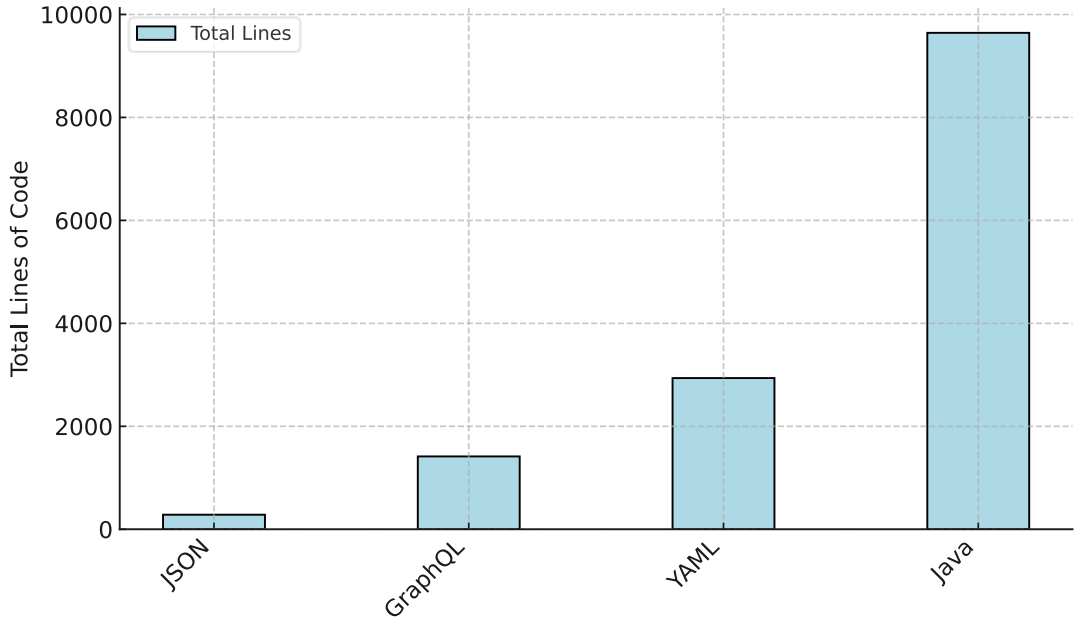
[Lines of Code (LOC)]

Figure 5.10 shows how many LOC we implemented per file type. Next to the java code, we use YAML[41] to store questions in files, GraphQL to define the queries and mutations and JSON for configuration files.

[40] https://github.com/SMARTAGE21/smartage-feedback-app

[41] https://yaml.org/

Figure 5.10: LOC per file type for the backend



5.3 *Quality Assurance*

To assure the quality of our code, we use component tests to test the backend and integration tests to test both the frontend and backend in combination. We describe the component tests in Section 5.3.1 and the integration tests in Section 5.3.2. To ensure the quality of deployment we explain how we use continuous integration (CI) and continuous deployment (CD) in Section 5.3.3.

5.3.1 Component tests

We use component tests that are responsible testing the query and mutation resolvers (explained in Section 5.3.1.1), the repositories (explained in Section 5.3.1.2) and the entities. We don't explain the tests for the entities, because they are very simple, testing only whether they can be initialized correctly and whether their getters and setters work correctly. The tests for the entities can be found in the folder *api/src/test/java/de/se/ifi/uniheidelberg/domain/route* folder in the repository[42] on branch dissertation. We use JUnit[43] as a testing framework. In total, we implemented 160 component tests and reach a coverage of 85% of Java LOC. We use linting to ensure that our code follows the correct syntax and adheres to formatting rules. We can only measure the coverage for the java code, because this code gets executed.

[42] https://github.com/SMARTAGE21/smartage-feedback-app

[43] https://junit.org/junit5/

5.3.1.1 Testing the queries and mutations

We use the abstract base classes BaseMutationResolverTest and BaseQueryResolverTest for establishing a consistent testing environment by setting up necessary dependencies, such as the repositories. Furthermore, with an abstract base class we can setup test data which is used by all tests that inherit from it. We also provide methods for setting the headers of the queries and mutations correctly, as they are only accepted when the headers contain the correctly encrypted user id as explained in Section 5.2.2.3. A typical test is designed to validate a specific query or mutation ensuring that the resolver correctly handles the request and produces the expected outcome. The structure of a test generally contains three steps: preparation, execution and validation. During the preparation, the test is set up with a context (e.g. inserting test data into the database). The execution phase simulates the sending of a query or mutation. In the validation phase, the response of the query or mutation is checked for correctness and for a mutation it is checked whether the data in the database is manipulated correctly. Table 5.12 gives an example for a query test that checks whether a user gets the correct questions on a specific day.

Table 5.12: Example query test that validates whether a user receives the correct questions at a specific day

Phase	Description
Preparation	• Insert Question1 into the database and schedule it to appear on first day and then again on the fifth day • Insert Question2 into database and schedule it to appear on the third day • Insert Question3 into the database and schedule it to appear on day 10 • Create user with id 1 and set its start date to 1.1.1900
Execution	• Send query responsible for getting the questions (name: QUESTIONS see Section 5.2.2.2) for user with id 1 at day 3.1.1900
Validation	• Check whether Question1 and Question2 are contained in the answer of the query

Table 5.13 gives an example for a mutation test that checks whether a user can successfully give an answer to a question.

Table 5.13: Example mutation test that validates whether a user can correctly create an answer to a question

Phase	Description
Preparation	• Insert Question1 into the database • Create user with id 1
Execution	• Send mutation responsible for creating an answer for Question1 (name: CREATE_ANSWER see Section 5.2.2.2) by user with id 1
Validation	• Check whether the response of the mutation does not include an error and check whether the answer of user with id 1 to the question is stored correctly in the database

The tests for the query and mutations resolvers can be found in the folder *api/src/test/java/de/se/ifi/uniheidelberg/domain/route/resolver* folder in the repository[44] on branch dissertation.

5.3.1.2 Testing the repositories

We test each repository through an individual test file. The structure of tests is equal to the tests of the query and mutation resolvers. Table 5.14 shows an example for a test of the repository AnswerRepository.

Table 5.14: Example test for the repository AnswerRepository

Phase	Description
Preparation	• Insert Question1 into the database • Insert Answer1 with creationDate “1.1.1900 12:00” for Question1 into the database • Insert Answer2, Answer3, Answer4 and Answer5 with creationDate “2.1.1900” for Question1 into the database • Insert Answer6 with creationDate “3.1.1900” for Question1 into the database
Execution	• Call method of repository that should return first 5 answers with creationDate between 2.1.1900 and 3.1.1900
Validation	• Check whether the response contains Answer2, Answer3, Answer4, Answer5 and Answer6 in the correct order.

[44] https://github.com/SMARTAGE21/smartage-feedback-app

The tests for the repositories can be found in the folder *api/src/test/java/de/se/ifi/uniheidelberg/domain/route* folder in the repository[45] on branch dissertation.

5.3.2 Integration tests

We use integration tests to test whether the frontend responses correctly to simulated user input. With the integration tests we are testing the complete application consisting out of the frontend, backend and database. We implemented 47 integration tests with the testing framework Cypress[46]. These tests cover 97% of the querys and mutations and 100% of the pages of SF, meaning that every page is visited at least once in a test. We implemented 7140 LOC as Vue code and 3464 LOC as Javascript. The structure of the frontend tests follows those of the backend (preparation, execution and validation). Table 5.15 shows an example for an integration test.

Table 5.15: Example integration test, that checks whether an answer can be submitted

Phase	Description
Preparation	• Insert Question1 into the database that shows for every user from the start • Create user with id 1
Execution	• Go to the page which represents W: QuestionView (U) as user with id 1 • Give inputs for the required subquestions • Submit the answer over the “Submit” button
Validation	• Check whether all questions are answered • Check whether answer appears on “Sent” page and “History” page

The integration tests can be found in the folder *ui/app/src/cypress/integration* folder in the repository[47] on branch dissertation.

5.3.3 Continuous integration and deployment

We use GitHub Actions[48] to automatically run our component and integration tests after each commit on the main branch. GitHub Actions is a feature of GitHub for executing code automatically. For the execution of the tests we create two pipelines, one for the frontend and one for the backend. Every time a file is changed in the frontend or backend and the change is committed, the corresponding pipeline is run.

[45] https://github.com/SMARTAGE21/smartage-feedback-app

[46] https://www.cypress.io/

[47] https://github.com/SMARTAGE21/smartage-feedback-app

[48] https://github.com/features/actions

This ensures that changes to the code are automatically tested. The pipeline executes the tests and if the tests are successful it creates a docker image from the frontend or backend. A Docker image[49] is a package that contains everything needed to run the application, including the code, runtime and libraries. Docker images provide portability and consistency across environments by packaging applications with all their dependencies, ensuring they run the same everywhere. After creating the docker image, it pushes the docker image to DockerHub. DockerHub[50] is a repository where Docker images can be stored and managed. Once the docker image is pushed to Docker Hub, we deploy it to a Docker Swarm[51] ensuring high availability of the application. We monitor the Docker Swarm through automated checks. Whenever an application is not reachable, the check sends an alert to the developer.

[49] https://docs.docker.com/reference/cli/docker/image/

[50] https://hub.docker.com/

[51] https://docs.docker.com/engine/swarm/

PART IV

TREATMENT VALIDATION

Chapter 6

Study context

This Chapter presents the data collection that is necessary to conduct the validation of the approach (see Chapters 7, 8, 9) and the improvement of the approach (see Chapter 10). The data collection is described in Section 6.1. The threats to validity are described in Section 6.1.2.

6.1 Data collection

In Section 6.1.1 we describe the datasets that we use in our analysis. In Section 6.1.2 we describe the conduction of the process to collect feedback and in Section 6.1.3 the conduction of the process to derive requirements.

6.1.1 Datasets

We use four datasets in our analysis. These datasets are listed in Table 6.1. G_{TOTAL} contains the answers of the users to IQ, as well as their messages and usage data. It contains 273 users, as at the time of analyzing the data this number of users has already completed the study. The dataset G_{CODED} contains the answers to IQ and messages from a subset of 64 users, which represents almost a quarter of the total number of users in G_{TOTAL}. From our perspective, this sample size is sufficient for a qualitative analysis while remaining manageable in terms of effort. The answers and messages of G_{CODED} were used to conduct the preparation of the feedback (Section 4.5.3) to enable the derivation of FUQ (Section 4.5.4.1). The dataset G_{FUQ} contains all answers to the FUQ. The dataset G_{FINAL} contains all answers to our final question regarding the satisfaction with the FUQ. All datasets contain only feedback regarding the apps SF and SV.

Table 6.1: Datasets

Measurement	G_{TOTAL}	G_{CODED}	G_{FUQ}	G_{FINAL}
Time range	02.06.23 - 02.09.24	02.06.23 – 28.11.23	22.05.24 – 16.07.24	08.07.24 – 15.07.24

Number of users	273 users who completed study	64 users who completed first 3 months of study	205 users who answered or skipped at least one FUQ	143 users who received the final question
Number of questions	63316 IQ	17283 IQ	8742 FUQ	143 final questions
Number of answers	54643	2943 answers with freetext from 15586 total answers	8251	136
Number of skipped questions	8673	1696	491	7
Messages	622	123	[not included]	[not included]
Usage data	Yes	Yes	No	No

6.1.2 Resulting data of the process to collect feedback

This is the data resulting from the process to collect feedback (see Section 4.4). we received 54643 answers to our 63316 IQ from 273 users who completed the study in the time of 02.06.23 - 02.09.24. We also received 622 messages in that time. The resulting data is stored in G_{TOTAL}.

6.1.3 Resulting data of the process to derive requirements

This is the data resulting from the process to derive requirements (see Section 4.5). The preparation of the feedback based on G_{CODED} resulted in 3002 statements of which were 425 CR. We bundled the CR into 88 topics. We provide the extracted CR, the topics and the total derived FUQ in our repository[52]. Table 6.2 gives an overview of how many FUQ were ask and when.

Table 6.2: Number of FUQ and timepoints

Measurement	**Round 1**		**Round 2**	
Iteration and number of users who received FUQ (*n)*	Iteration 1 (*n*=163)	Iteration 2 (*n*=141)	Iteration 1 (*n*=158)	Iteration 2 (*n*=149)
Time range when FUQ are asked	22.5. - 5.6. (14d)	15.6. – 25.6. (10d)	18.6. – 2.7. (14d)	5.7. – 15.7. (10d)
Number of FUQ	19	11	19	10
Number of FUQ1	4	-	5	-

[52] https://github.com/lradeck/dissertation/blob/main/Change_requests_Topics_FUQ.xlsx

Number of FUQ2	7	3	6	3
Number of FUQ3	8	8	8	7

In the first iteration we asked 19 FUQ (4 FUQ1, 7 FUQ2 and 8 FUQ3). In the second iteration, we asked 3 FUQ2 based on the answers of FUQ1 of the first iteration. This is because one FUQ1 received only one ACR, so we could ask a FUQ3 directly. We also asked 7 further FUQ3 based on the answers of FUQ2 of the first iteration. In the first iteration of the second round we asked another 19 FUQ (5 FUQ1, 6 FUQ2, 8FUQ3). In the second iteration we asked 3 FUQ2, because one FUQ1 did not receive any ACR and another FUQ1 only received one ACR, so we could ask a FUQ3 directly. We also ask 6 further FUQ3 based on the answers to the 6 FUQ2 of the first iteration. The answers to FUQ are stored in G_{FUQ}. In both rounds T_{CHANGE} and T_{ACCEPT} were always reached. It would have been possible to extend each round with one more iteration, to ask FUQ3 based on the answers of the remaining FUQ2, but we wanted to limit the effort of answering more questions for the users in our study. Based on the answers to FUQ3 we were able to derive 6 completely new requirements and 25 changes to existing requirements. Some examples for new requirements and requirement changes are shown in Table 6.3. The complete list of changes and new requirements is shown in Table C.1.

Table 6.3: Examples for derived requirements

App	New requirement or change of existing requirement
SV	New requirement: *SF: customizeStartPage* The user can customize the start page by adding links and applications to it.
SV	New requirement: *SF: filterNews* The user can filter the news by the city “Mannheim” and “Heidelberg”.
SF	New requirement: *SF: editSubmittedAnswer* The user can edit an already submitted answer.
SF	Change of requirement: *W: HistoryView* When displaying the button that allows the navigation to the details of an answer or message, the button should be visible clearly and not be hidden partially.
SF	Change of requirement: SF: *submitAnswer* When asking a question, there should be an answer option “I never did this”, so that the user can signalize that his/her experience is not sufficient to answer the question.
SV	Change of requirement: *W: LinkView* When displaying links, links that lead to apps should be highlighted so that it is clear that they lead to apps and not to web pages.

6.2 Threats to validity

When interpreting the results presented in this study, several threats to validity must be carefully considered. These threats apply for the following Chapters 7, 8, 9 and 10.

Construct validity. The question order, ambiguous wording or leading questions are a threat to construct validity. The sequence in which questions are presented can impact how the users interpret and answer to subsequent ones, possibly leading to biased feedback (Covell et al., 2012). Additionally, ambiguous wording in questions might result in varied interpretations, which could mean the answers don't accurately capture the user perspective. Moreover, the use of leading questions can bias answers towards a particular viewpoint. To alleviate this, we try to not ask questions that are leading and we use consistent wording throughout these questions. Additionally, having too many questions can lead to user fatigue, causing users to hurry through the questions or stop answering altogether. We try to limit this threat by asking only a maximum of five questions per day. Users might also provide answers they believe are expected or "correct" which could mask their true opinions or experiences. To address this, we explicitly tell users during a home visit that all feedback regarding the apps is welcome, whether positive or negative.

Internal validity. The positive relationship between project personnel and the SP, especially during home visits, might influence the answers, leading to more favourable feedback. Also, providing free tablets as incentives could influence the motivation of the users to give feedback. To mitigate potential biases in the answers, we ensure, especially during home visits, conversations remain focused on study-related topics and consciously avoid talking about personal matters.

External validity. The validation and the improvement of the approach (see Chapters 7, 8, 9 and 10) are based on feedback to specific apps (SF and SV), and the findings may not extend to different types of applications or user groups with different characteristics, such as younger individuals or individuals with different technology experience. Furthermore, the exclusion of users who meet specific criteria may limit the generalizability of the findings. This selection bias restricts our ability to apply conclusions to a broader population. Additionally, the lack of visibility of others' feedback and the absence of crowd interactions can limit external validity. The feedback in this isolated environment might not accurately reflect the feedback that users would give in a more interactive setting. Furthermore, we coded only part of the feedback. To mitigate concerns regarding external validity, we drew users from two highly diverse cities (Heidelberg and Mannheim).

Reliability is a concern, particularly with regard to how feedback is processed and analyzed. The extraction of change requests from the user feedback and the mapping to requirements and topics (see Section 4.5.3) involves a degree of interpretation, which could lead to inconsistencies when applied repeatedly. To counteract the threat of reliability we conducted an interrater agreement (IR) for the mapping to classes and

requirements. The IR reached a Kappa of 0.93 indicating very few disagreements. The Kappa value was calculated using Brennan & Prediger Kappa (Brennan and Prediger, 1981). There are many contributing factors for the high Kappa value. One factor was that 95% of the feedback consisted of answers to questions and 90% of the answers were given to questions that addressed a specific requirement. In cases where it was not clear to which requirement the answer statement should be mapped, the coders mapped the statement to the requirement which the question addressed. Another factor was that the feedback was in general very short and rarely needed to be split. Furthermore, the coders profited from a very detailed coding manual. A lot of ambiguities in the coding manual could be avoided, because a third coder conducted a coding on a smaller sample beforehand and the manual was refined based on the results. Both coders were also very familiar with the applications. One coder (the author) lead the development of the applications and the other coder already used the applications and was also familiar with their requirements. The coding was split into five parts, which were smaller in the beginning of the coding and larger at the end. The coders discussed their conflicts after each part to avoid divergence.

Chapter – 7

Validation of feasibility

This Chapter contributes to the knowledge goal 2 of this thesis: *Show that the approach is feasible to collect feedback and to derive requirements.* It validates the feasibility of the treatment by answering the research questions in Section 7.1. Section 7.2 presents the results of the validation and discusses them. Section 7.3 concludes the Chapter and answers the main RQ.

7.1 Research questions

The knowledge goal 2 is refined into the three research questions RQ1, RQ2 and RQ3 in Table 7.1. We ask RQ1 and RQ2 to validate the feasibility of the process to collect feedback and to derive requirements (see Chapter 4) when supported through SF. We ask RQ3, because interacting with SF consistently over an extended period is necessary to follow our process. This Section presents these research questions and the metrics which are used to answer the research questions. We define feasibility regarding the answering of questions so that the percentage of questions which got answered by at least 70% of users must be at least 30%. This is similar to the calculation of response rates for (online) surveys where the numbers of responses is counted, but often distinguishing partial and complete responses (AAPOR, 2004). A complete response in our case means that all questions must be answered. We do not require all questions to be answered, as the questions are asked over a time span and users can have various reasons to sometimes skip a question. We believe to our definition of feasibility is ambitious compared to typical response rates (Shih and Xitao, 2008), but it seems adequate under the condition that users only have to answer single questions.

Table 7.1: Research questions for the feasibility

	RQ	Metric
RQ1	**Is it feasible to collect feedback? (G_{TOTAL})**	
RQ1.1	Is it feasible to collect feedback through IQ?	The percentage of IQ which got answered by >= 70% of users must be >= 30%
RQ1.2	Is it feasible to collect feedback through messages?	The percentage of users that gave a message must be >= 70%

RQ2	**Is it feasible to derive requirements? (G_{TOTAL})**	
RQ2.1	Is it feasible to collect votes through FUQ?	At least 30% of the FUQ must receive votes from at least 70% of users
RQ2.2	Is it feasible to change or create requirements based on the answers to FUQ?	The effort to derive a requirement is less than five minutes per user
RQ3	**How feasible is the usage of SF? (G_{TOTAL})**	The average usage time per week of SF is higher than 10 minutes per week for at least 3 months and the average number of starts of SF per week is higher than 2 for at least 3 months

RQ1 "Is it feasible to collect feedback?" is refined into the RQ1.1 "Is it feasible to collect feedback through IQ?" and RQ1.2 "Is it feasible to collect feedback through messages?". We ask RQ1 and RQ2 because answers and messages represent the feedback that we use in our process. We answer RQ1.1 by assessing whether our definition of feasibility regarding the answering of questions is fulfilled. To answer RQ1.2 we validate whether at least 70% of users gave a message, as this reflects a substantial portion of the users in our view.

RQ2 "Is it feasible to derive requirements?" is refined into the RQ2.1 "Is it feasible to collect votes through FUQ?" and RQ2.2 "Is it feasible to change or create requirements based on the answers to FUQ?". We ask RQ2.1 because the number of votes decides which requirements can be changed or created. We ask RQ2.2 because the effort for changing or creating requirements must be manageable. To answer RQ2.1, similar to RQ1.1, we apply our definition of feasibility regarding the answering of questions. We answer RQ2.2 by checking whether the derivation of either a change to a requirement or the creation of a new requirement takes less than five minutes per user. Compared with analyzing interview results, we think this is reasonable.

RQ3 "How feasible is the usage of SF?" is answered by assessing whether the average usage time per week of SF is higher than 10 minutes per week for at least 3 months and the average number of starts of SF per week is higher than 2 for at least 3 months. We think that using SF for at least 10 minutes a week is the minimum to have enough time to reply to our questions. Furthermore, the users should start SF at least twice per week, to prevent the number of open questions from becoming too high, which could lead to skipping questions due to feeling overwhelmed. The condition regarding usage time and the number of starts should consist for at least 3 months, because we ask the most questions in this time. After 3 months the number of questions we ask declines.

7.2 Results and discussion

In this Section we answer the research questions RQ1, RQ2 and RQ3. We answer RQ1 in Section 7.2.1, RQ2 in Section 7.2.2 and RQ3 in Section 7.2.3.

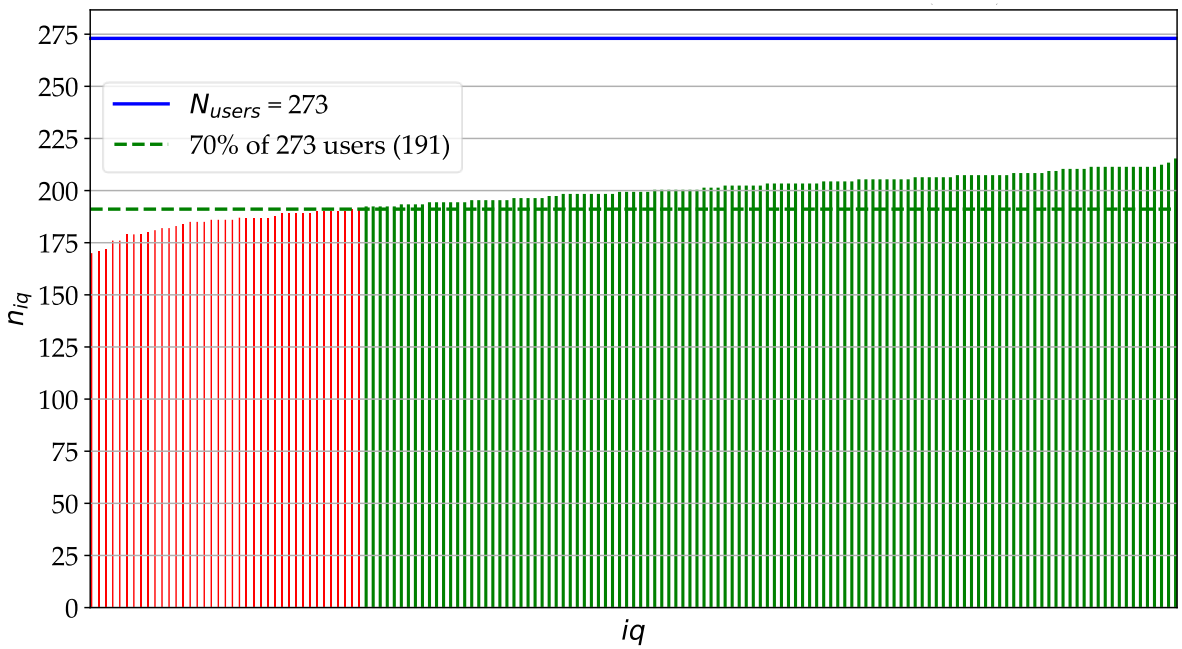
7.2.1 Feasibility of collecting feedback

In this Section we address the RQ1 "Is it feasible to collect feedback?" by answering RQ1.1 "Is it feasible to collect feedback through IQ?" in Section 7.2.1.1 and RQ1.2 "Is it feasible to collect feedback through messages?" in Section 7.2.1.2.

7.2.1.1 Is it feasible to collect feedback through IQ?

In this Section, we present the results for whether the collection of feedback through IQ is feasible. To conduct the validation we use the dataset GTOTAL and we exclude IQ with type *adaptive*, because these IQ are asked only under specific conditions and to only some SP. Furthermore, regarding IQ that get asked repeatedly, we only analyze the first answer for each SP. We do this, because not all IQ have the same number of repetitions. Of the 155 IQ that we analyzed, 116 (74,8%) are answered by more than 70% of users. These IQ are marked green in Figure 7.1. According to our threshold of 30%, this means that it is feasible to collect feedback through IQ.

Figure 7.1: Each IQ (*iq*) and the number of users that answered it (n_{iq})



[Discussion]

The percentage of IQ that are answered by more than 70% of users exceeds the threshold of 30% by far, indicating that users are very motivated to answer a lot of IQ. This could have several reasons. Users may feel a sense of obligation to answer the IQ because they were informed by the study staff that providing feedback is important for the study. Another reason could be that the users want to share their feedback to have impact on the improvement of the apps. It could also be, that the IQ are very understandable and easy to answer, which lowers the barriers to answer them. Furthermore, as the users receive the tablet as a reward after the study, this could create a sense of responsibility to earn the reward by actively participating.

7.2.1.2 Is it feasible to collect feedback through messages?

In this Section, we present the results for whether the collection of messages is feasible. We validate the feasibility of collecting feedback through messages by checking whether the percentage of users who gave messages is at least 70%. We use the dataset G_{TOTAL} and we exclude messages that were given on the day when smartFEEDBACK was introduced to the users by the study personnel. We exclude this day, because the users are instructed to test out sending messages on that day. Only 149 (54,6%) of 273 users gave a message, which is less than 70%. This means that the collection of messages is not feasible.

[Discussion]
Several factors may explain that the collection of messages is not feasible. One possible reason could be that users already feel they have sufficiently provided feedback through answering the IQ. Given the high number of answers regarding the IQ, they may feel that their feedback is already communicated. Furthermore, the perceived effort involved in sending a message could also play a role. While answering IQ might be seen as a quick task, where the goal and structure is already predefined, composing a message might require more cognitive effort.

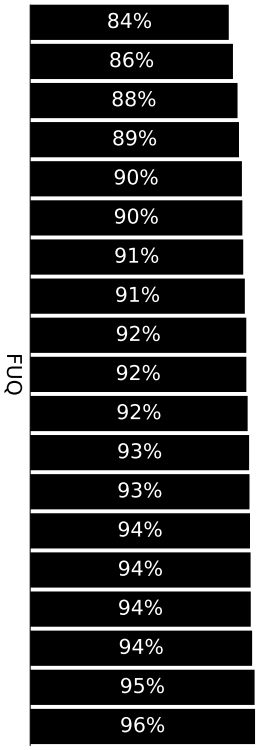
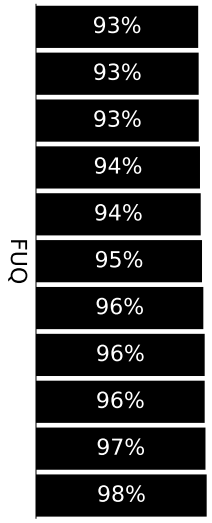
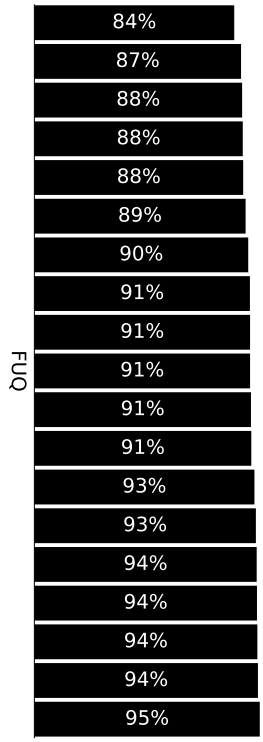
7.2.2 Feasibility of deriving requirements

In this Section we address RQ2 "Is it feasible to derive requirements?" by answering RQ2.1 "Is it feasible to collect votes through FUQ?" in Section 7.2.2.1 and RQ2.2 "Is it feasible to change or create requirements based on the answers to FUQ?" in Section 7.2.2.2.

7.2.2.1 Is it feasible to collect votes through FUQ?

In this Section, we analyze whether the collection of votes through FUQ is feasible. To conduct the validation we use the dataset G_{FUQ}. We define votes as answers to FUQ that represent an opinion regarding a change (e.g. "Yes", "Yes – but I have notes", "I don't care", "No" or a selection of proposed ACR). A FUQ can receive a maximum of one vote. An answer is not counted as a vote, when it doesn't represent an opinion (answering "I find the question not comprehensible" or "I cannot answer the question") or when the question is skipped, which means that no answer was given. All FUQ receive votes from more than 70% of users, which means that it is feasible to collect feedback through FUQ.

Table 7.2: FUQ and the percentage of users who gave votes

Round 1		**Round 2**	
Iteration 1 (*n*=163)	Iteration 2 (*n*=141)	Iteration 1 (*n*=158)	Iteration 2 (*n*=149)
22.5. - 5.6. (14d)	15.6. – 25.6. (10d)	18.6. – 2.7. (14d)	5.7. – 15.7. (10d)
			

[Discussion]

The users already answered our IQ very well, so it is not surprising that the FUQ get a lot of votes as well. Furthermore, as the FUQ ask about specific changes regarding the apps, they can be interesting for the users as they have the feeling of being able to contribute to concrete changes. It is also likely that users are motivated to answer the FUQ because they see that their given feedback is respected and reflected to them.

7.2.2.2 Is it feasible to change or create requirements based on the FUQ?

In this Section, we analyze whether the change or creation of requirements based on the FUQ is feasible. Table 7.3 shows the time effort for the conduction of the process to derive requirements.

Table 7.3: Time effort for one person for the conduction of the process to derive requirements

Step	**Duration**
(1) Mapping to classes and requirements (based on G_{CODED})	32 h
(2) Mapping to topics (based on G_{CODED})	4 h
(3) Derivation of FUQ for first iteration (based on G_{CODED})	9,5 h
Total effort regarding G_{CODED}	45,5 h

Effort per user in G_{CODED} (64 users) per requirement (31 requirements)	~1,4 min
(4) Derivation of FUQ for second iteration (based on G_{FUQ})	5 h
(5) Derivation of requirements (based on G_{FUQ})	1 h
Total effort regarding G_{FUQ}	6 h
Effort per user in G_{FUQ} (205 users) per requirement (31 requirements)	~0,1 min
Total effort:	51,5 h

We analyzed the dataset G_{CODED} including 64 users. We extracted 3170 statements from the feedback of these user that we mapped to classes and requirements. This mapping (1) took around one hour per 100 statements and around 32 hours in total. The 425 statements which were of class "Actionable change request" (ACR) or "Non-actionable change request" (NACR) were then mapped to topics (2). This mapping took around 4 hours. We then derived FUQ1, FUQ2 and FUQ3 from the topics. The derivation of FUQ for the first iteration (3) took around 9,5 hours. Some FUQ (e.g. FUQ3 which require mockups) took more time and others (e.g. FUQ1 without screenshots) required less time. Until this step we used the feedback of the 64 users of G_{CODED}. The following steps use the answers to FUQ (G_{FUQ}) of 205 users. The derivation of FUQ for the second iteration (4) took 5 hours. The derivation of requirements based on the answers of FUQ3 took 1 hour. The time effort for the conduction of the process took 51,5 hours. In total we could derive 31 requirements. The effort per user per requirement in G_{CODED} (see Table 7.3) was around 1,4 minutes and the effort per user per requirement in G_{FUQ} was around 0,1 minutes. When adding up the average efforts we receive an effort of around 1,5 minutes per requirement per user which is less than five minutes and thus feasible according to our threshold.

[Discussion]

Even though the effort for the derivation of changes to existing requirements and the creation of requirements is feasible, we think that it can further be reduced. For example, the use of a classifier that maps feedback to CR and requirements could reduce the effort of 32 hours substantially.

7.2.3 Feasibility of usage of smartFEEDBACK

In this Section, we analyze whether the usage of SF is feasible. We analyze the usage time and the number of starts of the users of the dataset G_{TOTAL} and include only data for the app SF.

[Results]

Figure 7.2 shows the average usage time per week in minutes for SF. The figure shows that the average usage time is consistently over 10 minutes until week 15, where the usage time drops below 10 minutes. Figure 7.3 shows the average number of starts of SF for each week. The figure shows that the average number of starts is consistently over 2 per week for all usage weeks. The usage of SF is feasible, because the average

usage time is above 10 minutes for at least 3 months and also the average number of starts is over 2 per week for at least 3 months.

Figure 7.2: Average usage time per week in minutes for SF (n=273)

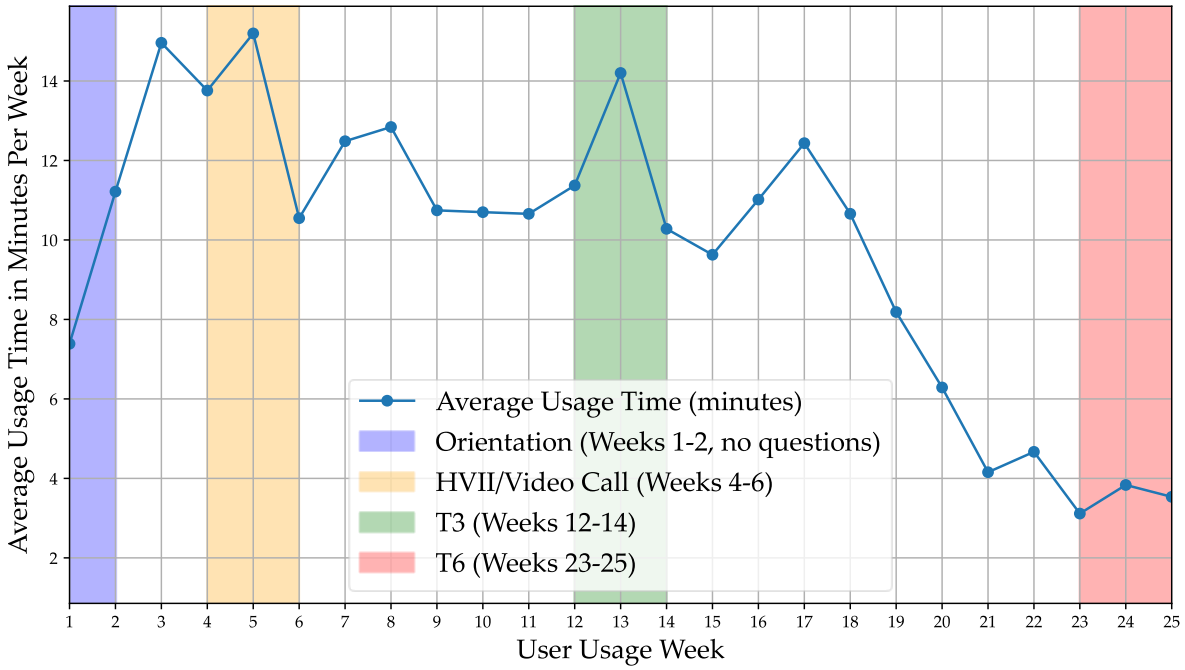
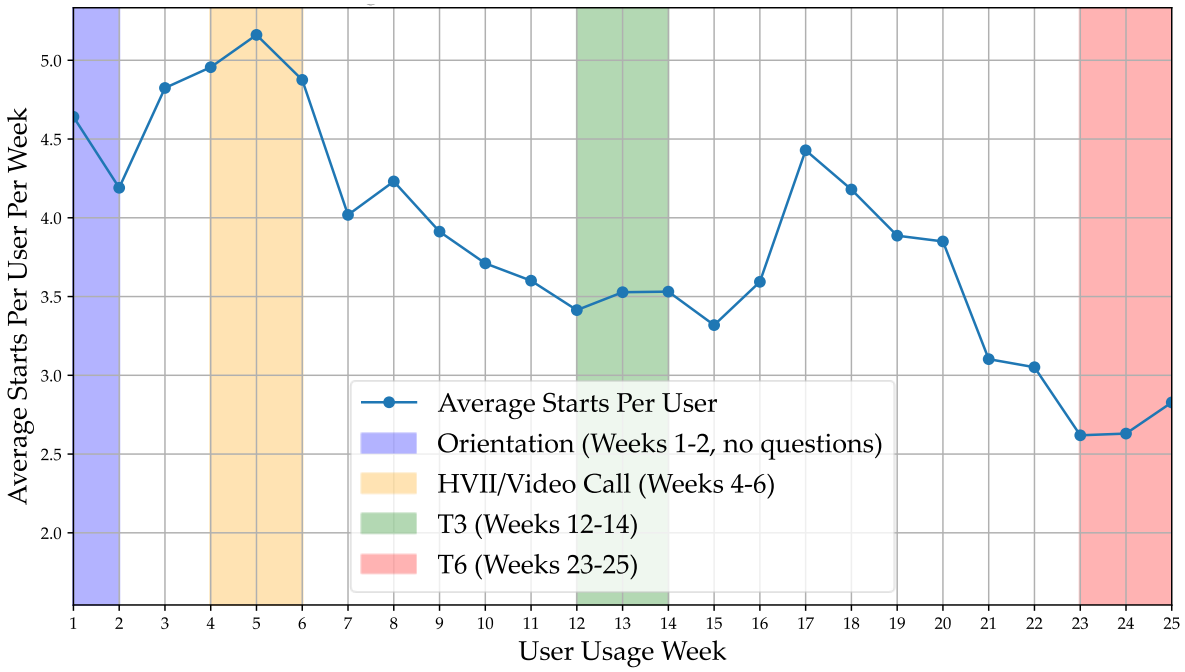


Figure 7.3: Average starts per user per week for SF (n=273)



[Discussion]

The average user usage time remained above the minimum threshold of 10 minutes per week for the first 15 weeks, which exceeds the required 3-month period. We see in Figure 7.2 the usage time peaks several times. The first peak is after five weeks and the time corresponds with the timepoint of the second house visit (HVII) or video call

(depending on the group of the SP). The next peak is after 13 weeks. This time falls into the time of T3 where users get asked a battery of questions through REDCap. The last peak is at week 17, but we don’t know the exact reasons for that. Regarding the average starts of SF in Figure 7.3, we also see peaks at week five and week 17, but no peak at week 13. It is possible that users who hadn't answered questions for a while, or hadn't used the tablet at all, began using the tablet again due to receiving questions through REDCap. This may have led them to open SF and realize that many questions were still unanswered. As a result, they spent more time answering these questions, which could explain the increase in usage time. However, since they didn’t need to start the app frequently, this could account for the lack of a noticeable peak in app starts around week 13.

7.3 Conclusion

This Chapter validates the feasibility of an approach for collecting feedback and deriving requirements, addressing knowledge goal 2 of the thesis. RQ1 assesses whether it is feasible to collect feedback through IQ and messages. We could show that RQ1 is feasible regarding the aspect of collecting feedback via IQ, but not regarding collecting feedback through messages. RQ2 examines the feasibility of collecting votes through FUQ and the feasibility of deriving requirements. We could show that RQ2 is feasible regarding the collection of votes through FUQ and regarding the effort for deriving requirements. RQ3 evaluates the feasibility of the usage of SF. The results indicate that the usage of SF is feasible. We conclude that the approach is feasible to collect feedback through IQ (but not to collect messages) and to derive requirements.

Chapter 8

Validation of effectiveness

This Chapter contributes to the knowledge goal 3 of this thesis: *Show that the approach is effective to collect feedback and to derive requirements.* It validates the effectiveness of the treatment to solve the problems *P1*, *P2* and *P3* by answering the research questions in Section 8.1. Section 8.2 presents the results of the validation and discusses them. Section 8.3 concludes the Chapter and answers the main RQ. The validation of the effectiveness addresses the problems in Section 1.1.

8.1 Research questions

The knowledge goal 3 is refined into three research questions (see Table 8.1). This Section presents these research questions and the metrics which are used to answer the research questions. We ask RQ4 to validate the effectiveness to control the timing of collecting feedback (addressing *P2: Control of timing of feedback collection*). In our approach we ask IQ at specific times, but this does not mean that the IQ are answered timely. We ask RQ5 to validate the effectiveness to collect complete feedback (addressing *P1: Completeness of feedback*). We address RQ5 through three sub RQ. We ask RQ5.1 to validate the effectiveness to collect feedback through IQ (addressing *P1.1 - A lot of feedback can be collected from a lot of users*). This is important because we want to get as many answers from as many users as possible. We ask RQ5.2 to validate effectiveness to map feedback to requirements (addressing *P1.2 - Feedback can be mapped to requirements*). This is important because we want to reduce the effort of mapping feedback to requirements. We ask RQ5.3 to validate the effectiveness to collect change requests (addressing *P1.3 - Feedback contains change requests*). This is important, because our process to derive requirements builds on the change requests. We ask RQ6 to validate the effectiveness to derive requirements (addressing *P3 – Support of change requests among users*). This is important because the change requests must have support among the users so that the derived requirements do not serve only the needs of individuals.

Table 8.1: Research questions for the effectiveness

RQ4	**Is it effective to control the timing of collecting feedback? (GTOTAL)**
	Addressed problem: *P2: Control of timing of feedback collection* Metric: The average time taken for a user to answer a question after the question is asked must be at maximum 3 days for at least 70% of users.
RQ5	**Is it effective to collect complete feedback?**
	Addressed problem: *P1: Completeness of feedback*
RQ5.1	Is it effective to collect feedback through IQ? (GTOTAL)
	Addressed problem: *P1.1 - A lot of feedback can be collected from a lot of users* Metric: The percentage of IQ which got answered by >= 70% of users must be >= 70%
RQ5.2	Is it effective to map feedback to requirements? (GCODED)
	Problem: *P1.2 - Feedback can be mapped to requirements* Metric: The percentage of comprehensible feedback must be at least 70%. Furthermore, less than 30% of the statements included in answers to IQ regarding functional or non-functional requirements need to be remapped to other requirements.
RQ5.3	Is it effective to collect change requests? (GCODED)
	Problem: *P1.3 - Feedback contains change requests* Metric: The ratio of the number of change requests per user must rank among the top three compared to other platforms
RQ6	**Is it effective to derive requirements?** (GCODED, GFUQ)
	Problem: *P3 – Support of change requests among users* Metric: At least 70% of the FUQ must receive votes from at least 70% of users who received the FUQ

RQ4 is answered by calculating the average time taken for a user to answer a question. The time must be at maximum three days for at least 70% of users. We choose three days because we think this should suffice in practice. For repeated IQ we only take into account the first time when the question is answered. This is because when an IQ is repeated for the second time, but the first time it was not answered, then there is only one answer.

We answer RQ5.1 by checking whether the percentage of IQ which got answered by >= 70% of users is at least 70%. We base this on our definition of feasibility in relation to answering questions and increase the percentage of users who respond from 30% to 70% to be even more ambitious.

We answer RQ5.2 by checking whether the percentage of comprehensible feedback is at least 70%. We think that 70% is a substantial part of feedback. Furthermore, as we

ask IQ that address functional and non-functional requirements directly, we check whether less than 30% of the statements resulting from the answers to these IQ need to be remapped to other requirements. We think that when only 30% of statements need to be remapped this represents a substantial reduction of effort.

We answer RQ5.3 by comparing the ratio of the number of change requests per user with those of other platforms identified in our mapping study (see Chapter 3). If our ratio is among the top three of other platforms we consider our collection of change requests effective because this highlights our platform's competitiveness. To compare our platform to others, we need to clarify terminology. Active users are users that interacted with the platform. Regarding SF, they either answered a question or sent a message. Regarding the other platforms, they either posted an idea, added a comment or expressed a vote. We interpret an idea in the context of SF in two ways. First, we interpret an idea as a change request. This is an wider interpretation, because our change requests are usually very short and for example do not contain all aspects necessary to create a user story out of them. Second, we interpret an idea as a validated derived requirement. This is a restrictive interpretation because the other platforms interpret an idea just as a "crowd input" (Wouters et al., 2022) which does not necessarily involve validation. We looked at each of the platforms that we use for our comparison again and checked if it was clear whether the ideas represent ideas which were voted on or not. However, the platforms always only gave the absolute number of ideas and the absolute number of votes. It was thus not clear how many ideas received votes. We use the dataset G_{CODED} for the comparison to the other platforms.

RQ6 is answered by assessing whether at least 70% of FUQ received votes from at least 70% of users who received the FUQ. This is similar to the effectiveness of answering IQ.

8.2 Results and discussion

In this Section we answer the research questions RQ4, RQ5 and RQ6. We answer RQ4 in Section 8.2.1, RQ5 in Section 8.2.2 and RQ6 in Section 8.2.3.

8.2.1 Effectiveness of timeliness

In this Section, we analyze whether the timeliness of collection of feedback through IQ is effective. We validate the feasibility of the timeliness of collection of feedback through IQ by analyzing the time between when an IQ is asked and answered. We use the dataset G_{TOTAL} for our analysis. The average time between receiving an IQ and answering it should be at maximum 3 days for at least 70% of the users for the timeliness of collection of feedback to be effective.

[Results]
The average time in days for all users between receiving an IQ and answering it is 18,1 days. As this time seems very high, we analyzed the distribution of the average time

to answer an IQ for the first time for each user to check for outliers. Additionally, we applied k-means clustering to group users into clusters. Figure 8.1 shows the clusters of users and their average time to answer an IQ for the first time.

Figure 8.1: Clusters of users and their average time to answer an IQ for the first time (n=258, 15 users did not answer IQ)

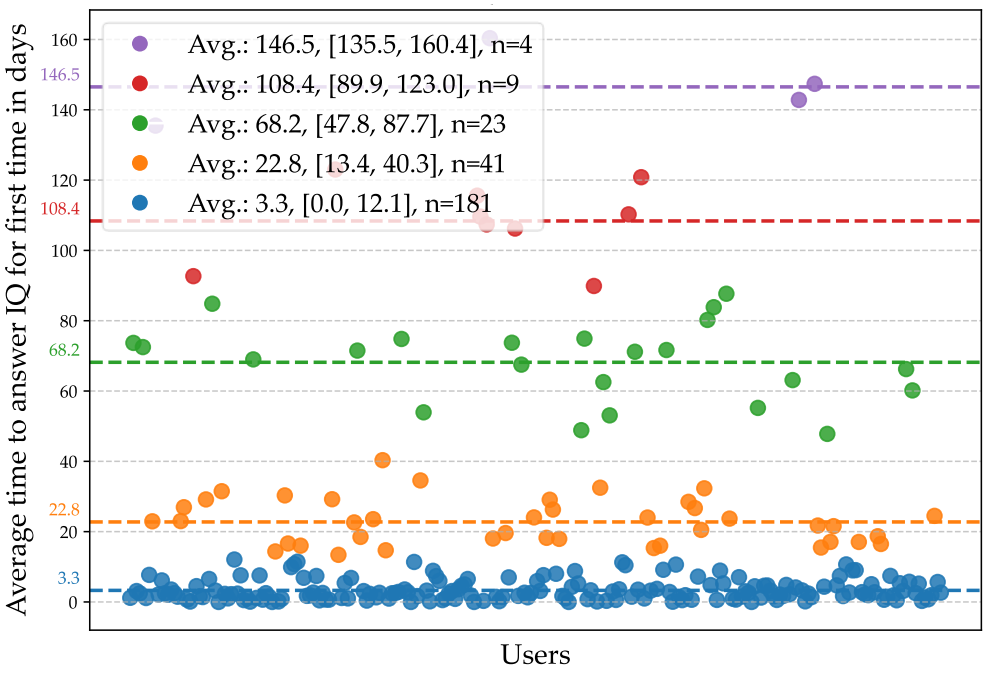


Figure 8.1 shows that there are 5 clusters of users of different size with different average times reaching from 3.3 days to 146.5 days. The biggest cluster with n=181 users has the lowest average time (3.3 days). In total, 258 users answered IQ. This means that 70% of the users who answered IQ, answered them on average 3.3 days after the IQ were received. We graciously see this as effective, because the threshold of 3 days is only exceeded by a few hours.

[Discussion]

While the majority of users falls into the cluster with the lowest average time to answer an IQ, there are smaller groups of users with much longer delays. Some users might only sporadically access SF, which would naturally lead to delays in answering IQ, because we ask IQ regularly. Additionally, some users might view the IQ as irrelevant or repetitive leading to procrastination in answering. Other potential reasons could include technical barriers in the context of interacting with the tablet or SF and thus less motivation in answering the IQ frequently.

8.2.2 Effectiveness of completeness

In this Section we answer the RQ5 “Is it effective to collect complete feedback?” by answering RQ5.1 “Is it effective to collect feedback through IQ?” in Section 8.2.2.1, RQ5.2 “Is it effective to map feedback to requirements?” in Section 8.2.2.2 and RQ5.3 “Is it effective to collect change requests?” in Section 8.2.2.3.

8.2.2.1 Is it effective to collect feedback through IQ?

In this Section we validate the effectiveness to collect feedback through IQ. We validate the effectiveness to collect feedback the same way we validate the feasibility, but with a threshold of 70% instead of 30% regarding the percentage of IQ which got answered by >= 70% of users.

[Results]
As described in Section 7.2.1.1, out of the 155 IQ that we analyzed, 116 (74,8%) are answered by more than 70% of users. According to our condition, this means the collection of IQ is not only feasible, but also effective.

[Discussion]
Several factors could contribute to this high response rate, we discuss these factors in Section 7.2.1.1.

8.2.2.2 Is it effective to map feedback to requirements?

In this Section we validate whether feedback can be effectively mapped to requirements. For this, we check whether the percentage of comprehensible feedback is at least 70%. Furthermore, we check whether less than 30% of the statements resulting from the answers to IQ, that address functional and non-functional requirements directly, need to be remapped to other requirements. We use G_{CODED} for this analysis.

[Results]
98,5% of the answers to IQ are comprehensible. All messages were comprehensible. There were 1849 statements included in the answers to IQ, that address functional and non-functional requirements directly. Only 265 statements (14,3%) needed to be remapped to other requirements. As more than 70% of feedback is comprehensible and furthermore less than 30% of the statements from answers to IQ, that address requirements directly, needed to be remapped, we consider it effective to map feedback to requirements.

[Discussion]
The comprehensibility of the feedback is very high. This could be because the users put effort into their answers and messages or because our questions put the answers into context, which makes it easier to understand them. Furthermore, only a small part (14,3%) of statements resulting from answers to IQ that addressed requirements directly, needed to be remapped. We think we could further reduce this percentage by adjusting our set of IQ. This is because a lot of answers to IQ regarding system functions contained UI details, which lead to remapping the contained statements to a workspace where the system function was included. This could be avoided by asking IQ that address workspaces directly.

8.2.2.3 Is it effective to collect change requests?

We answer RQ5.3 by comparing the ratio of the number of change requests per active user with those of other platforms that described their ratio. If our ratio is among the top three of other platforms, then we consider the collection of change requests effective.

[Results]

Table 8.2 shows the comparison to the other platforms. All users of GCODED were active. According to our wider definition of an idea, we collected 425 ideas, which correspond to the collected change requests and the ratio of ideas per active user is 6.64. According to our restrictive definition of an idea, we collected 39 ideas, which correspond to the validated derived requirements and the ratio is 0.60. To be among the top three, we must reach a ratio of 1.11. With our wider definition of idea, we exceed this threshold and with our restrictive definition of idea we do not meet the threshold. Thus, based on our threshold, we cannot conclude whether the collection of change request is effective or not.

Table 8.2: Comparison of SF to other platforms. US=User story, W=Wider, R=Restrictive

Platform	SF	Tournify	Kmar-Crowd		REfine	GARUSO
Product	SMART-AGE	Tournify	S-Sys	V-Sys	Qubus 7	Smart living
Feedback type	Freetext	US	US	US	Freetext	US
Active users	64	39	60	130	19	32
Ideas	W: 425 (R: 39)	57	32	78	21	56
Ideas/active user	W: 6.64 (R: 0.60)	1.46	0.53	0.60	1.11	1.75

[Discussion]

One has to keep in mind when looking at our comparison, that the definition of an idea is not clearly defined and the platforms in our comparison didn't report how many ideas received votes. We tried to address this by putting up a wider and more restrictive definition of an idea. Additionally, it is not clear whether the platforms did some kind of consolidation, such as removing duplicate ideas, before presenting their idea numbers. We also could not compare our full dataset that we collected in SF, as we only analyzed a part of it qualitatively. However, our dataset is still the second largest compared to the other platforms based on active user count. Furthermore, in our study the users are instructed to give feedback and they receive the tablet as a reward in the end. This could influence the motivation of the users positively.

8.2.3 Effectiveness of requirements derivation

In this Section we present and discuss results regarding the validation of the effectiveness to collect votes through FUQ (RQ6).

[Results]
As described in Section 7.2.2.1, all FUQ received votes by more than 70% of users. According to our condition, this means the collection of votes is not only feasible, but also effective.

[Discussion]
Compared to the IQ, where 74,8% of IQ got answered by more than 70% of users, the FUQ were answered even better, with all FUQ receiving votes from more than 70% of users. This could be due to the fact that the FUQ show the users that we valued their feedback by asking them about specific change requests that they submitted. Furthermore, the use of images and mockups might make the questions more interesting, leading to more answers. We analyze in Chapter 10 whether there is an influence between the characteristics of the FUQ and their vote count.

8.3 Conclusion

This Chapter validates the effectiveness of the approach for collecting feedback and deriving requirements, addressing knowledge goal 3 of the thesis. The effectiveness of timeliness of feedback collection (RQ4) could be validated. Regarding the effectiveness of collecting complete feedback (RQ5), we could validate that it is effective to collect feedback through IQ (RQ5.1) and to map feedback to requirements (RQ5.2). Regarding the effectiveness of the collection of change requests (RQ5.3) we could not clearly validate the effectiveness of the collection of change requests, because we have both a restrictive and wider interpretation of an idea and with the restrictive interpretation, the collection of change requests is not effective. The effectiveness of deriving requirements (RQ6) could be validated. We conclude that the approach is effective in collecting feedback timely and deriving requirements, as well as in collecting complete feedback, provided the wider interpretation of what defines an idea is applied.

Chapter 9

Validation of satisfaction

This Chapter contributes to the knowledge goal 4 of this thesis: *Show that the users are satisfied with the approach.* It validates the satisfaction by answering the research questions in Section 9.1. Section 9.2 presents the results of the validation and discusses them. Section 9.3 concludes the Chapter and answers the main RQ.

9.1 Research questions

The knowledge goal 4 is refined into three research questions (see Table 9.1). This Section presents these research questions and the metrics which are used to answer the research questions. We ask RQ7 to validate the satisfaction of the users with the platform. We address RQ7 by answering the three sub RQ, RQ7.1, RQ7.2 and RQ7.3. We ask RQ7.1 to validate whether the users are satisfied with the platform in general. This is important because if the users are not satisfied with the platform, this could limit the effectiveness of our approach. For the same reason we ask RQ7.2 and RQ7.3. We ask RQ7.2 to validate whether the users are satisfied with the implementation of the functional requirements and RQ7.3 to validate whether the users are satisfied with the implementation of the non-functional requirements. We ask RQ8 to validate whether the users are satisfied with our questions. This is important because the questions are the central part of our process. We ask RQ8.1 to validate whether the users are satisfied with the display of the *IQ*. We analyze the display of the IQ, because we unfortunately did not ask the users about the comprehensibility of the IQ. The question regarding the display of the IQ is the closest question regarding the satisfaction with the IQ that we have. We screened the feedback and the users often refer to the content of the IQ, so this is why we use the question as an alternative. We ask RQ8.2 to validate whether our FUQ are comprehensible. This is important because if the FUQ are not comprehensible, the FUQ will either not be answered or answered in a way that might not represent the intention of the user. We ask RQ9 to validate the satisfaction of the users with the process of asking the FUQ. If the users are not satisfied with the process of asking the FUQ, this could also lead to FUQ being answered less or not meaningfully.

Table 9.1: Research questions for the satisfaction

	Research question	Metric
RQ7	**How satisfied are the users with the platform? (GTOTAL)**	
RQ7.1	How satisfied are the users with the platform in general?	The System Usability Scale Score is at least 70
RQ7.2	How satisfied are the users with the implementation of the functional requirements?	The sum of answers indicating that the implementation of the functional requirement was liked is higher than the sum of answers indicating that the implementation of the requirement was disliked for 70% of requirements
RQ7.3	Are the users satisfied with the implementation of the non-functional requirements?	The sum of answers indicating that the implementation of the non-functional requirement was liked is higher than the sum of answers indicating that the implementation of the requirement was disliked for 70% of requirements
RQ8	**How satisfied are the users with the questions?**	
RQ8.1	How satisfied are the users with the display of the *IQ*? (GTOTAL)	The sum of answers indicating that the display of the IQ was liked is higher than the sum of answers indicating that the IQ were disliked
RQ8.2	Are the FUQ comprehensible? (GFUQ)	At least 70% of the FUQ must be found comprehensible by at least 70% of users who received the FUQ
RQ9	**How satisfied are the users with the with the process of asking FUQ? (GFINAL)**	Q is the final question regarding the satisfaction with the process of asking FUQ and is split into three sub questions: Q1) Satisfaction with the fact that we asked FUQ Q2) Satisfaction with the FUQ Q3) Intention to use For Q1, Q2 and Q3 at least 70% of the users who received the FUQ must choose "Yes" or "Neutral"

RQ7.1 is answered by calculating the System Usability Scale (SUS) Score (Brooke, 1995). A SUS Score of at least 70 is considered good (Sauro and Lewis, 2012). In this case, we assume that the users are satisfied with the platform. We answer RQ7.2 by checking whether the sum of answers indicating that the functional requirement was

liked is higher than the sum of answers indicating that the requirement was disliked. If this is the case for 70% of requirements, we consider that the users are satisfied with the functional requirements. We use the threshold for the feasibility of answering IQ as an orientation. We answer RQ7.3 the same way regarding the non-functional requirements.

RQ8 is refined into the RQ8.1 "How satisfied are the users with the display of the IQ?" and RQ8.2 "Are the FUQ comprehensible?". We answer RQ8.1 by checking whether the sum of answers indicating that the display of the IQ was liked is higher than the sum of answers indicating that the display of the IQ was disliked. If this is the case, we consider that the users are satisfied with the display of the IQ.

With RQ8.2 we analyze whether our FUQ are comprehensible. It is important that the FUQ are comprehensible, because otherwise the validity of the answers to the questions might be compromised. We require for the satisfaction with the FUQ that 70% of FUQ are answered by less than 30% users with "I don't understand the question". This threshold uses the threshold for the effectiveness of answering FUQ as orientation (70% of users must not answer "I don't understand the question").

Regarding RQ9 we ask one question each for three aspects: Q1) Satisfaction with being asked: Did you think it was good that we asked for your feedback on suggestions regarding SV and SF? Q2) Satisfaction with the questions: Did you like the questions we used to ask for your feedback on suggestions regarding SV and SF? and Q3) Intent to answer in the future: Would you also like to answer questions and provide suggestions for other software that you use? We evaluate the satisfaction for each aspect separately.

9.2 Results and discussion

In this Section we answer the research questions RQ7, RQ8 and RQ9. We answer RQ7 in Section 9.2.1, RQ8 in Section 9.2.2 and RQ9 in Section 9.2.3.

9.2.1 Satisfaction with the platform

In this Section we answer the RQ7 "How satisfied are the users with the platform?" by answering RQ7.1 "How satisfied are the users with the platform in general?" in Section 9.2.1.1, RQ7.2 "Are the users satisfied with the functional requirements?" in Section 9.2.1.2 and RQ7.3 "Are the users satisfied with the non-functional requirements?" in Section 9.2.1.3.

9.2.1.1 How satisfied are the users with the platform in general?

In this Section we validate the satisfaction of the users with the platform in general by checking whether the SUS Score of SF is at least 70. We use G_{TOTAL} for our analysis. We only include users who answered all of the SUS questions.

[Results]

133 users answered all of the SUS questions. The average SUS Score is 71.83. This SUS Score is slightly higher than 70, which means that we consider that the users are satisfied with the platform in general.

[Discussion]

The exclusion of users who did not answer all SUS questions could influence the results, as their opinions might differ from those who completed all questions. Additionally, the process of answering all SUS questions requires a certain level of effort and attention from users. This could indicate that the users who completed all SUS questions may represent a subset of users who are more interested in the platform. As the SUS questions consisted of 10 questions, the likelihood is high that one question is skipped. This could be the reason why only 133 users answered all SUS questions.

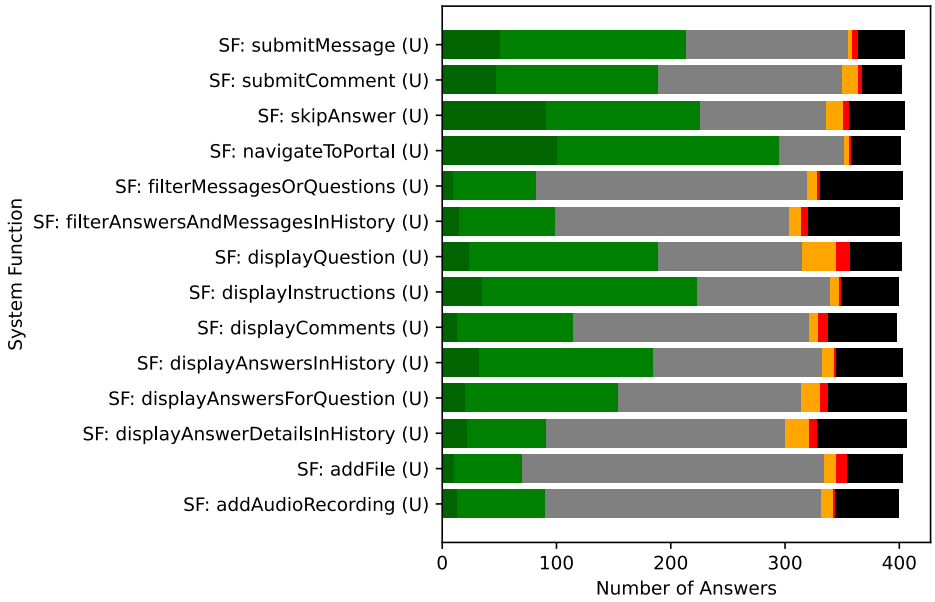
9.2.1.2 How satisfied are the users with the functional requirements?

In this Section we validate the satisfaction of the users with the functional requirements. Our IQ consist of questions asking about opinions regarding the functional requirements of SF. The questions are formulated like this: "How good do you find <functional requirement>?" and the answer options are "Very good", "Good", "Neutral", "Not good", "Not good at all". We use G_{TOTAL} for our analysis.

[Results]

Figure 9.1 shows a diagram which presents the proportions of answers for each IQ regarding a functional requirement. The color dark green represents "Very good", green represents "Good", gray "Neutral", orange "Not good" and red "Not good at all". Black represents the proportion of skipped answers.

Figure 9.1: Answers to IQ which ask about opinions regarding function requirements (n=273)



Based on Figure 9.1 we can see that the sum of answers indicating that the functional requirement was liked (“Very good”, “Good”) is higher than the sum of answers indicating that the requirement was disliked (“Not good”, “Not good at all”) for all functional requirements. Thus, we consider that the users are satisfied with the functional requirements.

[Discussion]

Some system functions received few “Very good” and “Good” answers, for example, SF: addFile (U), SF: addAudioRecording (U) and SF: filterMessagesOrQuestions (U). SF: addFile (U) is a rather advanced system function, requiring more technical experience. For example, it requires the user to search for a file using the tablet file explorer. Similarly, SF: addAudioRecording (U) might present a technical barrier, because it requires starting the voice message, speaking, stopping it and sending it. As for SF: filterMessagesOrQuestions (U), it may be perceived as less essential compared to other functionalities, leading to more neutral answers.

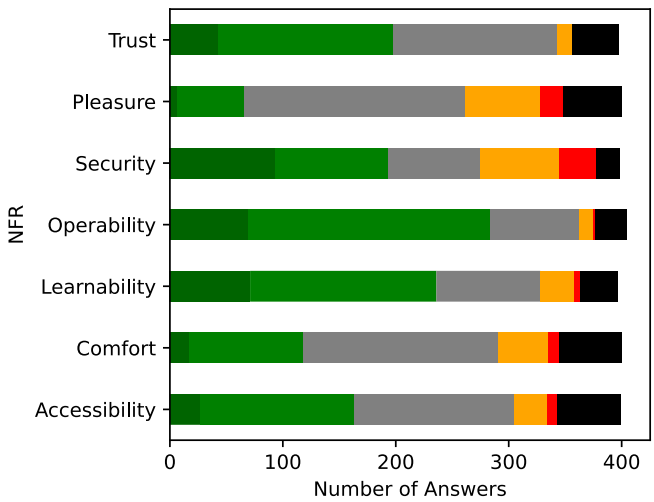
9.2.1.3 How satisfied are the users with the non-functional requirements?

In this Section we validate the satisfaction of the users with the non-functional requirements. We analyze the results to IQ that ask about opinions regarding non-functional requirements and that offer a likert scale (“Very good”, “Good”, “Neutral”, “Not good”, “Not good at all”). We also asked IQ regarding other non-functional requirements (User Error Protection and Effectiveness/Efficiency), but there we asked for freetext instead of a likert scale. For the analysis only the non-functional requirements with a likert scale are relevant. We use GTOTAL for our analysis.

[Results]

Figure 9.2 shows the answers to IQ which ask about opinions regarding non-functional requirements and that offer a likert scale.

Figure 9.2: Answers to IQ which ask about opinions regarding non-functional requirements and provide likert scale (n=273)



Only for the non-functional requirement “Pleasure”, the sum of positive answers is not higher than the sum of negative answers. For 6 of 7 non-functional requirements (more than 70%) the sum of positive answers is higher than the sum of negative answers. Thus, we consider that the users are satisfied with the non-functional requirements.

[Discussion]

The non-functional requirement "Pleasure" did not receive more positive answers than negative ones, indicating that the users did not find the platform as enjoyable as expected. This could be due to fact that we did not include any motivational elements such as gamification in our platform. Furthermore, it could just be that the effort of giving feedback outweighed the pleasure of using the platform. We could not analyze the answers of two non-functional requirements, because there the likert scale was not used. If both non-functional requirements would have more negative than positive answers, than the threshold would not be met.

9.2.2 Satisfaction with the questions

In this Section we answer the RQ8 “How satisfied are the users with the questions?” by answering RQ8.1 “How satisfied are the users with the display of the IQ?” in 9.2.2.1 and RQ8.2 “Are the FUQ comprehensible?” in Section 9.2.2.2.

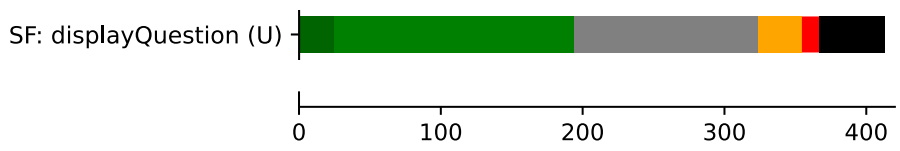
9.2.2.1 How satisfied are the users with the display of the IQ?

In this Section we validate the satisfaction of the users with the display of the IQ. For this we analyze the answers to the IQ “How good do you find the display of our questions?” and the answer options are “Very good”, “Good”, “Neutral”, “Not good”, “Not good at all”. We use GTOTAL for our analysis. In GTOTAL no FUQ were asked to the users. This means when the users receive the IQ “How good do you find the display of our questions?”, they don’t mistake “questions” for FUQ and thus only rate IQ.

[Results]

Figure 9.3 shows the answers to the IQ that asks about how users are satisfied with the display of IQ.

Figure 9.3: Answers to IQ that asks about how users are satisfied with the display of IQ (n=273)



The sum of positive answers is higher than the sum of negative answers. Thus, we consider that the users are satisfied with the display of the IQ. To know what the users did not like about the IQ, we screened the freetext of the answers which represent

"Neutral", "Not good" and "Not good at all". Several users highlighted that the IQ felt repetitive and similar, which they found tiring. Others expressed a desire for larger font sizes to improve readability. Additionally, some users suggested that the IQ could benefit from clearer explanations and a broader range of answer options.

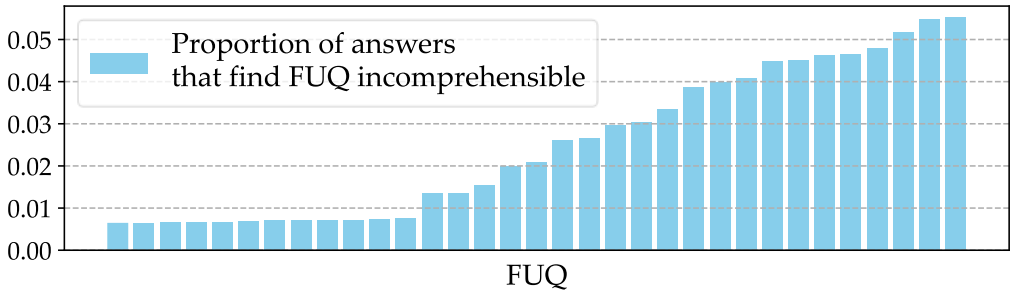
[Discussion]
Even though the IQ received some criticism, this accounted for only a small portion of the answers. The majority of answers was positive. The criticism regarding similarity of the IQ is understandable, as we use very differentiated IQ (e.g. regarding opinions, problems and improvements for a specific aspect). This level of detail might feel repetitive to users as they might respond with problems or improvements to an opinion question initially. When they subsequently receive a problem or improvement question regarding the same aspect then, they could feel like they already mentioned these aspects in a previous answer. Additionally, it is true that we repeat the IQ over the duration of the study.

9.2.2.2 How comprehensible are the FUQ?

In this Section we validate whether the FUQ are comprehensible to the users. For this we analyze the answers to the FUQ of the first iteration of round 1 and 2, because we asked the users there whether they find each FUQ comprehensible or not. We use G_{FUQ} for our analysis. We counted how many users received the FUQ and divided them by the number of the answers for the FUQ that state „I didn't understand the question (or proposal(s))".

[Results]
Figure 9.4 shows a bar chart which represent the proportion of answers to FUQ indicating that FUQ are incomprehensible.

Figure 9.4: Proportion of answers to FUQ indicating that FUQ are incomprehensible (n=205)



For all FUQ the percentage of answers which is "I didn't understand the question (or proposal(s))" is not more than around 5%. This means that according to our threshold the FUQ are comprehensible.

[Discussion]
The fact that a high number of users understand the FUQ suggests the questions along with their descriptions as text and images are not confusing to the users.

9.2.3 Satisfaction with the process of asking FUQ

In this Section we validate whether the users are satisfied with the process of asking FUQ. We use GFINAL for our analysis, which contains the answers to our final question, consisting of the sub-questions Q1, Q2 and Q3 (see Table 9.2).

[Results]

Table 9.2 shows the answers to the questions regarding the satisfaction with the process of asking FUQ. Almost 70% of users agreed to the fact that it was good that we asked for feedback on suggestions regarding SF and SV and 20% were neutral (Q1). 80% of users agreed that they liked the FUQ and 5% said they don't care (Q2). Regarding Q3, in total only 50% of users answered with "Yes" or were neutral. 35% of users do not intend to provide feedback for other software. We conclude that the users are satisfied with our process in the current study, but not motivated for further software.

Table 9.2: Answers to the questions regarding the satisfaction with the process of asking FUQ. A1=Yes, A2=I don't care, A3=No, A4=I find the question not comprehensible, A5=I cannot answer the question. Dataset: GFINAL (n=143)

Q	Question	A1	A2	A3	A4	A5
Q1	Did you think it was good that we asked for your feedback on suggestions regarding SF and SV?	68%	20%	2%	5%	5%
Q2	Did you like the questions we used to ask for your feedback on suggestions regarding SF and SV?	80%	5%	7%	5%	3%
Q3	Would you also like to answer questions and provide suggestions for other software that you use?	34%	16%	35%	12%	3%

[Discussion]
It is understandable that some of our users are not motivated to provide feedback for other software, as they are already heavily involved in giving feedback at the moment of answering this question and they already gave a lot of feedback throughout the study. We therefore believe that a reluctance to give feedback on other software does not imply that they disliked our process or questions, which is supported by the responses to Q1 and Q2.

9.3 Conclusion

This Chapter validates the satisfaction of the users with approach, addressing knowledge goal 4 of this thesis. The analysis confirmed that users are satisfied with

the platform (RQ7), because the System Usability Scale (SUS) score exceeded our threshold. Furthermore, functional and non-functional requirements were positively received. However, non-functional requirements such as “Pleasure” and certain advanced functionalities highlighted opportunities for improvement, suggesting that the platform could benefit from further refinements to enhance user enjoyment. User satisfaction with the questions (RQ8) was validated. The process of asking FUQ (RQ9) could be validated only partially, because users said they do not want to provide feedback for other software in the future. We conclude that the users are satisfied with the approach, except for wanting to give feedback for other software in the future.

Chapter 10

Improvement of the approach

This Chapter contributes to the knowledge goal 5 of this thesis: *Show that the effectiveness of the approach can be improved.* In Section 8 we validated, whether the problems *P1: Completeness of feedback (P1.1: A lot of feedback can be collected from a lot of users, P1.2: Feedback can be mapped to requirements, P1.3: Feedback contains change requests), P2: Control of timing of feedback collection* and *P3: Support of change requests among users* could be solved through our approach. For the problems *P1.1* and *P2* our results exceeded our defined thresholds only slightly. For *P1.3* the results with our restrictive definition of an idea didn't meet the threshold. The other problems could be resolved in such a way that the result was clearly above the defined threshold. We conclude that *P1.1*, *P1.3* and *P2* profit most from improvement. Our goal is to enhance the solving of each problem by identifying variables with a statistically significant impact on the problem and discussing how the approach can be adapted based on this knowledge. We call the variables that we analyze for the solving of the problems "*characteristics*". We describe the characteristics in Section 10.1. We describe the research questions in Section 10.2. We present the results of the research questions along with the discussion in Section 10.3. Section 10.4 concludes the Chapter and answers the main RQ.

10.1 Characteristics

We use three types of characteristics in our analysis. We use characteristics of users that are collected through questionnaires (e.g. age, gender, education) to analyze whether we can improve our approach by adapting better to the attributes of the individual users. We use characteristics of the usage behavior of the users to assess whether we can improve our approach based on insights into how users engage with our platform. Furthermore, we use the characteristics of IQ to improve our approach by assessing whether IQ with specific characteristics are more impactful than others.

Table 10.1 shows the characteristics of users that are collected through questionnaires. The full description of the characteristics can be seen in Table C.2.

Table 10.1: Characteristics of users that are collected through questionnaires

Characteristic	Explanation
u_age	Age
u_gender	Gender
u_abitur	Education (Abitur)
u_swe	Self efficacy (Jerusalem and Schwarzer, 2003)
u_mhdt	Media Use/Frequency of Technology Use Self-designed questionnaire, based on: (Wagner and Zank, 2022)
u_huadi	Frequency and Type of Internet Use Self-designed questionnaire, based on: (Vogel et al., 2020a)
u_mdpq	Mobile Device Proficiency Questionnaire (Roque and Boot, 2018)
u_techbio	Technology biography (Mollenkopf et al., 2000)
u_pus_peu	Perceived Usefulness & Perceived Ease of use Self-designed questionnaire, based on: (Davis, 1985)
u_intc	Intention to (continue) use Self-designed questionnaire, based on: (Bhattacherjee and Premkumar, 2004)
u_peen	Perceived Enjoyment Self-designed questionnaire, based on: (Davis, 1985)

The user usage characteristics are the usage time of SF (*u_usage_time_sf*), the usage time of SV (*u_usage_time_sv*), the total usage time of both apps (*u_usage_time*), the number of starts of SF (*u_number_of_starts_sf*), the number of starts of SV (*u_number_of_starts-_sv*) and the total number of starts of both apps (*u_number_of_starts*). Table C.3 gives more info about the unit of measurement.

Table 10.2 shows the characteristics of IQ that we use for our analysis. We don't use the characteristics *owner*, *purpose* and *type*, because these characteristics are highly correlated to *category* and *aspect*. For example, all questions with *type* "adaptive" have *category* "RI".

Table 10.2: Characteristics of IQ. For examples for IQ see Section 4.4.1.1.

Characteristic	Explanation
q_category_opinion	IQ that asks for opinion
q_category_problem	IQ that asks for problem

q_category_improvement	IQ that asks for improvement
q_category_RI	IQ that asks for reason for inactivity and improvement
q_category_other	This type of IQ can be formulated very freely
q_aspect_system	IQ that addresses app in general
q_aspect_functional	IQ that addresses functional requirement of app
q_aspect_non_functional	IQ that addresses non-functional requirement of app
q_app_smartVERNETZT	IQ regarding SV
q_app_smartFEEDBACK	IQ regarding SF

10.2 Research questions

Table 10.3 shows the research questions and the metrics which are used to answer the research questions. We ask these RQ to more effectively solve the problems *P1.1, P1.3* and *P2*. We ask RQ10 to assess how the effectiveness to collect feedback through IQ can be improved. We ask RQ11 to analyze how the effectiveness to collect CR can be improved. We ask RQ12 to assess how the timeliness of feedback collection can be improved. We do this to be able to collect feedback quickly at desired times.

Table 10.3: Research questions for the effectiveness

RQ10	**How to improve the effectiveness to collect feedback through IQ?**
	Addressed problem: *P1.1: A lot of feedback can be collected from a lot of users* Metric: Identify characteristics that significantly influence the collection of feedback Methodology: Analyze the influence of characteristics on whether an IQ was answered. Dataset: G_{TOTAL}
RQ11	**How to improve the effectiveness to collect change requests (CR)?**
	Addressed problem: P1.3*: Feedback contains change requests* Metric: Identify characteristics that significantly influence the collection of CR Methodology: Analyze the influence of characteristics on whether an answer contains a CR Dataset: G_{CODED}
RQ12	**How to improve the timeliness of feedback collection?**
	Problem: *P2 - Control of timing of feedback collection* Metric: Identify characteristics that significantly influence the timeliness of collection of feedback Methodology: Analyze the influence of characteristics of the users and the characteristics of IQ on the time taken to answer IQ Dataset: G_{TOTAL}

RQ10 "How to improve the effectiveness to collect feedback through IQ?"

We analyze the influence of the characteristics on whether an IQ was answered or not through a multivariate binary logistic regression and a correlation. A multivariate binary logistic regression is a statistical test that looks at multiple independent variables (in our case the characteristics) at the same time, to see how each one influences the likelihood of a dependent binary variable (for this RQ whether or not an IQ was answered) (Sheskin, 2004). For the regression, we use the characteristics of the users that are collected through questionnaires and the characteristics of the IQ as independent variables. As dependent variable we use whether the IQ was answered or not. The results of the regression show us which characteristics influence the dependent variable significantly and to which degree. We examine the influence of user usage characteristics separately, focusing on specific correlations, such as whether users with more usage time of SF also provide more answers to IQ related to SF.

RQ11 "How to improve the effectiveness to collect change requests (CR)?"

To answer RQ11 we also use a multivariate binary logistic regression with the same characteristics as RQ10 with the same reasoning, but for analyzing whether an answer contains a CR or not. We also conduct correlations in the same manner as RQ10, but regarding the submission of CR instead of answers.

RQ12 "How to improve the effectiveness to derive requirements?"

To answer RQ12 we conduct a multivariate linear regression. For the regression we use the same characteristics as RQ10 with the same reasoning, but for analyzing how long an IQ took to answer. Furthermore, we conduct correlations to analyze whether the user usage characteristics correlate with the average time to answer an IQ.

10.3 Results and discussion

In this Section we explain answer the research questions RQ10, RQ11 and RQ12. To understand the results, we describe the statistical terminology first in Section 10.3.1. We answer RQ10 in Section 10.3.2, RQ11 in Section 10.3.3 and RQ12 in Section 10.3.4.

10.3.1 Statistical terminology

To interpret the results of the regressions and the correlations we explain statistical terminology in this Section (Sheskin, 2004). For each regression we describe the values n and p. n is the sample size. p indicates whether overall the set of independent variables explains the independent variable significantly better than having no independent variables at all. We adopt the common threshold of using $p < 0.05$ for significance. For a binary regression we describe *Pseudo* R^2 and for linear regression we describe R^2. *Pseudo* R^2 and R^2 are numbers between 0 and 1 that represent to what extent the independent variables explain the dependent variable. A low number means that there are many other variables that influence the dependent variable. In psychological research it is common that these are below 0.1 (Xu et al., 2022). For each

independent variable we list β, *OR*, *p* and *(OR-1)*100*. β represents the direction and strength of the influence of the independent variable on the dependent variable. *OR* means "Odds Ratio". It is derived from β and explains how the odds (likelihood an event will happen compared to the likelihood that it will not happen) of the independent variable change with each unit increase in the dependent variable. For example, assume that "u_age" is one of our independent variables. If the OR for u_age is 0.97, it means that for every additional year in age, the odds of the independent variable (e.g. the IQ is answered) decreases by about 3%. The 3% represent *(OR-1)*100.* For variables like the aspect of an IQ which don't represent a number but a value, such as "q_aspect_system," "q_aspect_functional," or "q_aspect_non-functional", one value is chosen as "reference value". This reference value serves as the baseline to which we compare the other aspects. For the aspect of an IQ we use "q_aspect_system" as reference value, for the category of an IQ we use "q_category_opinion" and for the app of an IQ we use "q_app_smartFEEDBACK". This means for example that when analyzing which characteristics of an IQ influence whether an IQ is answered, if we find that the *OR* for the characteristic "q_aspect_functional" is 0.52, it means that functional IQ have 48% lower chance of being answered compared to IQ with aspect "system".

For correlations we list *r* and *p*. *r* is the Pearson correlation coefficient. It measures the strength and direction of the linear relationship between two variables, with values ranging from -1 (perfect negative relationship) through 0 (no relationship) to +1 (perfect positive relationship). *r* between 0.1 and 0.3 is considered as a weak relationship, *r* between 0.3 and 0.5 as a moderate and *r* greater than 0.5 as a strong relationship. *p* is used again to show significance.

10.3.2 Improvement of the effectiveness to collect feedback through IQ

Table 10.4 shows the results of the binary logistic regression for RQ10. Our analysis consisted of 60.778 IQ which were either answered or not (*n*). *Pseudo* R^2 is 0.066 and *p* is <0.001, which indicates that overall our results are statistically very unlikely to be due to chance. We identified independent variables that influence significantly whether an IQ was answered. These variables are listed in in Table 10.4. The table shows that there are IQ characteristics and user questionnaire characteristics that influence the chance of the IQ being answered positively or negatively, with a tendency of the IQ characteristics being more influential. When an IQ is of subject *q_aspect_functional*, the chance of receiving an answer is reduced by 48% compared to when the IQ would have *q_aspect_system*. When an IQ has the characteristic *q_category_functional*, the chance of receiving an answer is reduced by 45% compared to when the IQ would have *q_category_opinion*. The chance of an IQ receiving an answer decreases by 30% for each unit the SP's *u_pus_peu* (Perceived Usefulness & Perceived Ease of use of IT) increases. If the IQ has the characteristic *q_aspect_non_functional* the chance of receiving an answer is 27% lower than when it would be with characteristic *q_aspect_system*. The chance for an IQ to be answered by a female (*u_geschlecht* = 1) user is 27% lower than to be answered by a male user

(*u_geschlecht* = 0). The chance for an IQ to receive an answer increases with 26% for each unit of increase *u_peen* (Perceived Enjoyment) of the SP. It also increases by 23% when the IQ addresses SV compared to when it addresses SF. The chance of an IQ getting answered from a user with high school degree (german: "Abitur") is 20% less than without high school degree. For the remaining characteristics and their influence, see Table 10.4.

Table 10.4: Significant results regarding the influence of user questionnaire characteristics and IQ characteristics on whether an IQ was answered. *n*=60778, *Pseudo* R^2=0.066, *p*=<0.001

Independent variable	β	*OR*	*p*	*(OR-1)*100*
q_aspect_functional	-0.65	0.52	<0.001	-48%
q_category_improvement	-0.60	0.55	<0.001	-45%
u_pus_peu	-0.36	0.70	<0.001	-30%
q_aspect_non_functional	-0.32	0.73	<0.001	-27%
u_geschlecht	-0.30	0.74	<0.001	-26%
u_peen	0.23	1.26	<0.001	26%
q_app_smartVERNETZT	0.21	1.23	<0.001	23%
u_abitur	-0.22	0.80	<0.001	-20%
q_category_other	-0.21	0.81	<0.001	-19%
q_category_problem	-0.15	0.86	<0.001	-14%
u_intc	0.11	1.12	<0.001	12%
u_techbio	-0.12	0.88	0.001	-12%
u_age	-0.03	0.97	<0.001	-3%
u_swe	0.02	1.02	<0.001	2%
u_mdpq	0.01	1.01	<0.001	1%

In Table 10.5 we show the results regarding the correlation of user usage characteristics with the number of answers to IQ differentiated by app. The correlations show that there is a significant moderate relationship between the usage time of a user in SF and the number of answers given to IQ that address SF, as well as the total number of answers to IQ. Furthermore, there is a significant weak relationship between the usage time of a user in SV and the number of answers given to IQ that address SV. There is significant a moderate relationship between the number of starts of SF of a user and the number of answers given to IQ that address SF, as well as the total number of answers to IQ. There is a significant moderate relationship between the number of starts of SV of a user and the number of answers given to IQ that address SV.

Table 10.5: Results regarding the correlation of user usage characteristics with the number of answers to IQ.

Variable pair	r	p
u_usage_time_sf Number of answers to IQ of SF	0.57	<0.001
u_usage_time_sv Number of answers to IQ of SV	0.26	<0.001
u_usage_time_sf Number of total answers	0.57	<0.001
u_number_of_starts_sf Number of answers to IQ of SF	0.51	<0.001
u_number_of_starts_sv Number of answers to IQ of SV	0.32	<0.001
u_number_of_starts_sf Number of total answers	0.51	<0.001

[Discussion]

The low *Pseudo* R^2 of the binary logistic regression indicates that there are many other unmeasured variables that influence the whether a user gives an answer to IQ or not. IQ that address functional or non-functional aspects were less likely to be answered than IQ that address the system as a whole. This may be because more open IQ are easier to answer. For instance, IQ about specific functions might remain unanswered due to lack of usage of the functions or the inability to understand which function in the app is meant. IQ regarding non-functional requirements might be conceived as too irrelevant (e.g. asking for the comfort, pleasure or accessibility when using the app – see Figure 9.2 RQ7.3). IQ that ask for improvements were less likely to be answered compared to those that ask for opinions. This might be because we ask IQ regarding improvements always after IQ regarding opinions and problems. When the user answered the IQ regarding their opinions and problems, they often already submitted everything that they wanted to say and they skip the IQ regarding improvement. We also recognize that user with a higher score regarding perceived usefulness and perceived ease of use regarding technology (*u_pus_peu*) and users with a more technological experience (*u_techbio*) are less likely to answer IQ. This could be because these users are less interested in using the SMART-AGE apps overall, as these are more focused on users with less technological experience. Users with a high score regarding their perceived enjoyment with technology (*u_peen*) are more likely to answer the IQ. This might be because these users like using the tablet or the apps and thus use it more often. Furthermore, older users and users with high school degree are less likely to answer IQ. Older users might be less interested in using SF to give feedback and users with high school degree could be more selective in answering the questions.

The correlation of user usage characteristics with their number of answers to IQ shows weak to moderate but statistically significant relationships. Higher usage time or number of starts of a user regarding an app correlates with an increased amount of answered IQ from that user regarding that app (for SF more than for SV). Also, the usage time and the number of starts of SF by a user correlates with the total number of answers given by the user. These results were expected, as for answering IQ, the app SF must be used, which increases the usage time and the number of starts. Furthermore, answers regarding SV cannot be answered without experience in using SV.

[Ideas for improving the effectiveness to collect feedback through IQ]
To increase the likelihood of an IQ to be answered that addresses functional aspects, we could ask that IQ only when it is clear based on the monitoring data that the user has used the function already. This reduces the chance that the user doesn't answer the IQ, because of a lack of experience. Furthermore, to reduce the likelihood that the user doesn't answer the IQ because of inability to understand which function in the app is meant, we could provide explanations and screenshots for the function. To make use of the insight that user with high perceived enjoyment of technology are more likely to answer IQ, we could try to increase the enjoyment of SF by experimenting with motivational elements such as gamification, even though we initially decided against gamification during the design of SF because of lacking evidence in literature. Regarding the insight that more usage time and more starts of the apps increase the likelihood to answer IQ, we could make more use of notifications in SF. For example, we could send the users a notification each day when new IQ can be answered. Currently, we do this only when the number of unanswered IQ is high. However, we did not analyze whether this influences the usage of SF, so this is just an assumption that is yet to be validated.

10.3.3 Improvement of the effectiveness to collect CR

The results in Table 10.6 show the influence of user questionnaire characteristics and IQ characteristics on whether an IQ was answered and a CR is contained in the answer. The binary logistic regression, performed on 16510 IQ (n), yielding a *Pseudo* R^2 of 0.080 and remaining statistically significant (p<0.05). The table shows that there are IQ characteristics and user questionnaire characteristics that influence the chance of the IQ answer containing a CR positively or negatively, with a tendency of the IQ characteristics being more influential. The category of an IQ can be identified as a key predictor for whether the IQ was answered whilst also containing a CR. If the category is *q_category_RI* or *q_category_improvement*, than the likelihood of the IQ being answered with a CR is 80% or 66% higher than the reference *q_category_opinion*. Furthermore, when a user has a high school degree, the likelihood of an IQ being answered with a CR is 74% higher. If the category of the IQ is *q_category_other* or *q_category_problem* than the likelihood of an IQ receiving an answer with a CR is 66% or 60% lower than the reference *q_category_opinion*. The likelihood of a user answering

an IQ with a CR increases by 55% for each unit of *u_intc* (Intention to (continue) use IT). Furthermore, if the IQ addresses SV instead of SF, the likelihood is 40% lower to receive a CR. Also, when the user is female, the likelihood of answering an IQ with a CR is 37% higher. For the remaining characteristics and their influence, see Table 10.6.

Table 10.6: Significant results regarding the influence of the user questionnaire characteristics and IQ characteristics on whether an IQ was answered and contained a CR.
(n=16510, *Pseudo* R^2=0,080, p=<0.001)

Input variable	β	*OR*	*p*	*(OR-1)*100*
q_category_RI	0.59	1.80	0.01	80%
u_sozd_schule	0.55	1.74	<0.001	74%
q_category_improvement	0.51	1.66	<0.001	66%
q_category_other	-1.09	0.34	<0.001	-66%
q_category_problem	-0.92	0.40	<0.001	-60%
u_intc	0.44	1.55	<0.001	55%
q_app_smartVERNETZT	-0.52	0.60	<0.001	-40%
u_geschlecht	0.31	1.37	0.02	37%
u_peen	-0.27	0.77	0.001	-23%
u_mdpq	0.02	1.02	<0.001	2%

In Table 10.7 we correlate the user usage characteristics with the number of change requests differentiated by app. There is a moderate relationship between the usage time of a user regarding SF and the number of CR regarding SF from that SP, as well as the total number of CR from that SP. Furthermore, the number of starts of SF of a user correlates with the number of CR for SF from that SP, as well as with the total number of CR from that SP. There is no significant correlation between the usage time/number of starts of SV and the number of CR for SV.

Table 10.7: Results regarding the correlation of user usage characteristics with the number of CR

Variable Pair	r	*p*
u_usage_time_sf Number of CR for SF	0.33	<0.001
u_usage_time_sf Total number of CR	0.33	<0.001
u_number_of_starts_sf Number of CR for SF	0.22	<0.001
u_number_of_starts_sf Total number of CR	0.20	0.001

[Discussion]
IQ that ask for improvement (*q_category_RI* and *q_category_improvement*) increase the likelihood of receiving a CR on an IQ the most compared to IQ with *q_category_opinion*. This makes sense, as this is the goal of these IQ. Interestingly, IQ with *q_category_RI* (adaptive IQ) are even better than *q_category_improvement* for receiving CR. This could be due to the fact that the adaptive IQ ask for the reason for inactivity and users might feel the need to justify themselves by giving a detailed answer. IQ with *q_category_problem* decrease the likelihood of receiving a CR through an IQ compared to IQ with *q_category_opinion*. This suggests that while users might mention issues, these issues represent problems with the existing functionality and do not represent our definition of change requests which address the change of requirements. Users with a high school degree are more likely to provide CR in their answers. This could be because they answer the IQ in more detail than other SP. Furthermore, users with higher u_*intc* (Intention to (continue) use IT) are more likely to give CR. This might be, because these users are interested in continually using IT in the future and are thus more motivated to contribute to its improvement.

Regarding the correlation of user usage characteristics with the number of CR, the usage characteristics for SV show only weak and non-significant correlations for the number of CR related to SV, in contrast to SF, where more usage or more starts lead to more CR. It makes sense that usage and starts of SF correlates with the number of CR, because usage and starts of SF are needed to give CR (a CR cannot be submitted without using SF). This is not the case for SV, as users could submit a lot of CR regarding SV without using SV a lot.

[Ideas for improving the effectiveness to collect CR]
To increase the likelihood of an IQ to be answered with a CR, the logical conclusion based on our results would be to ask more IQ that ask for improvement. Specifically, we could ask more adaptive IQ that ask for improvements regarding used functionalities. At least, the order of IQ should be changed, so that IQ addressing improvement are not always asked after IQ addressing opinion and problem. As users without high school degree tend to give less CR, we could try to formulate our IQ in easier language or with more explanations.

10.3.4 Improvement of the effectiveness to collect feedback timely

We analyzed the influence of questionnaire characteristics of the users and the characteristics of IQ on the time taken to answer IQ through a linear regression. The regression did not yield any significant results. We further calculated a correlation between the user usage characteristics and the average time difference to answer IQ differentiated by app. The results are shown in Table 10.8. The usage time of SF did not correlate significantly with the average time difference to answer IQ neither for both SF and SV nor for SF and SF individually. The usage time of SV did also not correlate significantly with the average time difference to answer IQ for SV. The

number of starts of SF showed an inverse moderate significant correlation with the average time difference to answer IQ (regarding SF), meaning that the more SF is started, the less the average time difference of answering IQ (regarding SF) is. The number of starts of SV showed an inverse, but weak significant correlation on the average time difference to answer IQ regarding SV.

Table 10.8: Significant results regarding the correlation of user usage characteristics average time difference to answer IQ

Variable Pair	r	*p*
number_of_starts_sf average time taken to answer IQ	-0.35	<0.001
number_of_starts_sf average time taken to answer IQ regarding SF	-0.35	<0.001
number_of_starts_sv average time taken to answer IQ regarding SV	-0.25	0.001

[Discussion]
The linear regression did not yield any significant predictors. The correlation however identified the number of starts of SF as a significant predictor on both the average time taken to answer IQ and the average time taken to answer IQ regarding SF. This makes sense, as for timely answering IQ it is necessary to start SF regularly. The correlation of the number of starts of SV on the average time taken to answer IQ could be due to the fact that when users answer IQ regarding SV, they also start SV for example to try out a functionality that an IQ addresses.

[Ideas for improving the effectiveness to collect feedback timely]
One idea for improving the effectiveness to collect feedback timely is to integrate the possibility to answer IQ regarding SV in SV. This way when users start SV, but not SF, they can still answer our IQ regarding SV leading to a reduced time taken between the IQ overall. Another way would be to integrate a linking functionality from SV to SF so that users are reminded of SF more and the transition from one app to the other is facilitated.

10.4 Conclusion

This Chapter validates that the effectiveness of the approach can be improved, addressing knowledge goal 5 of this thesis. The analysis identified significant predictors that are relevant for improving the solving of the problems *P1.1: A lot of feedback can be collected from a lot of users*, *P1.3 Feedback contains change requests* and *P2*: *Control of timing of feedback collection*.

With RQ10 we identified significant predictors for the collection of answers to IQ (addressing *P1.1*). The aspect and category of an IQ were the most influential factors in determining whether it will be answered or not. IQ addressing functional or non-functional aspects were less likely to receive answers than those focusing on the system as a whole. Similarly, IQ asking for improvements were less likely to be answered compared to those asking about opinions. Also, all users usage time and number of starts were predictors for the amount of answers to IQ independently of the app. Our ideas to improve the effectiveness of answer collection included to ask IQ addressing functional aspects only when the functional aspect was already used by the SP, avoiding IQ that cannot be answered due to a lack of experience. Furthermore, ideas included to give clearer reference to functionality in IQ, as well as using notifications more intensely to drive app usage.

With RQ11 we identified significant predictors for the collection of answers to IQ that contain CR (addressing *P1.3*). The category of an IQ and the education of the user were the most important predictors. When the category of the IQ included asking for improvement or when the user had a high school degree, the chance of receiving an answer including a CR increased strongly. Additionally, a longer usage time of SF and a higher number of starts of SF contributed to a higher number of collected CR. Our ideas to improve the effectiveness of collecting CR included to ask more IQ that ask for improvements or to change ask IQ addressing improvements before other IQ. Furthermore, to make the IQ easier to understand for users without high school degree, we propose to formulate the IQ in easier language and to add explanations.

With RQ12 we identified significant predictors for the timely collection of answers to IQ (addressing *P2*). The only significant predictors on the time taken to answer IQ that we could identify were the number of starts of SF and the number of starts of SV. These correlated inversely, meaning that higher number of starts lead to less average time taken to answer IQ. Our ideas for improving the effectiveness to collect feedback more timely include the integration of the possibility to answer IQ regarding SV in SV or the integration of a linking functionality facilitating the switch to SF.

We conclude that we could identify predictors regarding the problems *P1.1*, *P1.3* and *P2* through which we could come up with ideas to improve the effectiveness of the approach.

PART V

CONCLUSION AND OUTLOOK

Chapter – 11

Conclusion

This Chapter summarizes the goals of this thesis and its contributions. The contributions are valuable for practitioners that collect feedback and derive requirements, as well as for researchers. The goals of this thesis consisted of one technical research goal and five knowledge goals.

Knowledge goal 1 was to understand the current state and practice of collecting feedback over platforms (Part II). We conducted a systematic mapping study and found that feedback is collected on platforms using either free text or templates. The feedback is collected in single or multiple phases, either in a governmental, research or commercial setting. Evaluations primarily focus on platform acceptance and user participation, while the evaluation of the content of feedback is rare. Platforms typically support submitting, commenting, and voting on feedback, with additional features like gamification. The results of our mapping study are valuable for researchers as a basis for future systematic mapping studies and for an orientation regarding the design of individual approaches.

The technical research goal was to develop the treatment for three identified problems reported in literature that occur during collection of feedback and the derivation of requirements: *P1) Completeness of feedback*, *P2) Control of timing of feedback collection* and *P3) Support of change requests among users*. The treatment, also called approach, consists of the process to collect feedback and derive requirements and the platform that supports the process (Part III). *P1) Completeness of feedback* is treated by asking a variety of IQ to every user (addressing *P1.1: A lot of feedback can be collected from a lot of users*), including requirements specific IQ (addressing *P1.2: Feedback can be mapped to requirements*) and IQ asking for improvements (addressing *P1.3: Feedback contains change requests*). *P2) Control of timing of feedback collection* is treated by asking these IQ at either fixed timepoints and *P3) Support of change requests among users* is treated by asking FUQ to validate and refine change requests among the users. Adopting our approach is particularly interesting for practitioners who want to collect feedback and derive requirements regarding their products. This is because the approach not only addresses the identified problems but it is also highly customizable regarding the

configuration of questions and it is easily set up, due to the simplicity of deploying SF as a container.

Knowledge goal 2 was to show that the approach is feasible to collect feedback and to derive requirements (Chapter 7). The results indicate that collecting feedback through IQ and collecting votes through FUQ was feasible, while collecting messages was not. Requirements could be successfully derived with manageable effort per user and the usage of SF proved feasible, with sustained activity over three months. The results provide other researchers with valuable benchmarks for assessing the feasibility of their approach to collect feedback collection and derive requirements.

Knowledge goal 3 was to show that the approach is effective to collect feedback and to derive requirements by analyzing whether the problems could be solved (Chapter 8). The results show that sufficiently complete feedback (addressing *P1*) could be collected, because the amount of feedback collected exceeded our threshold (addressing *P1.1*), as well as because only a small part of the feedback needed manual remapping to other requirements (addressing *P1.2*) and because our collection of change requests exceeded those of other platforms when we interpreting the results with a wider definition of an idea (addressing *P1.3*). Furthermore, the feedback could be collected timely (addressing *P2*) and the support of change request among the users could be validated, as all FUQ received more votes than our defined threshold (addressing *P3*). The results are valuable for practitioners, as *P1*, *P2*, and *P3* are industry-relevant problems. However, it should be noted that these findings were obtained in a study setting with the specific target group of older adults. This means that additional testing and adjustments may be necessary to ensure the problems can also be solved effectively in other settings with different age groups.

Knowledge goal 4 was to show that the users are satisfied with the approach (Chapter 9). The users were satisfied with the platform itself, indicated by a high System Usability Score (SUS). The satisfaction with the implementation of the functional and non-functional requirements could also be validated, even though the implementation of the requirements could benefit from further improvements. The satisfaction with the presentation of the IQ and with the comprehensibility of the FUQ could also be validated. The process of asking FUQ could be validated partially, because even though users appreciated being asked for feedback and despite that they liked the FUQ, they indicated that they were not motivated to give feedback for other software.

Knowledge goal 5 was to show that the effectiveness of the approach can be improved (Chapter 10). Our goal was to increase the effectiveness in solving the problems *P1.1 A lot of feedback can be collected from a lot of users*, *P1.3 Feedback contains change requests* and *P2 Control of timing of feedback collection*. Regarding *P1.1* we found out that IQ addressing functional or non-functional aspects were less likely to receive answers than those focusing on the system as a whole and that IQ asking for improvements were less likely to be answered compared to those asking about opinions. To increase the likelihood of an IQ to be answered that addresses functional aspects, we could ask that IQ only when it is clear based on the monitoring data that the user has used the

function already. This reduces the chance that the user doesn't answer the IQ, because of a lack of experience. Furthermore, to reduce the likelihood that the user doesn't answer the IQ because of inability to understand which function in the app is meant, we could provide explanations and screenshots for the function. We also identified that app usage time and number of starts influenced the collection of feedback significantly. This means that reminding the users more through notifications to use SF could increase the amount of collected feedback. Regarding *P1.3* we identified that IQ that ask for improvement, especially adaptive IQ, were more likely to yield CR. So even though these IQ are less likely to be answered in general, when they are answered, they yield more CR. Also, users with high school degree were more likely to answer IQ with CR. To improve CR collection, we suggest to use more IQ to ask for improvements or to ask those IQ before other IQ. Also, we suggest to formulate IQ in easier language or to add explanations for users without high school degree. Regarding *P2* the number of starts of SF and SV were significant predictors for timely IQ answers, with more starts reducing the time taken to answer the IQ. Our proposed improvements include allowing to answer IQ within SV and enable users to navigate from SV to SF easily.

In summary the dissertation contributed an approach that enables researchers and practitioners to collect feedback and derive requirements without facing the problems *P1*, *P2* and *P3*. Additionally, we contribute the mapping study that can serve as a foundation for future systematic mapping studies or as a guide for designing individual feedback platforms. We also offer a dataset of change requests collected in SMART-AGE, providing insights into real-world feedback from older adults. This dataset is useful for researchers and practitioners aiming to understand the specific needs and preferences of this user group. Lastly, we contribute a validation of the approach's feasibility, effectiveness, user satisfaction and improvement, offering a benchmark for researchers to compare their approaches.

Chapter 12

Outlook

This Chapter explores future work, focusing on leveraging advancements in large language models (LLMs), to automate and enhance our approach. We also discuss prior actions worth considering before implementing our approach in an industrial setting. We believe that LLMs can replace a large part of the manual steps required to conduct our process to derive requirements. The derivation of requirements can be supported by LLMs because the extraction of CR and their mapping to topics and requirements is a classification problem. However, the derivation of FUQ, especially the creation of mockups will likely still require human assistance, because at the time of writing this thesis, images often still contain hallucinated text. Nevertheless, we believe that automatic mockup creation will be possible in the future as well. Assuming that the steps of our approach can be fully automated, changes to existing requirements or new requirements that are validated by the users could be proposed continuously to the requirements engineers without any manual effort involved. It is even imaginable, that based on the proposed requirements, automatic code updates are triggered and feedback is collected automatically for the new software version again through A/B testing. Independently of the support through LLMs, our approach should be improved based on the validated requirements that we derived for SF from the users and the identified improvements regarding knowledge goal 5. Before applying our approach in an industrial setting, we believe it is important to conduct a more thorough assessment of potential threats to external validity. Specifically, we think evaluating whether the approach is effective with a younger target group and in a context where users do not receive a reward for their participation. Furthermore, in industry feedback already exists from the collection through various channels like emails or social media. This means that it would also be possible to skip the process to collect feedback and only apply our process to derive requirements. To make use of the various existing feedback channels, it is also imaginable, to adjust SF for collecting feedback and asking questions through channels that allow interactions with the users (e.g. Email or Slack) directly instead of through the UI of SF.

PART VI

APPENDIX

A Supplementary Material for the Problem Investigation

A.1 Methodology

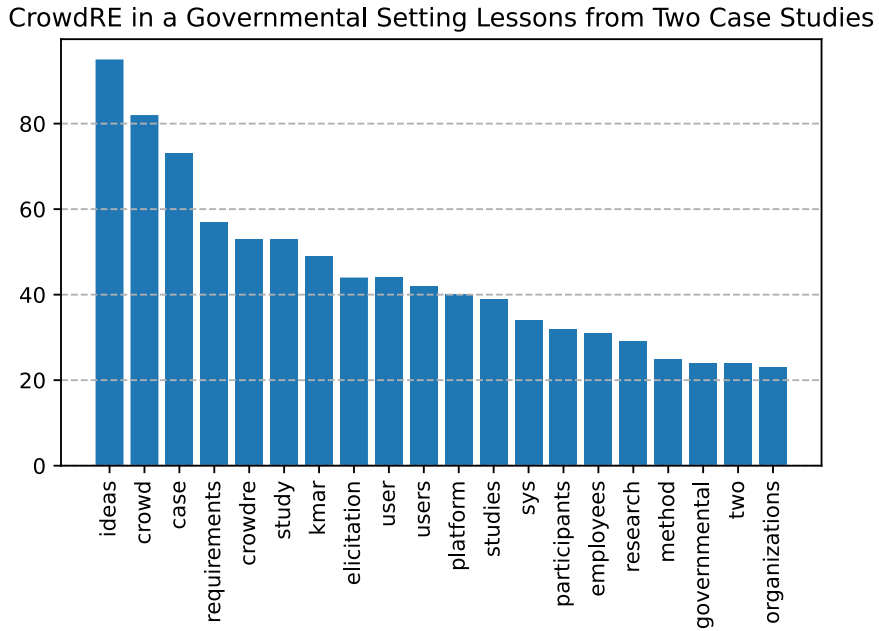
Table A.1.1: Known relevant articles

Ref.	Title	Platform
(Wouters et al., 2021)	CrowdRE in a Governmental Setting: Lessons from Two Case Studies	KMar-Crowd
(Kolpondinos and Glinz, 2020)	GARUSO: a gamification approach for involving stakeholders outside organizational reach in requirements engineering	GARUSO
(Menkveld et al., 2019)	User story writing in crowd requirements engineering: The case of a web application for sports tournament planning	Tournify
(Sharma & Sureka, 2018)	CRUISE: A platform for crowdsourcing Requirements Elicitation and evolution	CRUISE
(Snijders et al., 2015)	REfine: A gamified platform for participatory requirements engineering	REfine
(Renzel et al., 2013)	Requirements Bazaar: Social requirements engineering for community-driven innovation	Bazaar
(Fernandes et al., 2012)	iThink : A game-based approach towards improving collaboration and participation in requirement elicitation	iThink

Table A.1.2: Identification of alternatives for root search terms over analysis of 20 most frequent words of known relevant articles. Adopted alternatives for root search terms are bold and green.

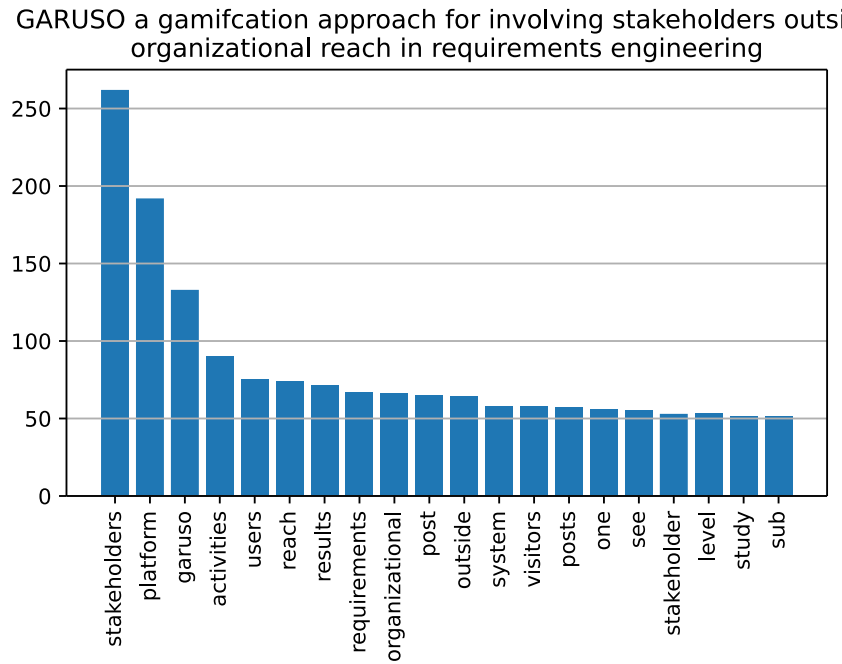
Ref.	20 most frequent words	Term adoption

(Wouters et al., 2021)



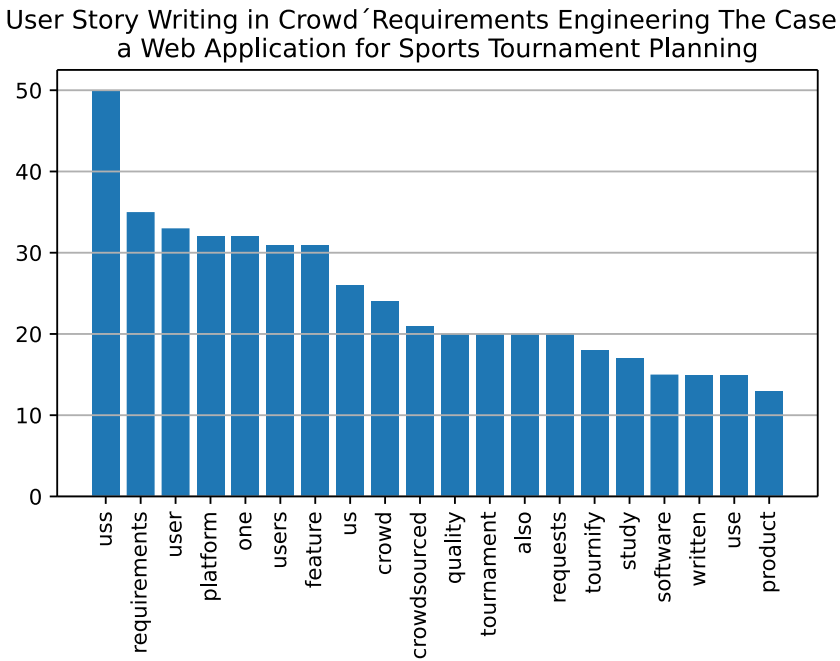
words	reason	mapped root search term
ideas	feedback can be an idea	feedback
crowd	users form a crowd	user
case	not relevant	
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
CrowdRE	leave away, to also include platforms which don't mention CrowdRE	
study	too specific	
KMar	specific approach	
elicitation	to elicit can be used for "to collect"	collect
user	already in search term	
users	already in search term	
platform	already in search term	
studies	too specific	
sys	not clear what is meant	
participants	feedback can be collected from e.g. platform participants	user
employees	employees can be users of a platform	user
research	too broad	
method	too broad	
governmental	not relevant	platform
two	not relevant	
organizations	not relevant	user

(Kolpondinos & Glinz, 2020)



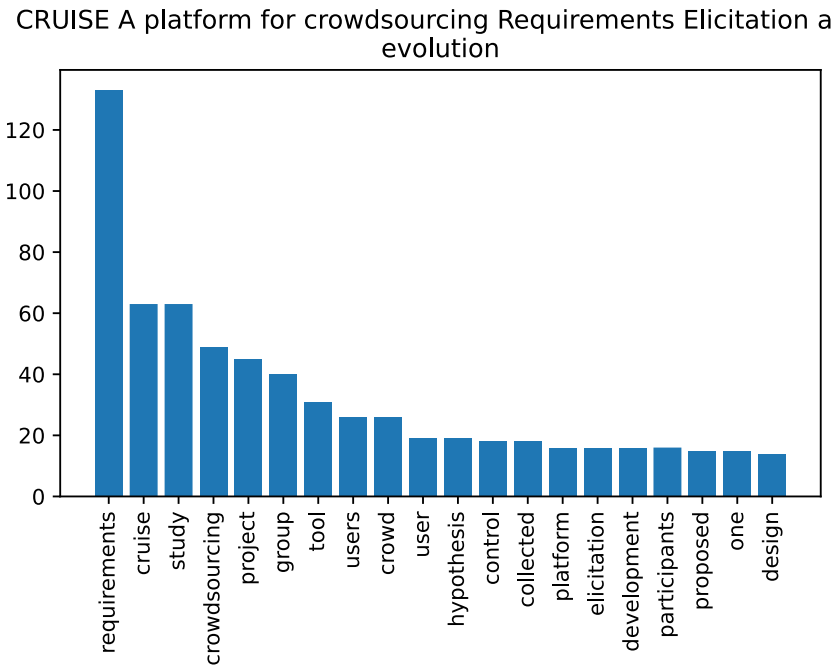
words	reason	mapped root search term
stakeholders	Stakeholders can be users of a platform	user
platform	already in search term	
garuso	specific approach	
activities	not relevant	
users	already in search term	
reach	not relevant	
results	not relevant	
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
organizational	not relevant	
post	feedback can be a post on a platform	feedback
outside	not relevant	
system	not relevant	
visitors	visitors of a platform	user
posts	See “post”	
one	not relevant	
see	not relevant	
stakeholder	See “stakeholders”	
level	not relevant	
study	too specific	
sub	not clear	

(Menkveld et al., 2019)



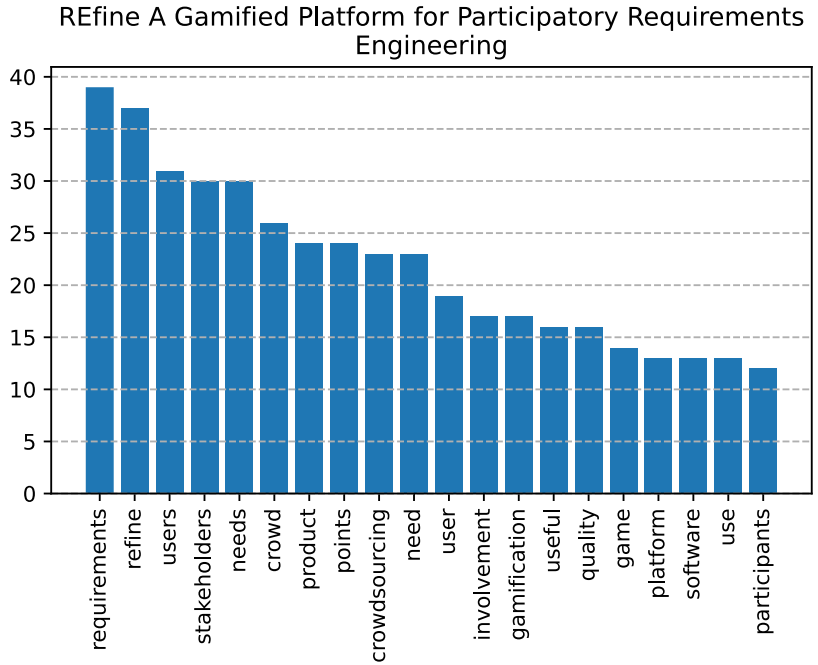
words	reason	mapped root search term
uss	Leave away to also include platforms that do not collect user stories (uss)	
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
user	already in search term	
platform	already in search term	
one	not relevant	
users	already in search term	
feature	not relevant	
us	Leave away to also include platforms that do not collect user stories (uss)	
crowd	users form a crowd	user
crowdsourced	Crowdsourcing is a means of collective contribution	collect
quality	not relevant	
tournament	not relevant	
also	filling word	
requests	request for feedback	collect
tournify	specific approach	
study	too specific	
software	a platform is a software	platform
written	not relevant	
use	not relevant	
product	cannot be mapped to root search term	

(Sharma & Sureka, 2018)



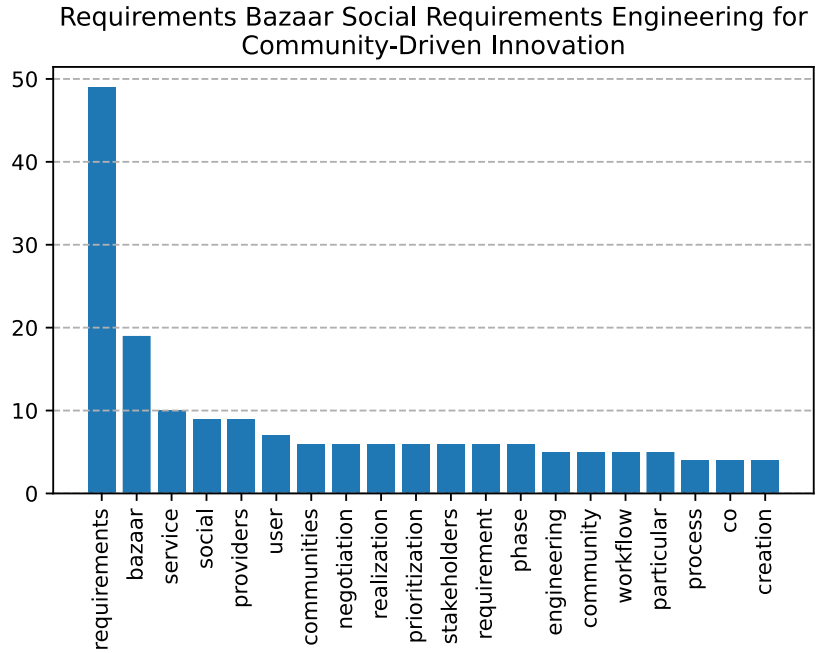
words	reason	mapped root search term
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
cruise	specific approach	
study	too specific	
crowdsourcing	Crowdsourcing is a means of collective contribution	collect
project	too general	
group	too general	
tool	a platform could also be named tool	platform
users	already in search term	
crowd	users form a crowd	user
user	already in search term	
hypothesis	too general	
control	too general	
collected	already in search term	
platform	already in search term	
elicitation	to elicit can be used for "to collect"	collect
participants	feedback can be collected from e.g. platform participants	users
proposed	too general	
one	not relevant	
design	not relevant	

(Snijders et al., 2015)



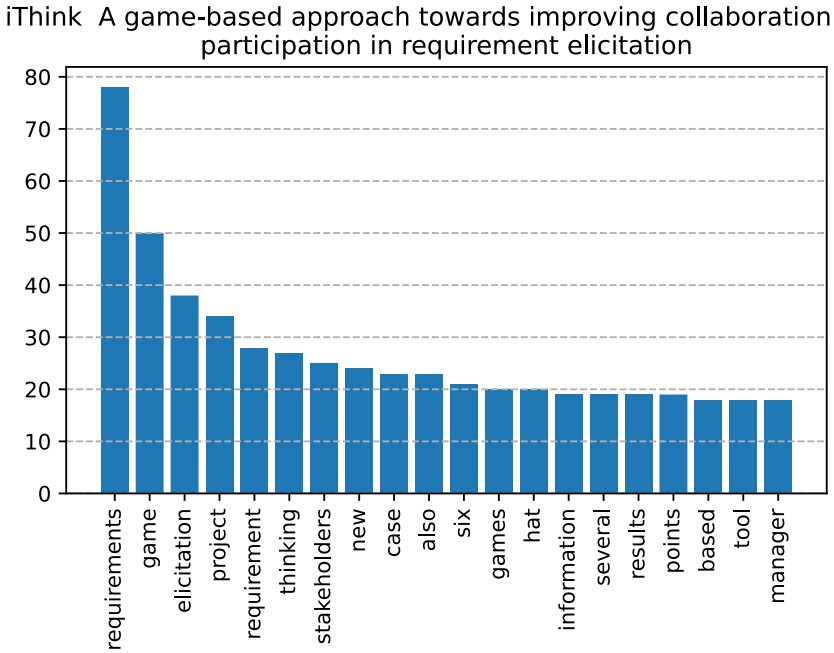
words	reason	mapped root search term
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
refine	too broad	
users	already in search term	
stakeholders	Stakeholders can be users of a platform	user
needs	too broad	
crowd	users form a crowd	user
product	cannot be mapped to root term	
points	cannot be mapped to root term	
crowdsourcing	Crowdsourcing is a means of collective contribution	collect
need	duplicate	
user	already in search term	
involvement	cannot be mapped to root term	
gamification	not relevant	
useful	not relevant	
quality	not relevant	
game	not relevant	
platform	already in search term	
software	a platform is a software	platform
use	too broad	
participants	feedback can be collected from e.g. platform participants	user

(Renzel et al., 2013)



words	reason	mapped root search term
requirements	Some platforms collect feedback in form of requirements (e.g. user stories)	feedback
bazaar	specific aproach	
service	already in search term	
social	not relevant	
providers	not relevant	
user	already in search term	
communities	a community of users	community
negotiation	Users can discuss and negotiate the relevance of feedback	
realization	not relevant	
stakeholders	Stakeholders can be users of a platform	user
requirement	to also include platforms that only collect feedback	
phase	not relevant	
engineering	not relevant	
community	See “communities”	
workflow	not relevant	
particular	not relevant	
process	too broad	
co	not relevant	
creation	not relevant	

(Fernandes et al., 2012)



words	reason	mapped root search term
game	not relevant	
elicitation	to elicit can be used for "to collect"	collect
project	not relevant	
requirement	Some platforms collect feedback in form of requirements (e.g. user stories)	
thinking	not relevant	
stakeholders	Stakeholders can be users of a platform	user
new	not relevant	
case	not relevant	
also	filling word	
six	not relevant	
games	not relevant	
hat	not relevant	
information	too broad	
several	not relevant	
results	not relevant	
points	not relevant	
based	not relevant	
tool	a platform could also be named tool	platform
manager	not relevant	

Table A.1.3: Identification of alternative terms for root search terms over analysis of known relevant articles. Adopted search terms are green and bold. Terms that are already part of the search term are orange.

Ref.	collect		user		feedback		platform	
	alternative	**reason**	**alternative**	**reason**	**alternative**	**reason**	**alternative**	**reason**
(Wouters et al., 2021)	collecting	already a search term	european	too specific	learning	not relevant	activity	too broad
	content	not relevant	author	too specific	section	not relevant	**application**	a platform is an application
	request	already a search term	leader	too specific	overview	not relevant	engineering	too broad
	cast	not relevant	avatar	too specific	support	not relevant	usage	too broad
	set	not relevant	employee	already a search term	**comment**	a comment is feedback	use	too broad
	crowd	already a search term	user	already a search term	**answer**	an answer to a question is feedback	**software**	a platform is software
	gather	can be used for "collect"	dummy	not relevant	literature	not relevant		
	elicit	already a search term	visitor	already a search term	classic	not relevant		
			client	can be used for "user" e.g. product client	service	not relevant		
			legislator	not relevant	practice	not relevant		
			broad	too broad	analysis	not relevant		
			mvp	not relevant	text	too broad		
			supplier	not relevant	usage	too broad		
			stakeholder	already a search term	process	too broad		
			neutral	not relevant	comparing	not relevant		
			sceptic	not relevant	explanation	not relevant		
			participant	already a search term	effort	not relevant		
					review	a review is feedback		

Ref.	collect		user		feedback		platform	
	alternative	**reason**	**alternative**	**reason**	**alternative**	**reason**	**alternative**	**reason**
(Kolpondinos & Glinz, 2020)	request	already a search term	equal	not relevant	summary	too broad	activity	not relevant
	set	not relevant	author	not relevant	game	not relevant	**application**	a platform is an application
	group	too broad	user	already a search term	conduct	not relevant	engineering	too broad
	gather	can be used for "collect"	achiever	not relevant	**comment**	A comment is feedback	usage	too broad
	elicit	already a search term	visitor	already a search term	**answer**	an answer to a question is feedback	world	not relevant
			stakeholder	already a search term	service	not relevant	use	too broad
			person	abstract version of user	survey	not relevant	body	not relevant
			strategist	too specific	**rating**	a rating is feedback	**software**	a platform is software
			worker	too specific	analysis	not relevant	technology	too broad
			individual	abstract version of user	behavior	not relevant		
			participant	already a search term	text	too broad		
			socializer	not relevant	segmentation	not relevant		
			player	not relevant	play	not relevant		
			member	member of a platform	effort	not relevant		
			explorer	not relevant	**review**	a review is feedback		
					announcement	not relevant		
					question	too broad		
					use	not relevant		
					research	too broad		
					defining	not relevant		

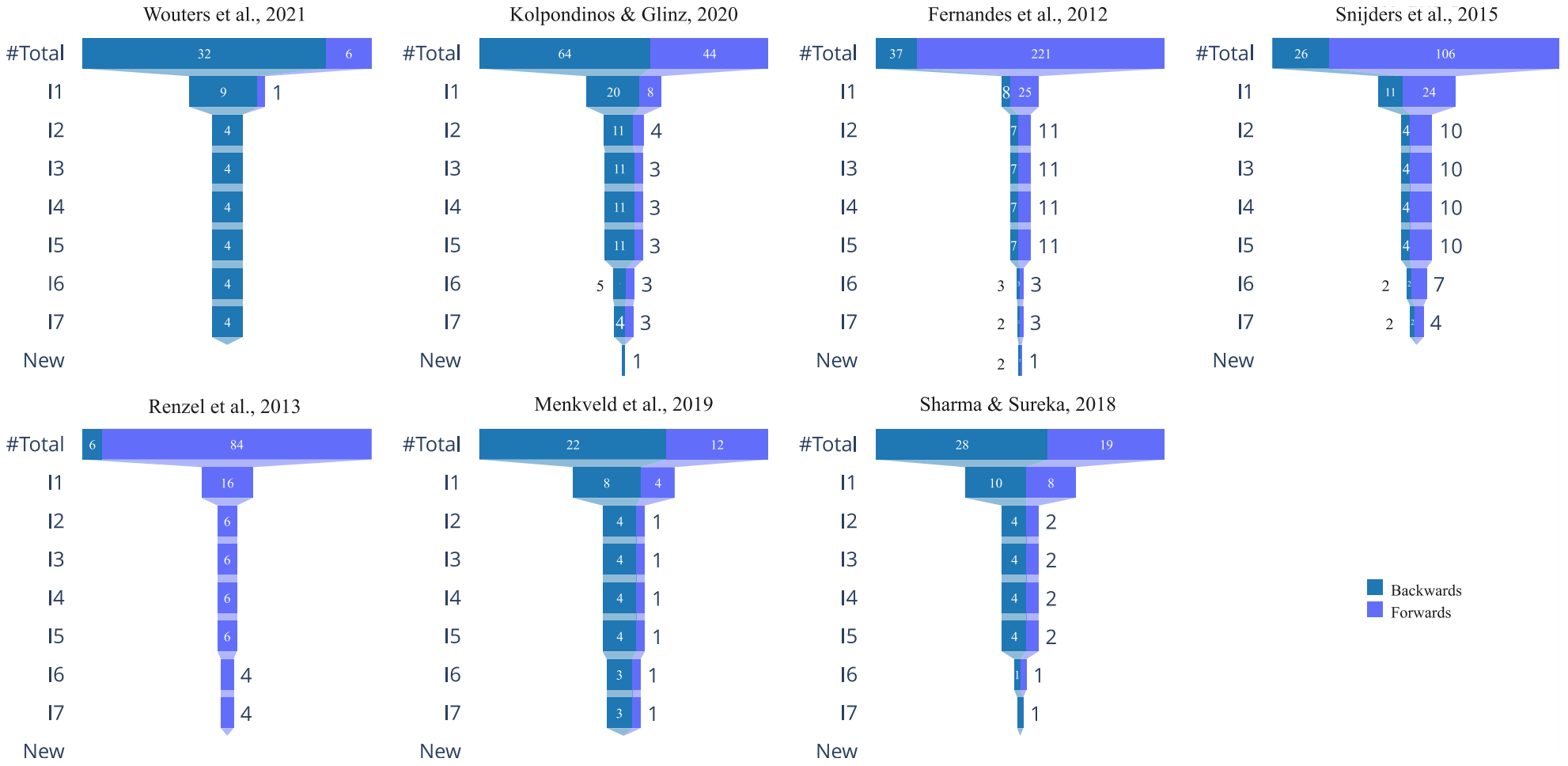
Ref.	collect		user		feedback		platform	
	alternative	**reason**	**alternative**	**reason**	**alternative**	**reason**	**alternative**	**reason**
(Fernandes et al., 2012)	**asking**	Asking someone to give feedback	european	not relevant	summary	too broad	activity	not relevant
	appeal	not relevant	user	already a search term	game	not relevant	**application**	a platform is an application
	set	not relevant	student	too specific	comparison	not relevant	engineering	too broad
	group	too broad	great	not relevant	**comment**	a comment is feedback	usage	too broad
	gather	can be used for "collect"	stakeholder	already a search term	learning	not relevant	world	not relevant
	elicit	already a search term	**person**	abstract version of user	section	not relevant	use	too broad
			designer	too specific	support	not relevant	body	not relevant
			european	not relevant	**answer**	an answer to a question is feedback	**software**	a platform is software
			user	already a search term	document	not relevant	technology	too broad
			student	too specific	literature	not relevant		
			great	not relevant	blog	not relevant		
			stakeholder	already a search term	asking	not relevant		
					rating	a rating is feedback		
					creation	not relevant		
(Snijders et al., 2015)	set	not relevant	participant	already a search term	practice	not relevant	activity	too broad
	group	too broad	customer	already a search term	**review**	a review is feedback	**application**	a platform is an application
	crowd	already a search term	user	already a search term	game	not relevant	engineering	too broad
	elicit	already a search term	stakeholder	already a search term	**comment**	a comment is feedback	use	too broad
	set	not relevant	player	too specific	learning	not relevant	**software**	a platform is software
					overview	not relevant	technology	too broad
					read	not relevant		
					rating	a rating is feedback		
					analysis	not relevant		
					process	not relevant		
					job	not relevant		

Ref.	collect		user		feedback		platform	
	alternative	**reason**	**alternative**	**reason**	**alternative**	**reason**	**alternative**	**reason**
(Renzel et al., 2013)	elicit	already a search term	leader	not relevant	project	not relevant	activity	too broad
			individual	abstract version of user	**comment**	a comment is feedback	**application**	a platform is an application
			engineer	too specific	learning	not relevant	engineering	too broad
			provider	too specific	section	not relevant	instrumentation	not relevant
			user	already a search term			usage	too broad
			stakeholder	already a search term			use	too broad
			developer	too specific			artifact	too broad
							software	a platform is software
(Menkveld et al., 2019)	request	already a search term	author	too specific	task	not relevant	utilization	too broad
	set	not relevant	user	already a search term	statement	too broad	**application**	a platform is an application
	group	too broad	customer	already a search term	measurement	not relevant	engineering	too broad
	crowd	already a search term	familiar	not relevant	project	not relevant	world	not relevant
	elicit	already a search term	organizer	not relevant	summary	too broad	use	too broad
	control	not relevant	administrator	too specific	**comment**	a comment is feedback	**software**	already a search term
			planner	too specific	overview	not relevant		
			person	abstract version of user	section	not relevant		
			participant	already a search term				
			engineer	too specific				
			developer	too specific				
(Sharma and Sureka, 2018)	avoid	not relevant	individual	already a search term	**review**	a review is feedback	**software**	not relevant
	include	too general	visionary	too specific	study	too general	engineering	too broad
	give	too general	customer	already a search term	investigating	not relevant	usage	too broad
	end	not relevant	advocate	too specific	comparison	not relevant		
	feature	not relevant						
	make	too general						
	gather							

Listing A.1.1: Final search term in IEEE command search format

```
("Abstract": collect OR "Abstract": crowdsourc* OR "Abstract": negotiat* OR "Abstract": request OR "Abstract": elicit OR
                                        "Abstract": gather)
                                              AND
 ("Abstract": user OR "Abstract": stakeholder OR "Abstract": crowd OR "Abstract": participant OR "Abstract": visitor OR
"Abstract": employee OR "Abstract": community OR "Abstract": communities OR "Abstract": client OR "Abstract": person OR
                                    "Abstract": individual OR "Abstract": customer)
                                              AND
                                ("Abstract": platform OR "Abstract": software OR "Abstract": tool)
                                           AND NOT
                                    "Document Title": blockchain
                                           AND NOT
                                  "Document Title" machine learning
                                           AND NOT
                                   "Document Title": deep learning
                                           AND NOT
                                  "Document Title": classification
                                           AND NOT
                                "Document Title": sentiment analysis
                                           AND NOT
                                     "Abstract": "fake review"
                                           AND NOT
                                     "Abstract": "code review"
                                           AND NOT
                                     "Abstract": "the idea of"
                                              AND
                                   "Abstract": requirement*
                                              AND
                                    "Abstract": engineering
```

Figure A.1.1: Included articles by *I1* - *I7* (Snowballing)



A.2 Results

Table A.2.1: Literature review

Ref.	Plat-form	Background and Motivation	RQs and problems	Principal ideas	Contribution
(Wouters et al., 2021)	KMar-Crowd	**Background** Crowd-based Requirements Engineering is a recent paradigm that promotes the active participation of a large number of stakeholders in RE. **Motivation** for evaluating the effectiveness of pull feedback: • The volume of studies is by far too limited for organizations to assess the potential and pitfalls of adopting pull-based elicitation practices **Motivation** for or studying the effectiveness of crowd-based elicitation in a governmental setting: • None of the existing studies were executed in a governmental setting	**RQs** *RQ1*: Can CrowdRE be used in a governmental setting to complement the requirements elicitation practices? **Problems** • Requirements elicitation and evolution are more constrained in governmental settings.	The paper presents two case studies of CrowdRE within the Royal Netherlands Marechaussee using the approach KMar-Crowd which adapts CrowdRE ideas to the needs of governmental organizations.	• KMar-Crowd description • Overlap comparison between KMar-Crowd-collected requirements and traditionally elicited requirements through techniques such as interviews task analysis and introspection - demonstrated via "S-Sys" case study involving 135 participants 32 ideas and over 300 votes. • Dynamics testing of a larger crowd - assessment of the utility of crowd-generated ideas through "V-Sys" case study with 385 participants 78 ideas and over 500 votes where elicitation was not previously conducted.
(Kolpondinos & Glinz, 2020)	GARUSO	**Background** The success probability of a software system strongly depends on the stakeholders participation in RE activities **Motivation** for developing a strategy for identifying stakeholders outside organizational reach • Typically the techniques used for identifying stakeholders assume that they can be identified among the members of the software	**RQs** • *RQ1*: How can we identify stakeholders outside organizational reach over diverse online channels? • RQ2: How can we build a platform that supports the collaborative participation of stakeholders outside organizational reach in eliciting and prioritizing requirements?	**Principal idea** The article describes the GARUSO approach which provides a strategy for identifying stakeholders outside organizational reach and a social media platform that enables large-scale collaborative elicitation and prioritization of requirements with gamification elements to motivate participation. and reports on its empirical evaluation	**Contribution** • Strategy for identifying stakeholders outside organizational reach based on exploratory study results. • Comprehensive description provision of GARUSO platforms' architecture and user interface. • Empirical demonstration of the approach effectiveness. • Derivation of initial design principles for identification and

		organization. However these assumptions no longer hold for many of todays software systems. **Motivation** for gamification elements of GARUSO: • Existing social media based RE platforms provide support for large-scale collaboration they assume that the collaborating stakeholders can be told to participate which is not the case for stakeholders outside organizational reach.	• RQ3: How effective is the GARUSO approach in attracting stakeholders outside organizational reach and supporting the collaborative elicitation and prioritization of requirements by these stakeholders? **Problems** • Todays software systems have stakeholders that are outside organizational reach • Stakeholders are not told to participate in RE activities.		participation of stakeholders outside organizational reach.
(Menkveld et al., 2019)	Tournify	**Background** The process of extracting informal stakeholders needs and translating them into formal specifications is a key process in Requirements Engineering (RE) **Motivation** for user stories • User stories may improve the quality of crowdsourced requirements **Motivation** for user involvement • improve system acceptance • diminish project failure • deliver greater system understanding • improve customer loyalty • broaden the market	**RQs** Not explicitly stated. Could be: • *RQ1*: What is the user participation of the platform? • *RQ2*: What is the quality and complexity of the user stories? • *RQ3*: What is the perceived usefulness of the platform? **Problems** • User stories sometimes lack context information	**Principal idea** The authors investigate how the platform can be employed to enable crowd workers to express requirements in the form of User Stories. They implement and validate the platform in the case of a web application for sports tournament planning.	**Contribution** • Introduction of the platform integrated into Tournify Tournament manager. • Report on evaluation of the platform including user participation and quality and complexity assessment of elicited user stories.
(Sharma & Sureka.	CRUISE	**Background** Crowdsourcing has aroused a lot of interest in Requirements Engineering (RE) research community. **Motivation** for crowdsourcing	**RQs** • *RO1*: How to design and develop a requirements elicitation platform?	**Principal idea** The authors propose the platform called CRUISE which is aimed at involving users in gathering analyzing validating prioritizing and	**Contribution** • Introduction of CRUISE platform highlighting its features and architecture.

		• Crowdsourcing approach can possibly meet the challenge of involving business users during requirements elicitation analysis prioritizing and negotiation	• *RO2*: How to investigate the effectiveness of the proposed platform in terms of the quality completeness and coverage of the elicited requirements? • *RO3*: How to compare and contrast the proposed crowdsourcing based platform with traditional approaches and discuss the limitations of the proposed approach and future research directions? **Problems:** • Deciding whether guest users should be allowed to contribute to projects or if only registered users have the permission. • Determining whether registration should be controlled or if any user can register and contribute to projects. • Identifying who holds the ownership of the project. […]	negotiating requirements. They conduct an experimental study to investigate the feasibility and viability of CRUISE.	• Report on evaluation of the platforms' applicability.
(Snijders et al., 2015)	REfine	**Background** In software product development stakeholder involvement is typically limited to representatives from Software Product Organizations (SPOs) and key clients excluding important stakeholders such as current and potential users. However two emerging trends crowdsourcing and gamification offer potential solutions by enabling access to a larger pool of	**RQs** Not explicitly stated They could be: • *RQ1*: What is the user participation of REfine? • RQ2: *What* is the acceptance of REfine? **Problems** • The role of end-users is often underestimated	**Principal idea** The authors propose REfine a gamified platform for requirements elicitation and refinement by involving a crowd of stakeholders. They conduct a case study where they analyze user participation user acceptance and expert opinions.	**Contribution** • Introduction of REfine platform providing details on its features and architecture. • Explanation of REfines role as an element of a crowd-centric requirements engineering method. • Report on initial evaluation of REfine through a case study specifically applied in the context

		stakeholders and keeping them motivated through feedback loops. **Motivation** for involving stakeholders • improved acceptance of a system • higher chances of project success • greater system understanding by the users • improved customer loyalty • broadened market • more accurate user requirements	• Interacting with users is challenging especially in terms of gaining access and obtaining consensus • Crowdsourcing responses are often noisier than expert data		of a governance risk and compliance tool.
(Renzel et al., 2013)	Bazaar	**Background** Traditional Requirements Engineering (RE) techniques are currently challenged by the massive scale openness diversity and uncertainty experienced with the Web. **Motivation** The innovation potential of niche communities often remains inaccessible to service providers due to a lack of awareness and effective negotiation between these two groups. Bringing together communities and service providers allows for requirements elicitation and realization.	**RQs** No RQs. Only the platform is presented **Problems: -**	**Principal idea** The authors present Requirements Bazaar a platform for Social Requirements Engineering (SRE).	**Contribution** • A description of requirements bazaar the co-creation workflow the workspace integration and the personalizable requirements prioritization
(Fernandes et al., 2012)	iThink	**Background** Requirements elicitation is a critical activity of the information systems development life cycle. **Motivation** • Stakeholders can be gathered at the same time and place • Lower logistic costs	**RQs** • *RQ1*: What is the effectiveness of the platform? • *RQ2*: What is the acceptance of the platform? **Problems** • Lack of user involvement	**Principal idea** The authors present a game-based platform called iThink that aims at improving the participation in a requirement elicitation process. The effectiveness and acceptance of the platform is evaluated in two case-studies.	**Contribution** • Demonstration of the effectiveness and acceptance of the platform.

(Lohmann et al., 2009)	WPFSRE	**Background** Social Software Engineering (SSE) is the application of processes, methods, and tools to enable community-driven creation, management, deployment, and use of software in online environments **Motivation** Existing RE tools are primarily designed for a small group of experts and provide limited support for collaboration among a diverse and large group of stakeholders. These tools often require additional tools for communication and collaboration, leading to a lack of transparency and traceability in the RE process.	**RQs** Not explicitly stated They could be: • *RQ1:* How to engage a larger group of stakeholders in the RE process using social software concepts? **Problems** Balancing conflicting demands such as simplicity and community orientation with the need for sufficient formality to meet typical requirements engineering demands like structured access	**Principal idea** The authors present a web platform that enables geographically distributed stakeholders to collaboratively collect, discuss, semantically enrich, and classify software requirements.	**Contribution** • Demonstration of the approach that integrates social software concepts with requirements engineering to enhance stakeholder engagement and collaboration. • Demonstration of the platform
(Yang et al., 2008)	WikiWinWin	**Background** There is already a tool called "EasyWinWin" to capture and negotiate requirements involving multiple stakeholders. **Motivation** EasyWinWin lacks features regarding requirements negotation. "WikiWinWin" is developed as a successor to EasyWinWin with more features,	**RQs** Not explicitly stated They could be: • *RQ1:* How to adopt the wiki technology to support active stakeholder participation and collaborative requirements negotiation **Problems** Consistency checking for resolved issues was not automated, requiring significant effort from the Shaper to facilitate task The system needed specific user interface improvements to enhance ease of use for stakeholders involved in the negotiation process.	**Principal idea** The authors presents WikiWinWin, a wiki-based system designed as a potential successor to EasyWinWin, aimed at facilitating collaborative requirements negotiation.	**Contribution** • Demonstration of the WikiWinWin requirements negotiation process • Case study of using WikiWinWin

(Laporti et al., 2009)	Athena	**Background** Quality requirements are essential for project success. **Motivation** The motivation for the authors in proposing the Athena approach is to address the limitations of traditional requirements elicitation methods, which often fail to capture the complete and nuanced requirements due to communication gaps, ambiguity, and the inherent complexities of stakeholders' needs	**RQs** Not explicitly stated They could be: • RQ1: How to capture detailed requirements through a collaborative storytelling approach? • RQ2: How to transform narratives into structured scenarios and then into use cases? **Problems** • Difficulty of converting narratives into structured formats	**Principal idea** The authors introduce Athena, a collaborative method for eliciting system requirements collective storytelling. This approach begins with stakeholders sharing narratives about their experiences with existing systems, which are then synthesized into a unified story. These stories evolve into scenarios, ultimately defining use cases, facilitating a progression from user narratives to actionable specifications.	**Contribution** • Approach and platform to support interaction • Experimental analysis to show effectiveness of the proposed approach
(Vogel et al., 2020b)	CrowdCore	**Background** Crowd-based RE comprises "automated or semiauto- mated approaches to gather and analyze information from a crowd to derive validated user requirements". **Motivation** The motivation for the authors to build a feedback platform is to ensure long-term realization of a software product's intended benefits post-implementation	**RQs** Not explicitly stated They could be: • How to develop design principles for continuous internal crowd-based RE? • How to implement a platform and process that implements the design principles? **Problems** • Delay in implementing a large volume of collected requirements could lead to user dissatisfaction and decreased participation. • The success of the approach depends on the crowd's ability to produce quality requirements, necessitating user empowerment and education.	**Principal idea** The authors of the paper propose leveraging crowdsourcing for software requirements engineering (RE) within organizations as their principal idea. They identify design principles for this through a literature review and develop a process and platform that implement the design principles.	**Contribution** • Design principles for crowdsourcing of requirements engineering • Process and platform that implement design principles

Table A.2.2: Literature synthesis (RQ1 and RQ2)

Ref.	Abr.	RQ1.1 (What feedback is collected?)	RQ1.2 (How is the feedback collected?) C=Crowd Feedback related functionalities are underlined Aspects with bullet points happen simulatenously	RQ1.4 (How is the collection influenced by the REengs?)	R2.1 (n what environment is the feedback collected?)	RQ2.2 (How many users is feedback collected from?)	RQ2.3 (How long is feedback collected?)
(Wouters et al. 2021)	KMar-Crowd	**Main feedback** • User Stories (called "Ideas") **Meta feedback** • Votes • Comments	4 Phases of KMar-Crowd method **1. Preparation** 1.1 Create core team 1.2 Prepare one core question (unclear) 1.3 Deploy Crowd **2. Ideas generation** 2.1 C: Submit main feedback 2.2 C: Vote main feedback 2.3 C: Comment main feedback **3. Refinement** 3.1 Write summary 3.2 C: Vote main feedback 3.3 C: Comment main feedback **4. Response and execution** 4.1 Comment main feedback 4.2 Develop and share timeline 4.3 Invite to focus group 4.4 Execute sprints Feedback is collected in phases 2,3 and 4	Requirement engineers write summaries and present them to users. Requirement engineers respond to ideas of users (phase 3 of KMar-Crowd)	**Product** S-Sys and V-Sys (operational systems) **Users** Employees of a large governmental organization (Royal Netherlands Marechaussee)	**S-Sys case study** From 135 users, 60 users gave feedback **V-Sys case study** From 385 users, 130 users gave feedback	**S-Sys case study** 33 days **V-Sys case study** 56 days

(Kolpondinos and Glinz 2020)	GARUSO	**Main feedback** User Stories (called "Post") • Title that describes a wish • Description of the wish • Description of the benefit one gets when wish is realized • Benefit label • Image upload that clarifies wish **Meta feedback** Sub-Post • Additional benefit description • Category Votes of post/sub-posts Rating of posts	• C: Submit main feedback • C: Submit sub-posts • C: Rate posts • C: Vote posts • C receive emails with summaries There is no specific phase where feedback is given. Giving feedback is initiated all the time autonomously by the users.	The requirement engineers send summaries of platform activities at day 19, 25, 31, 47 over email	**Product** Smart living application of Empa, the Swiss federal research institute for materials science and technology **Users** Acquired over internet	From 726 users, 32 users gave feedback	92 days
(Menkveld et al. 2019b)	Tournify	**Main feedback** User Stories • Role • Goal • Benefit • Category **Meta feedback** • Comments • Votes	• C: Submit main feedback • C: Vote main feedback • C: Comment main feedback There is no specific phase where feedback is given. Giving feedback is initiated all the time autonomously by the users.	• The REengs initiated the first request and commented on some of the requests during the study • They were also able to label features as in development or done.	**Product** Tournify Tournament Manager **Users** Users of Tournify Tournament Manager	From 157 users, 39 users gave feedback	35 days
(Sharma and Sureka 2018)	CRUISE	**Main feedback** • Free text **Meta feedback** • Comment • Score	• C: Submit main feedback • C: Comment main feedback • C: Score main feedback There is no specific phase where feedback is given. Giving feedback is	No influence.	**Product** Student registration tool **Users**	18 undergraduate students	Not described.

			initiated all the time autonomously by the users.		Undergraduate students		
(Snijders et al., 2015)	REfine	**Main feedback** • Free text **Meta feedback** • Comments • Votes	CCRE method **1. Feasibility analysis (crowdsourcing potential)** **2. Context analysis (Stakeholder, feedback channel)** **3. Crowdsourcing preparation (crowd)** **4. Crowd involvement** • C: Submit main feedback • C: Comment main feedback • C: Vote main feedback • C: Create branches of main feedback **5. Requirement identification (requirement)** **6. Focus group execution (sub-crowd, decision)** **Sprint (release)** Feedback is only given in phase 4	• The REengs provision the guidelines • The REengs delete irrelevant needs • The REengs send weekly updates to improve activity of users	**Product** Qubus, a Governance Risk and Compliance (GRC) web platform for compliance. Company: KPMG **Users** Employees, clients and users of clients	From 19 users, 19 users gave feedback	35 days
(Renzel et al. 2013)	Bazaar	**Main feedback** User Stories **Meta feedback** • Voting • Comments	**1. Idea Generation** • C: Create main feedback **2. Idea Selection** Negotiation among stakeholders takes place, until one service provider commits to take the lead for realization **3. Idea Realization** Refinement and negotiation continue. **4. Release**	REengs comment ideas.	**Product** Not described. **Users** Not described.	Not described.	Not described.

			A final solution is acknowledged, possibly leading to new user stories In phase 2 and 3: • C: Vote main feedback • C: Comment main feedback Feedback is given in phases 1, 2 and 3				
(Fernandes et al. 2012)	iThink	**Main feedback** • Free text **Meta feedback** • Comments • Ratings	**1. Project manager creates initial requirements** **2. Feedback collection** • C: Submit main feedback • C: Comment main feedback • C: Rate main feedback Feedback is given only in Phase 2	No influence.	**Product** First case study: Information system of childcare center Second case study: course management system **Users** Employees of the childcare center	First case study: From 7 employees, 7 gave feedback Second case study: From 17 students, 17 gave feedback	Not described.
(Lohmann et al., 2009)	WPFSRE	**Main feedback** • Free text **Meta feedback** • Comments • Votes (agreement or disagreement • Ratings (5 point scale regarding quality) • Relations	• C: Submit main feedback • C: Comment main feedback • C: Vote main feedback • C: Rate main feedback • C: Define relations between main feedback There is no specific phase where feedback is given. Giving feedback is initiated all the time autonomously by the users.	REengs supervise and moderate discussions	**Product** Not described. **Users** Not described.	Not described.	Not described.

(Laporti et al., 2009)	Athena	**Main feedback** • Stories (Template) • Scenarios (Template) • Use cases (Template) **Meta feedback** • Comments	**1. Collection of stories** • C: Submit stories • C: Comment stories **2. Transform stories to scenarios** • C: Submit scenarios • C: Comment scenarios **3. Transform scenarios to use cases** • C: Submit use cases • C: Comment use cases Feedback is given in phases 1,2 and 3	No influence.	**Product** System that sells movie tickets on the web **Users** No info	From 6 users, 6 gave feedback	4 hours
(Yang et al., 2008)	WikiWinWin	• Free text (called "Win condition" = Stakeholder objectives) • Rating	**1. Set up WinWin Negotiation Context** **2. Negotiate WIOAs (Win Condition, Issue, Option, Agreement)** • C: Submit main feedback • C: Rate main feedback Feedback is given only in phase 2	No influence.	**Product** Medieval East Asian Tombs Database system **Users** Graduate students	From 6 graduate students, 6 gave feedback	2 sessions of 2 hours in-negotiation meeting
(Vogel et al., 2020)	CrowdCore	**Main feedback** • Free text **Meta feedback** • Comments • Votes	**1. Ideation** • C: Submit main feedback • C: Comment main feedback **2. Consolidation** Product Owner (PO) determines which requirements proceed to voting phase. **3. Voting** • C: Vote main feedback **4. Decision** PO select requirements based on expected value for users. Users give feedback over comments. • C: Comment main feedback Feedback is given in phase 1, 3 and 4.	PO motivates users to interact and participate by providing incentives such as praise and encouragement.	**Product** Seaport organization **Users** Employees of the organization	Not described	Not described

Table A.2.3: Literature synthesis (RQ3 and RQ4)

Ref.	Abr.	RQ3.1 (How is the user participation evaluated and what are the results?)	RQ3.2 (How is the feedback evaluated and what are the results?)	RQ3.3 (How is the platform evaluated and what are the results?)	RQ3.4 (How do the findings of the feedback affect the software?)	RQ4.1 (What functionalities to collect feedback does the platform offer?)	RQ4.2 (What additional functionalities does the platform offer?)	RQ4.3 (Is the platform accessible?)
(Wouters et al. 2021)	KMar-Crowd	**S-Sys Interactions** • Main feedback: 32 • Votes: 284 • Comments: 28 **V-Sys Interactions** • Main feedback: 78 • Votes: 453 • Comments: 78 **S-Sys Users** • Invited users: 478 • Accessed users: 135 • Contributing users: 60 **V-Sys Users** • Invited users: 2393 • Accessed users: 385 • Contributing users: 130	**KANO model** • Must-be (S: 13, V: 50,6%) • One-dimensional (S: 10, V: 36,7%) • Attractive (S: 7, V: 12,7%) **Gathered earlier** • Completely (S: 19) • Partly (S: 6) • Not at all (S: 5) **Complete** Complete for dev teams (S: 11) Enough for MVP (V: 59,5%) Enough for product (V: 27,8%) **Granularity** • Epic (V: 40,5%) • User Story (V: 54,5%) • Not applicable (5,1%)	• **Acceptance** **Way of working** (S/V: positive) • **Gamification** (V: Did not increase motivation)	Not described.	• Submit main feedback • Vote feedback • Comment feedback	Gamification elements • Receiving points • Receiving badges • Show a leaderboard • Receiving stars	It is internally accessible from the KMar Network.

(Kolpondinos and Glinz 2020)	GARUSO	**Interactions** • Main feedback: 56 • Logins: - • Comments: - **Users** • Invited users: - • Accessed users : 726 • Contributing users: 32	No feedback is evaluated.	Not described.	Not described.	• Submit main feedback • Submit sub-post • Rate posts and sub-posts • Vote posts and sub-posts	Gamification elements • Receiving badges • Do challenges • Receiving points • Get up levels • Receiving rewards Onboarding mechanism FAQ page Overview of an extract of feedback of other users Sharing posts and sub-posts with users and social media	Not described.
(Menkveld et al. 2019b)	Tournify	**Interactions** • Main feedback: 57 • Votes: 89 • Comments: 14 **Users** • Invited users: 337 • Accessed users : 157 • Contributing users: 39	**Quality** (Quality US framework) • 52% of the user stories met all quality aspects and 48% of the user stories contained one or more easily preventable error(s). **Complexity** (amount of work to implement US) • 9/10 crowd- sourced USs can be developed within one workday. 1 US could not be estimated, because	**Acceptance Questionnaire (perceived usefulness)** • 10 users answered questionnaire and they found the platform as very useful.	Not described.	• Submit main feedback • Vote main feedback • Comment main feedback	• Main feedback overview	Not described.

			it was formulated too vaguely. 7 USs were already implemented but overlooked by the user.					
(Sharma and Sureka 2018)	CRUISE	Not described.	**Gathered earlier** • Comparison of requirements elicited by interviews and CRUISE • The results are comparable	Not described.	Not described.	• Submit main feedback • Comment main feedback • Score main feedback	• Moderator can finalize main feedback for development • Import and export of main feedback • User and role management • Dashboard of main feedback • Controlled user registration • Follow main feedback	Not described.
(Snijders et al., 2015)	REfine	**Interactions** • Main feedback: 21 • Votes: 130 • Comments: 37 **Users** • Invited users: 37 • Accessed users : 19 • Contributing users: 19	Not described.	**Acceptance Questionnaire** • 17 users found the process as difficult, more useful and more engaging compared with previous feedback experiences • Voting and commenting were found as more useful than branching • The participants agreed that the game elements made the experience more pleasant **Interview with experts**	Not described.	• Submit main feedback • Comment main feedback • Vote main feedback • C: Create branches of main feedback	Gamification elements • Roles • Exploration • Points • Group forming • Endorsements • Leaderboards Contact page Main feedback overview User profile	Not described.

				• Experts found crowdsourcing useful for requirements elicitation, negotiation, and specification, and agreed that CCRE improves the quality of the RE process and provides valuable requirements.				
(Renzel et al. 2013)	Bazaar	Not described.	Not described.	Not described.	Not described.	• Submit main feedback • Vote main feedback • Comment main feedback	• Synchronization with issue trackers • List of ranked main feedback • Follow and share main feedback	A ready-to-use installation is available at requirements-bazaar.org.
(Fernandes et al. 2012)	iThink	• Main feedback (F: 10, S: 22) • Comments (F: 15, S: 86)	F = First case study S = Second case study **Sentiment** • Positive comments (F: 6, S: 48) • Neutral comments (F: 3, S: 32) • Negative comments (F: 6, S: 6)	F = First case study S = Second case study **Acceptance** • High level of satisfaction (F)	Not described.	• Submit main feedback • Comment main feedback • Rate main feedback	Gamification elements • Points • Stars • Ranking Support for multiple projects at once Filter main feedback Group main feedback	Not described.
(Lohmann et al., 2009)	WPFSRE	Not described.	Not described.	Not described.	Not described.	• Submit main feedback • Comment feedback • Vote feedback • Rate feedback • Define relations	• Edit main feedback • Revision history (track, rollback changes) • Overview over feedback	In general everyone with internet access and a web browser can access the wiki.

						between feedback	• Tree navigation along classification for main feedback • Tagging for main feedback • Word cloud for tags • Export main feedback	Only registered users can add and edit. The web platform itself doesn't exist anymore.
(Laporti et al., 2009)	Athena	Not described.	Not described.	Not described.	Not described.	Submit main feedback • Submit stories • Submit scenarios • Submit use cases • Comment feedback	• Administrator role • Manage users • Create project • Giving user moderator role • Project glossary (user can add new terms)	Not described.
(Yang et al., 2008)	WikiWinWin	• Main feedback (win conditions): 62	Not described.	**Feature comparison** WikiWinWin is compared to EasyWinWin and SOP-Wiki. The results were that WikiWinWins main strengths are the exchange of ideas and knowledge, the content editing and versioning. Main weaknesses are its lacking automated consistency checking and problems with conflicts during editing conflicts.	Not described.	• Submit main feedback • Rate feedback	• Edit main feedback • Synchronous collaboration • Exporting main feedback • Content editing and versioning	Not described.

(Vogel et al., 2020)	CrowdCore	• Not described.	Not described.	**Acceptance** **Conversation with experts** The approach was found effective in involving users in the requirements engineering process. Concerns were raised about the applicability of the approach for all software types and the need for users to trust product owners to implement prioritized requirements.	Not described.	• Submit main feedback • Comment main feedback • Vote main feedback	• User profile page • Main feedback overview page • Main feedback quality check • Search functionality • POs can give effort estimation • Status system: Open, Backlog, In Progress, Implemented, Closed • Subscribe to main feedback	Not described.

B Supplementary material for the treatment Design

B.1 Coding

The goal of the coding is to assign specific *coding attributes* to feedback that is collected over the app "smartFEEDBACK" in the context of the SMART-AGE research project. Feedback can either be an answer to a question or a message. The coder conducts the coding over an excel sheet that is provided by the supervisor. The excel sheet is provided in the repository[53].

1. Reading the feedback and feedback context

The first part of coding feedback, whether it is an answer or a message, is reading the feedback itself and its context. The columns in the excel sheet that represent the feedback and its context are highlighted blue.

The data that is relevant for an answer and its context is described in Table B.1.1. The table lists the data along with an explanation, examples and the columns of the excel sheet that contain the data.

Table B.1.1**:** Data that is given to the coder for coding freetext answers

Data	Explanation	Example	Column
All subquestions and the answers to all subquestions of a question.	A question can be divided into two subquestions. A subquestion can either be a selection (e.g. choosing "Very good") or a freetext. All answers are provided, but only the freetext answer is coded.	Question: Wie gut finden Sie das Anzeigen von Links zu Webseiten in smartVERNETZT? Warum? Subquestion 1: Wie gut finden Sie das Anzeigen von Links zu Webseiten in smartVERNETZT? (Selection:) Subquestion 2: Warum? (Freitext)	subquestion_1 answer_for_subquestion_1 subquestion_2 answer_for_subquestion_2
App	The app that the question belongs to.	smartVERNETZT	app
Non-functional requirement (NFR) that the question addresses	The NFR which the question addresses. If the question does not address an NFR, the field is empty.	Question: Wie leicht fiel es Ihnen smartFEEDBACK zu erlernen? NFR: Learnability (Product Quality Model – Usability)	associated_nfr
System Function (SF) that the	The SF which the question addresses. If the question does not address an SF, the field is	Question: Wie gut finden Sie das Anzeigen von Links zu Webseiten in	associated_sf

[53] https://github.com/lradeck/dissertation/blob/main/coding_template_and_examples.xlsx

question addresses	empty. SFs for the app smartFEEDBACK are listed in Table B.1.5. SFs for the app smartVERNETZT are listed in Table B.1.7.	smartVERNETZT? Warum? SF: Display Links

Table B.1.2 lists example questions and answer options.

Table B.1.2: Example questions and answer options

Example Question	Answer options
How good do you find the history function in SF? Why?	Likert scale selection (Very good, good, etc.) Freetext
Are there any problems with displaying the history in SF? If yes, which ones?	Yes/No selection Freetext
Can the display of the history in SF be improved? If yes, how?	Yes/No selection Freetext
Are you concerned about the security of your data in smartVERNETZT? Why?	Yes/No selection Freetext
What is the reason that you have not looked at a question in SF in the last week? How could the app be im-proved so that you use it more often?	Freetext Freetext
I can well imagine using SV regularly. Why is that?	Likert scale selection, Freetext

The data that is relevant for a message and its context is described in Table B.1.3.

Table B.1.3: Data that is given to the coder for coding messages

Data	Explanation	Example	Column
Title	The title of the message	„Löschen von Neuigkeiten“	title
App	The app that the question belongs to.	smartVERNETZT	app
Description	Freetext of message	Question: Wie leicht fiel es Ihnen smartFEEDBACK zu erlernen? NFR: Learnability (Product Quality Model – Usability)	description
Contains voicemessage?	The message can contain a voice message (Yes/No)	Yes	contains_voicemessage
Transcript	If the message contains a voice message, the transcript is stored here.	“Ich wünsche mir …”	transcript

2. Coding

The coder assigns the *coding attributes* of Table B.1.4 to the feedback. The columns in the excel sheet that represent the *coding attributes* are highlighted orange. Some coding attributes only need to be coded under certain conditions. Excel automatically colors cells that are not necessary in gray.

Table B.1.4: *Coding attributes* that are generated by the coder during the coding of feedback.

Data	Format	Explanation	Example	Column
Feedback is comprehensible	Yes/No	The comprehensibility of the feedback is checked. The feedback can either be comprehensible or not. Comprehensible means, the coder understands the feedback syntactically and semantically.	Example for syntactically incorrect feedback „Dihj tut mir nicht gbn" Example for semantically incorrect feedback: "Das Design ist mein Haus"	comprehensible
The following data is coded for comprehensible feedback				
Extracted statement 1..n	One or more subsequent sentences. A sentence can also be divided into subordinate clauses ("Nebensätze")	The feedback is split into statements if it contains multiple parts that can be associated to different classes (see Table B.1.9).	See excel tab "Beispiele": Row 9	statement[N] – where N is number of statement
The following data is coded per statement				
Class (see Table B.1.9)	Selection from predefined set (see Table B.1.9)	(see Table B.1.9)	(see Table B.1.9)	class[N]
The following data is coded for each statement that is class: • **Positive, neutral or negative statement about app or app context** • **Actionable change request** • **Non-actionable change request** • **Problem**				
Associated requirement(s) **Note:** For the process this is only required for CR, but we need the requirement also for not CR to answer RQ2.2	Selection from predefined set (Table B.1.5, Table B.1.6, Table B.1.7, Table B.1.8). The associated SF or NFR of the question is listed as the default value in excel for Req1.	In general: Map the statement to a NFR, UT, ST, SF, W. <u>For NFR</u>: Map the statement to an NFR if it addresses mainly non-functional aspects. **Example**: Excel Row 10 In the case the statement addresses a non-functional aspect about a SF or W, map it as well to the SF or W. **Example**: Excel Row 11		Req[N]

		For UT, ST, SF & W: Map the statement always to the most technical requirement (SF & W > ST >UT) where ">" means "more technical than". Explanation: Map the statement to a system function (SF) if it addresses mainly functional aspects. **Example**: Excel Row 6 Map the statement to a workspace (W) if it addresses mainly aspects about the user interface, even if a system function is mentioned. **Example**: Excel Row 7 Map the statement to a SubTask, (ST) if (1) it mainly addresses new or existing aspects about the SubTask (**Example**: Excel Row 12) or (2) it is a change request that a) wishes for changes of the system support for this SubTask (**Example**: Excel Row 4) or b) is actionable and wishes for new system functions/workspaces to be added to the SubTask (**Example**: Excel Row 5) Map the statement to a UserTask (UT), if it addresses aspects about the app or app context which cannot be associated with an existing Sub Task (ST). **Example**: Excel Row 3		
The following data is coded only for statements that have the class "Actionable change request" or "Non-actionable change request"				
Reason for change request		The coder gives a reason for why the statement is a change request	See excel tab "Beispiele": Row 4,5,6,7	Reason CR[N]
Reason for choosing "Actionable change request" oder Non-Actionable	Freetext	Actionable: The coder gives a reason for choosing why he/she thinks the change request is actionable. Please give an	See excel tab "Beispiele": Row 4,5,6,7	Reason (Non)Actionable[N]

		explanation of how the change looks like. Non-actionable: The coder gives a reason for choosing why he/she thinks the change request is non-actionable. Please give an explanation of why it is not clear how the change looks like.	
Reason for associated requirement	Freetext	The coder gives a reason for choosing why he/she thinks the requirements is associated and not another (e.g. Workspace vs. System Function). (**Example**: Excel Rows 6,7) If the requirement is a SubTask: Indicate the reason why it is mapped to a SubTask: (1) it mainly addresses new or existing aspects about the SubTask (**Example**: Excel Row 12) or (2) it is a change request that a) wishes for changes of the system support for this SubTask (**Example**: Excel Row 4) or b) is actionable and wishes for new system functions/workspaces to be added to the SubTask (**Example**: Excel Row 5)	Reason Req[N]

Table B.1.5: smartFEEDBACK: User Tasks (UT) and Subtasks (ST) and System Functions (SF)*

Requirement
UT1: Older Adults Give And Manage Feedback
UT1S1: Express Own Feedback
SF: Navigate To SMART-AGE Portal
SF: Display Question
SF: Skip Question
SF: Display Feedback Form
SF: Display Application Information
SF: Add File
UT1S2: Discuss Given Feedback
SF: Display Comment
SF: Add Comment
SF: Add Audio Recording
SF: Filter Private Answer And Feedback
SF: Display Private Answer And Feedback
UT1S3: Review Given Feedback
SF: Display History Details
SF: Filter History
SF: Display History

Table B.1.6: smartFEEDBACK: Workspaces (W) and non-functional requirements (NFR)*

Requirement
W1: Question View
W3: Private Answer And Feedback View
W4: Feedback View
W5: History View
W6: Instruction View
W7: Comment View
W8: SMART-AGE Portal View
W9: Detail View
W10: Sidebar View
W11: Public Answer And Feedback View
NFR: Time Behaviour
NFR: User Error Protection
NFR: Accessibility
NFR: Modifiability
NFR: Compatibility
NFR: Security
NFR: Learnability
NFR: Operability
NFR: Comfort
NFR: Pleasure
NFR: Trust
NFR: Usefulness, Effectiveness, Efficiency

* The listed requirements have been revised in Section 5.1. See excel sheet tab “Info” for a mapping of requirements.

Table B.1.7: smartVERNETZT: User Tasks (UT) and Subtasks (ST) and System Functions (SF)

Requirement
UT1: Older Adults Inform Themselves About Various Topics
UT1S1: Get Information About Health Related Topics
SF: Navigate To SMART-AGE Portal
SF: Display Link
SF: Add Personal Link
SF: Display Personal Link
SF: Delete Personal Link
SF: Display Categories
SF: Display Application
SF: Display external Website
UT1S2: Get Information About Leisure Activities
[Same SF as UT1S1]
UT1S3: Get Information About News
[Same SF as UT1S1]
SF: Display News Notifications
SF: Delete News Notification
SF: Add Event To Calendar
UT1S4: Get Information About The Weather
[Same SF as UT1S1]
SF: Display Weather Information

Table B.1.8: smartVERNETZT: Workspaces (W) and non-functional requirements (NFR)

Requirement
W1: Home View
W2: Category View
W3: Link View
W4: External Website View
W5: SMART-AGE Portal View
W6: Native App View
NFR: Modifiability
NFR: Accessibility
NFR: User Error Protection
NFR: Time Behaviour
NFR: Compatibility
NFR: Security
NFR: Learnability
NFR: Operability
NFR: Comfort
NFR: Pleasure
NFR: Trust
NFR: Usefulness, Effectiveness, Efficiency

Table B.1.9: Classes for feedback coding

Class	Explanation	Example
Irrelevant statement	Statement that does **not** describe the app or app context and is thus not relevant for coding. **Definition of app context:** smartFEEDBACK: • Older adults (OAs) give and manage feedback smartVERNETZT: • OAs inform themselves about various topics • OAs entertain themselves • OAs communicate with people	"Es regnet heute"
Positive, neutral or negative statement about app or app context	Statement that describes the app or app context in a positive, neutral or negative way. OAs organize their everday life	Positiv: "smartFEEDBACK finde ich gut" Neutral: "smartFEEDBACK ist eine App zum Abgeben von Feedback" Negative: „Ich gehe nicht gerne joggen" "smartFEEDBACK ist blöd"
Actionable change request	Statement explicitly or implicitly requesting a change. The coder can derive a requirement improvement or a new requirement based on the statement.	Explicit: "Ich will Funktion X" Implicit: "Ich finde die Schriftgröße sehr klein"
Non-actionable change request	Statement explicitly or implicitly requesting a change. The requirements engineer needs more information to derive a requirement improvement or a new requirement based on the statement.	Explicit: "Ich will, dass ich Dinge schneller finde" Implicit: "Aus meiner Sicht kann man Dinge nicht schnell finden"
Problem	Statement describing that an existing functionality **does not work**.	„Es ist nicht möglich zu filtern"
Cannot answer question	Statement indicating that the question could not be answered.	„Ich kann die Frage nicht beantworten" oder „Ich verstehe die Frage nicht"
Reference to other answer	Statement indicating that the question has already been answered by another response.	„Das habe ich schon bei der vorherigen Antwort beschrieben"

3. Examples

Further examples for coding are given in the tab "Beispiele" of the Excel sheet. The excel sheet can be found in the repository.

B.2 Selection of FUQ

We choose which FUQ we want to ask the users based on their characteristics and based on additional aspects, such as the type of associated requirement and the associated app. We first selected 8 FUQ3 based on their characteristics, where we aimed for an equal distribution. We present the selection of FUQ3 for round 1 iteration 1 in Table B.2.**1**. We also list additional aspects such as the associated requirement type and the app in the table, because these are relevant for the selection of FUQ1 and FUQ2. The abbreviations for the associated requirement types are: SF = System function, W = Workspace, UTS = Subtask and UT = User Task. The app abbreviations are SF = smartFEEDBACK and SV = smartVERNETZT.

Table B.2.1: Selection of FUQ3 based on their characteristics for round 1 iteration 1

Type	Readable ID	Screenshot	**Characteristics**				**Additional aspects**	
			Innovative	Solution proposed through mockup	Percentage associated users	Number of FUQ	Associated requirement type	App
FUQ3	AV1	Figure B.2.52	Yes	No	> 2%	1	SF	SF
FUQ3	AV2	Figure B.2.53	Yes	No	<2%	1	SF	SV
FUQ3	AV3	Figure B.2.54	Yes	Yes	>2%	1	SF	SF
FUQ3	AV4	Figure B.2.55	Yes	Yes	<2%	1	W	SV
FUQ3	AV5	Figure B.2.56	No	No	>2%	1	W	SF
FUQ3	AV6	Figure B.2.57	No	No	<2%	1	W	SF
FUQ3	AV7	Figure B.2.58	No	Yes	>2%	1	SF	SV
FUQ3	AV8	Figure B.2.59	No	Yes	>2%	1	W	SF

We then selected 8 FUQ1 and FUQ2 also alone based on their characteristics, where we aimed for an equal distribution (Table B.2.2).

Table B.2.2: Selection of 8 FUQ1 and FUQ2 based on their characteristics for round 1 iteration 1

Type	Readable ID	Screenshot	Characteristics				Additional aspects	
			Innovative	High cognitive effort	Percentage associated users	Number of FUQ	Associated requirement type	App
FUQ1	FF1	Figure B.2.41	Yes	High	> 2%	1	SF	SF
FUQ1	FF2	Figure B.2.42	Yes	High	<2%	1	SF	SV
FUQ2	FF3	Figure B.2.43	Yes	Moderate/Low	>2%	1	SF	SF
FUQ2	FF4	Figure B.2.44	Yes	Moderate/Low	<2%	1	W	SF
FUQ2	FF5	Figure B.2.45	No	High	>2%	1	UT	SF
FUQ2	FF6	Figure B.2.46	No	High	<2%	1	W	SF
FUQ2	FF7	Figure B.2.47	No	Moderate/Low	>2%	1	SF	SV
FUQ1	FF8	Figure B.2.48	No	Moderate/Low	>2%	1	W	SV

To address different requirements and apps equally, we decided to select the last 3 FUQ1 and FUQ2 based on the additional aspects (Table B.2.3).

Table B.2.3: Selection of 3 additional FUQ1 and FUQ2 based on the additional aspects for round iteration 1

Type	Readable ID	Screenshot	Characteristics				Additional aspects	
			Innovative	High cognitive effort	Percentage associated users	Number of FUQ	Associated requirement type	App
FUQ1	FF9	Figure B.2.49	Yes	High	>2%	1	UTS	SF
FUQ2	FF10	Figure B.2.50	Yes	Moderate/Low	>2%	1	W	SV
FUQ2	FF1	Figure B.2.51	No	High	>2%	1	UT	SV

For the second round, we applied the same strategy of selecting FUQ. We couldn't achieve an equal distribution of FUQ3 regarding their characteristics, because for some combinations of characteristics no FUQ3 existed anymore. This was the case for 2 FUQ3. We crossed out the combinations in Table B.2.4.

Table B.2.4: Selection of FUQ3 based on their characteristics for round 2 iteration 1

Type	Readable ID	Screenshot	Characteristics				Additional aspects	
			Innovative	Solution proposed through	Percentage of associated users	Number of FUQ	Associated requirement type	App
FUQ3	2_AV1	Figure B.2.82	Yes	No	> 2%	1	SF	SF
FUQ3	2_AV2	Figure B.2.83	Yes	No	<2%	1	UTS	SF
FUQ3	2_AV3	Figure B.2.84	Yes	No	<2%	1	W	SF
FUQ3	2_AV4	Figure B.2.85	Yes	Yes	>2%	1	W	SV
FUQ3	2_AV5	Figure B.2.86	Yes	Yes	<2%	1	UT	SF

FUQ3	2_AV6	Figure B.2.87	Yes	Yes	<2%	1	W	SF
~~FUQ3~~	-	-	~~No~~	~~No~~	~~>2%~~	-	-	-
~~FUQ3~~	-	-	~~No~~	~~No~~	~~<2%~~	-	-	-
FUQ3	2_AV7	Figure B.2.88	No	Yes	>2%	1	W	SF
FUQ3	2_AV8	Figure B.2.89	No	Yes	>2%	1	W	SF

The selection of the FUQ1 and FUQ2 for round 2 based on their characteristics is shown in Table B.2.5.

Table B.2.5: Selection of 8 FUQ1 and FUQ2 based on their characteristics for round 2 iteration 1

Type	Readable ID	Screenshot	**Characteristics**				**Additional aspects**	
			Innovative	High cognitive effort	Percentage of associated users	Number of FUQ	Associated requirement type	App
FUQ2	2_FF1	Figure B.2.71	Yes	High	> 2%	1	SF	SV
FUQ2	2_FF2	Figure B.2.72	Yes	High	<2%	1	SF	SV
FUQ2	2_FF3	Figure B.2.73	Yes	Moderate/Low	>2%	1	SF	SF
FUQ1	2_FF4	Figure B.2.74	Yes	Moderate/Low	<2%	1	SF	SV
FUQ1	2_FF5	Figure B.2.75	No	High	>2%	1	SF	SV
FUQ1	2_FF6	Figure B.2.76	No	High	<2%	1	SF	SF
FUQ2	2_FF7	Figure B.2.77	No	Moderate/Low	>2%	1	SF	SV
FUQ1	2_FF8	Figure B.2.78	No	Moderate/Low	>2%	1	SF	SV

The selection of the FUQ1 and FUQ2 for round 2 iteration 1based on their additional aspects is shown in

Table B.2.6. There were no more FUQ1 or FUQ2 associated with User Tasks or Subtasks.

Table B.2.6: Selection of 3 additional FUQ1 and FUQ2 based on the additional aspects for round 2 iteration 1

Type	Readable ID	Screenshot	Characteristics				Additional aspects	
			Innovative	High cognitive effort	Percentage of associated users	Number of FUQ	Associated requirement type	App
FUQ2	2_FF9	Figure B.2.79	Yes	Moderate/Low	>2%	1	W	SF
FUQ1	2_FF10	Figure B.2.80	Yes	High	<2%	1	W	SV
FUQ2	2_FF11	Figure B.2.81	No	Moderate/Low	>2%	1	W	SF

B.3 Requirements

Figure B.3.1: Domain data diagram

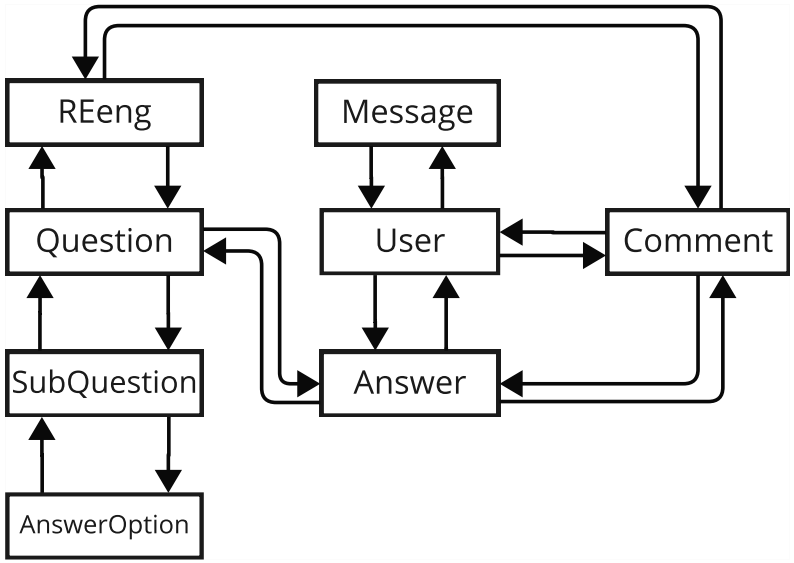


Table B.3.1: System functions (*SF*) for the user

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
Name: *SF: submitAnswer (U)* **Input:** *W: QuestionView (U)*					
The user can submit an answer for a question. Required fields of the answer are marked with a "*". These fields must be filled out.	A question exists that is not yet answered by the user	The answer is submitted to the database	After submission a notification appears indicating that the answer was submitted successfully. If there are no more questions to be answered or skipped, then a message appears indicating this and a confetti animation is played. If there are more questions, then the next question is shown.	(E1) The user discards the changes to the answer by navigating to another workspace. (E2) Required fields are not filled out.	(R1) All required fields of the answer must be filled out.
Name: *SF: skipAnswer (U)* **Input:** *W: QuestionView (U)*					
The user can skip an answer.	A question exists that is not yet answered by the user	The answer is marked as skipped in the database.	After submission a notification appears indicating that the answer was skipped. If there are no more questions to be answered or skipped, then a message appears indicating this and a confetti animation is played.	-	-
Name: SF: *submitMessage (U)* **Input:** *W: MessageView (U)*					
The user can submit a message. The message has a title, content and is associated with an app.	-	The message is saved in the database.	After submission a notification appears indicating that the message was submitted successfully.	(E1) The user discards the changes to the answer by navigating to another workspace. (E2) Required fields are not filled out.	(R1) The title of the message is required and it is required that the content of the message is not empty. The user must also choose an associated app or select "other".
Name: *SF: addAudioRecording (U)* **Input:** *W: MessageView (U), W: CommentView (U)*					
The user can add an audio recording to a message or a comment. The user presses on the	The user must allow the app to	The audio recording is added	After recording an audio recording, the audio recording appears in the message or comment and the user can play the	(E1) The user does not allow	(R1) The audio recording is stopped automatically when it reaches a length of two minutes.

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
record button and can stop the message by pressing to stop button.	use the microphone.	to the message or comment.	audio recording again or start recording a new one.	the app to use the microphone	
Name: *SF: submitComment (U)* **Input:** *W: CommentView (U)*					
The user can submit a comment for an answer or message. The comment can contain text and a audio recording (see *SF: addAudioRecording (U)*)	An answer or message exists.	The comment is saved in the database.	After submission a notification appears indicating that the comment was submitted successfully.	(E1) Required fields are not filled out.	(R1) The content of the comment must not be empty. There must be either an audio recording or text.
Name: *SF: remindUser (U)* **Input:** -					
The user is reminded to answer question when there are more than 5 questions to be answered on a Monday.	There are 5 questions open that need to be answered on a Monday. The user allowed the app to send push messages .	A push message is sent to the user reminding him to answer the questions.	The push message is displayed on the device of the user.	(E1) The user did not permit the app to send push messages to the device.	-
Name: *SF: displayQuestionsWithAnswers (U)* **Input:** *W: SentView (U)*					
The user can display all questions that have answers. The questions are shown with their title, app and their number of answers.	There is at least one question that has an answer.	-	The first five questions that have answers are shown.	(E1) There are no questions that have answers. If this is the case then a text appears indicating that the list is empty.	(R1) There are only 5 questions shown at one time. The user can load more questions with *SF: displayMoreQuestionsWithAnswers(U)* (R2) Questions with only skipped answers are not shown.
Name: *SF: displayMoreQuestionsWithAnswers (U)* **Input:** *W: SentView (U)*					
The user can display five more questions that have answers.	There are at least five questions that have an answer.	-	Five more questions are shown to the user. If there are less than five questions remaining, then these remaining questions are shown.		(R1) Questions with only skipped answers are not shown.
Name: *SF: displayAnswersForQuestion (U)* **Input:** *W: DetailedQuestionView (U)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The user can display all answers for a question. The answers are shown with their shortened content, their creation data and their number of comments.	There is at least one answer to the question.	-	The first five answers for the question are shown.	-	(R1) There are only 5 answers shown at one time. The user can load more answers with *SF: displayMoreAnswersForQuestion (U)* (R2) Skipped answers are not shown.
Name: *SF: displayMoreAnswersForQuestion (U)* **Input:** *W: DetailedQuestionView (U)*					
The user can display five more answers that for a question	There are at least five answeres that have an answer.	-	Five more answers are shown to the user. If there are less than five answers remaining, then these remaining answers are shown.		(R1) Skipped answers are not shown.
Name: *SF: displayMessages (U)* **Input:** *W: SentView (U)*					
The user can display all messages. The messages are shown with their title, app and their number of comments.	There is at least one message.	-	The first five messages are shown.	-	(R1) There are only 5 messages shown at one time. The user can load more messages with *SF: displayMoreMessages(U)*
Name: *SF: displayMoreMessages (U)* **Input:** *W: SentView (U)*					
The user can display five more messages.	There are at least five messages.	-	Five more messages are shown to the user. If there are less than five messages remaining, then these remaining messages are shown.	-	-
Name: *SF: displayComments (U)* **Input:** *W: DetailedAnswerView (U), W: DetailedMessageView (U)*					
The user can display the comments for an answer or message.	An answer or message exists.	-	The first five comments of the answer or message are shown.		(R1) There are only 5 comments shown at one time. The user can load more comments with *SF: displayMoreComments (U)*
Name: *SF: displayMoreComments (U)* **Input:** *W: DetailedAnswerView (U), W: DetailedMessageView (U)*					
The user can display five more comments.	There are more than five comments.	-	Five more comments are shown to the user. If there are less than five comments remaining, then these remaining comments are shown.	-	-
Name: *SF: sortComments (U)* **Input:** *W: DetailedAnswerView (U), W: DetailedMessageView (U)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The user can sort the comments for a question.	There exist at least one comment.	-	The comments are sorted by the selected criterium.	-	(R1) The comments can be sorted by these criteria: • Date of creation
Name: *SF: displayAnswerDetails (U)* **Input:** *W: DetailedAnswerView (U)*					
The user sees the content of the answer (app and all subquestions and their answers).	The answer was not skipped.	-	The content of the answer is shown.	-	-
Name: *SF: displayMessageDetails (U)* **Input** *W: DetailedMessageView (U)*					
The user sees the content (title and text or audio recording) of the message.	-	-	The content of the message is shown.	-	-
Name: *SF: sortQuestions (U)* **Input:** *W: SentView (U)*					
The user can sort the questions.	There exists at least one question that has an answer.	-	The questions are sorted by the selected criterium.	-	(R1) The questions can be sorted by these criteria: • Number of answers • Creation date of last answer • Creation date of last comment
Name: *SF: sortAnswersOfQuestion (U)* **Input:** *W: DetailedQuestionView (U)*					
The user can sort the answers for a question.	There exist at least one answer.	-	The answers for the question are sorted by the selected criterium.	-	(R1) The answers can be sorted by these criteria: • Date of creation
Name: *SF: sortMessages (U)* **Input:** *W: SentView (U)*					
The user can sort the messages.	There exists at least one message.	-	The messages are sorted by the selected criterium.	-	(R1) The messages can be sorted by these criteria: • Date of creation
Name: *SF: filterfMessagesOrQuestions (U)* **Input:** *W: SentView (U)*					
The user can filter the questions and the messages.	There exists at least one question that has an answer or one message.	-	The answers of the question and the messages are filtered by the selected criterium.	-	(R1) The questions and messages can be filtered by these criteria: • Creation date of last answer of question (Last week, this week, today) • App (SF, SI, SV, Other, Exercise)
Name: *SF: filterAnswersOfQuestion (U)* **Input:** *W(U): DetailedQuestionView (U)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The user can filter the answers of a question.	There exists at least one question that has an answer.	-	The questions are filtered by the selected criterium.	-	(R1) The answers can be filtered by these criteria: • Creation date of answer (Last week, this week, today) •
Name: *SF: filterOnlyMessagesOrQuestions (U)* **Input:** *W: SentView (U)*					
The user can decide whether to show only messages or only questions.	There exists at least one question that has an answer or a message.	-	Only the questions or the messages are shown based on the selected criterium.	-	(R1) The user can show only the messages or only the questions by choosing the corresponding type in the filter.
Name: *SF: navigateToAnswerDetailsInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can navigate to W: DetailedAnswerHistoryView (U).	There exist at least one answer.	-	The user is now in W: DetailedAnswerHistoryView (U).	-	-
Name: *SF: navigateToMessageDetailsInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can navigate to W: DetailedMessageHistoryView (U).	There exist at least one message.	-	The user is now in W: DetailedMessageHistoryView (U).	-	-
Name: *SF: displayAnswersInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can display all answers. The answers are shown with their question title, their creation date and information whether they were answered or skipped.	There is at least one answer.	-	The latest five answers are shown.	-	(R1) There are only 5 answers shown at one time. The user can load more answers with *SF: displayMoreAnswersInHistory (U)* (R2) Skipped answers are also shown.
Name: *SF: displayMoreAnswersInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can display five more answers.	There are at least five answers.	-	Five more answers are shown to the user. If there are less than five answers remaining, then these remaining answers are shown.		(R1) Skipped answers are also shown.
Name: *SF: displayMessagesInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can display all messages. The messages are shown with their title and their creation date.	There is at least one message.	-	The latest five messages are shown.	-	(R1) There are only 5 messages shown at one time. The user can load more answers with *SF: displayMoreMessagesInHistory (U)*

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
Name: *SF: displayMoreMessagesInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can display five more messages.	There are at least five messages.	-	Five more messages are shown to the user. If there are less than five messages remaining, then these remaining messages are shown.	-	-
Name: *SF: filterAnswersAndMessagesInHistory (U)* **Input:** *W: HistoryView (U)*					
The user can filter the answers and the messages.	There exists at least one answer or one message.	-	The answers and the messages are filtered by the selected criterium.	-	(R1) The questions and messages can be filtered by these criteria: • Creation date (Custom range, today)
Name: *SF: displayInstructions (U)* **Input:** *W: InstructionsView (U)*					
The instructions for the application are displayed automatically.	-	-	The instructions are shown.	-	-
Name: *SF: navigateToQuestionView (U)* **Input:** *W: SidebarView (U)*					
The user can navigate to the question view.	-	-	The question view is shown.	-	-
Name: *SF: navigateToMessageView (U)* **Input:** *W: SidebarView (U)*					
The user can navigate to the message view.	-	-	The message view is shown.	-	-
Name: *SF: navigateToSentView (U)* **Input:** *W: SidebarView (U)*					
The user can navigate to the sent view.	-	-	The sent view is shown.	-	-
Name: *SF: navigateToHistoryView (U)* **Input:** *W: SidebarView (U)*					
The user can navigate to the history view.	-	-	The history view is shown.	-	-
Name: *SF: navigateToInstructionView (U)* **Input:** *W: SidebarView (U)*					
The user can navigate to the instruction view.	-	-	The instruction view is shown.	-	-

Table B.3.2: System functions for the REeng

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
Name: *SF: displayQuestions (R)* **Input***W: QuestionView (R)*					
The REeng can display the first ten questions. The ID, the question title and the status whether or not the question is enabled or disabled are shown-	-	-	The first ten questions are shown.	-	(R1) The first ten questions are shown initially. The Reeng can display more questions through *SF: displayMoreQuestions (R)*
Name: *SF: displayMoreQuestions (R)* **Input:** *W: QuestionView (R)*					
The REeng can display ten more questions.	There are more than ten questions that have an answer.	-	Ten more questions are shown to the user. If there are less than ten questions remaining, then these remaining questions are shown.	-	-
Name: *SF: sortQuestions (R)* **Input:** *W: QuestionView (R)*					
The REeng can sort the questions.	There is at least one questions.	-	The questions are sorted by the selected criterium.	-	The questions can be sorted by these criteria: • ID • Question name • Disabled/Enabled
Name: *SF: searchQuestions (R)* **Input:** W*: QuestionView (R)*					
The REeng can search questions by text. If the text matches a part of a subquestion, then the corresponding question(s) is shown.	-	-	The question(s) that contain the text in at least one of their subquestion are shown.	-	-
Name: *SF: enableQuestion (R)* **Input:** W*: QuestionView (R)*					
The REeng can enable a question.	The question must exist and be disabled.	-	The question is enabled.	-	-
Name: *SF: disableQuestion (R)* **Input:** W*: QuestionView (R)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The REeng can disable a question.	The question must exist and be enabled.	-	The question is disabled.	-	-
Name: *SF: createQuestion (R)* **Input:** *W: CreateQuestionView (R)*					
The REeng can create a question. The REeng specifies the question title, the app, as well as all subquestions and answer options	-	-	The question is created.	(E1) The question is not shown to the user	(R1) The question must contain at least one subquestions and corresponding answer options.
Name: *SF: editQuestion (R)* **Input:** *CreateQuestionView (R)*					
The REeng can edit the question and its properties (title, app, all subquestions and answer options).	The question must exist.	-	The edited properties are saved.	(E1) The question is not shown to the user	(R1) The question must contain at least one subquestions and corresponding answer options.
Name: *SF: scheduleQuestion (R)***Input:** *W: ScheduleQuestionView (R)*					
The REeng can schedule a question to be enabled and disabled.	The question must exist.	-	The question is now scheduled based on the configured conditions.	-	(R1) The question can be scheduled based on these conditions: • Enable question at specific days. If the question is not answered it will not be asked multiple times when multiple days are entered. • Disable question at specific date
Name: *SF: displayAnswersForQuestion (R)* **Input:** *W: DetailedQuestionView (R)*					
The REeng can display all answers for a question. The answers are shown with their shortened content, their creation data and their number of comments.	There is at least one answer to the question.	-	The first five answers for the question are shown.	-	(R1) There are only 5 answers shown at one time. The REeng can load more answers with *SF: displayMoreAnswersForQuestion (R)* (R2) Skipped answers are not shown.
Name: *SF: displayAnswerDetails (R)* **Input:** *W: DetailedQuestionView (R), W: DetailedAnswerView (R)*					
The REeng sees the content of the answer (app and all subquestions and their answers).	The answer was not skipped.	-	The content of the answer is shown.	-	-
Name: *SF: sortQuestionsWithAnswers (R)* **Input:** *W: ResultsView (R)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The REeng can sort the questions.	There exists at least one question that has an answer.	-	The questions are sorted by the selected criterium.	-	(R1) The questions can be sorted by these criteria: • Number of answers • Creation date of last answer • Creation date of last comment
Name: *SF: filterQuestionsAndAnswers (R)* **Input***W: ResultsView (R)*					
The REeng can filter the questions and messages.	There exists at least one question that has an answer or one message.	-	The questions and messages are filtered by the selected criterium.	-	(R1) The questions and messages can be filtered by these criteria: • Creation date of last answer of question or creation date of message (Last week, this week, today) • App (SF, SI, SV, Other, Exercise)
Name: *SF: sortAnswers (R)* **Input:** *W: DetailedQuestionView (R)*					
The REeng can sort the answers for a question.	There exists at least one answer.	-	The answers for the question are sorted by the selected criterium.	-	(R1) The answers can be sorted by these criteria: • Date of creation • Creation date of last comment
Name: *SF: filterAnswers (R)* **Input:** *W: DetailedQuestionView (R)*					
The REeng can filter the answers of a question .	There exists at least one question that has an answer.	-	The answers are filtered by the selected criterium.	-	(R1) The answers can be filtered by whether or not they were already commented by the REeng.
Name: *SF: displayMessages (R)* **Input:** *W: ResultsView (R)*					
The REeng can display all messages of all users. The messages are shown with their title, app and their number of comments.	There is at least one message.	-	The first five messages are shown.	-	(R1) There are only 5 messages shown at one time. The REeng can load more messages with *SF: displayMoreMessages(R)*
Name: *SF: displayMessageDetails (R)* **Input:** *W: DetailedMessageView (R)*					
The REeng sees the content (title and text or audio recording) of the message.	-	-	The content of the mesage is shown.	-	-
Name: *SF: sortMessages (R)* **Input:** *W: ResultsView (R)*					
The REeng can sort the messages.	There exists at least one message.	-	The messages are sorted by the selected criterium.	-	(R1) The messages can be sorted by these criteria: • Date of creation

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
					• Creation date of last comment
Name: *SF: filterMessages (R)* **Input:** *W: ResultsView (R)*					
The REeng can filter the messages.	There exists at least one message.	-	The messages are filtered by the selected criterium.	-	(R1) The messages can be filtered by these criteria: • Creation date of message (Last week, this week, today) • App (SF, SI, SV, Other, Exercise) • Commented by the REeng (Yes/No)
Name: *SF: displayQuestionsWithAnswers (R)* **Input:** *W: ResultsView (R)*					
The REeng can display all questions that have answers. The questions are shown with their title, app and their number of answers.	There is at least one question that has an answer.	-	The first five questions that have answers are shown.	(E1) There are no questions that have answers. If this is the case then a text appears indicating that the list is empty.	(R1) There are only 5 questions shown at one time. The REeng can load more questions with *SF: displayMoreQuestionsWithAnswers(R)* (R2) Questions with only skipped answers are not shown.
Name: *SF: displayMoreQuestionsWithAnswers (R)* **Input:** *W: ResultsView (R)*					
The REeng can display five more questions that have answers.	There are at least five questions that have an answer.	-	Five more questions are shown. If there are less than five questions remaining, then these remaining questions are shown.		(R1) Questions with only skipped answers are not shown.
Name: *SF: displayMoreAnswersForQuestion (R)* **Input:** *W: DetailedQuestionView (R)*					
The REeng can display five more answers that for a question	There are at least five answers that have an answer.	-	Five more answers are shown. If there are less than five answers remaining, then these remaining answers are shown.		(R1) Skipped answers are not shown.
Name: *SF: displayMoreMessages (R)* **Input:** *W: ResultsView (R)*					
The REeng can display five more messages.	There are at least five messages that have an answer.	-	Five more messages are shown. If there are less than five messages remaining, then these remaining messages are shown.	-	-

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
Name: *SF: displayComments (R)* **Input:** *W: DetailedAnswereView (R), W: DetailedMessageView (R)*					
The REeng can display the comments for an answer or message.	An answer or message exists.	-	The first five comments of the answer or message are shown.		(R1) There are only 5 comments shown at one time. The REeng can load more comments with *SF: displayMoreComments (R)*
Name: *SF: displayMoreComments (R)* **Input:** *W: DetailedAnswereView (R), W: DetailedMessageView (R)*					
The REeng can display five more comments.	There are more than five comments.	-	Five more comments are shown. If there are less than five comments remaining, then these remaining comments are shown.	-	-
Name: *SF: sortComments (R)* **Input:** *W: DetailedAnswerView (R), W: DetailedMessageView (R)*					
The REeng can sort the comments for a message or answer.	There exist at least one comment.	-	The comments are sorted by the selected criterium.	-	(R1) The comments can be sorted by these criteria: • Date of creation
Name: *SF: submitComment(R)* **Input:** *W: CommentView (R)*					
The REeng can submit a comment for an answer or message. The comment can contain text and a audio recording (see *SF: addAudioRecording (U)*)	An answer or message exists.	The comment is saved in the database.	After submission a notification appears indicating that the comment was submitted successfully.	(E1) Required fields are not filled out.	(R1) The content of the comment must not be empty. There must be either an audio recording or text.
Name: *SF: displayUsers (U)* **Input:** *W: UserView (R)*					
The REeng can view all users that use smartFEEDBACK.	-	-	-	-	The users are shown with their ID and the date when they first started to use smartFEEDBACK.
Name: *SF: filterUsers (R)* **Input:** *W: UserView (R)*					
The REeng can filter users by their start date.	There must be at least one user.	-	Only users are shown who have a start date in the specified range or today.	-	The REeng can input a range for the start date or he/she can filter users which start date is today.
Name: *SF: navigateToQuestionView (R)* **Input:** *W: SidebarView (R)*					
The REeng can navigate to the W: QuestionView (R).	-	-	W: QuestionView (R) is shown.	-	-
Name: *SF: navigateToResultsView (R)* **Input:** *W: SidebarView (R)*					

Description	Pre-condition(s)	Postcondition(s)	Output	Exception(s)	Rule(s)
The REeng can navigate to W: ResultsView (R)	-	-	W: ResultsView (R) is shown.	-	-
Name: *SF: navigateToUsersView (R)* **Input:** *W: SidebarView (R)*					
The REeng can navigate to W: QuestionView (R)	-	-	W: QuestionView (R)is shown.	-	-

Virtual windows for the user

Figure B.2.1: Virtual window for W: QuestionView (U)

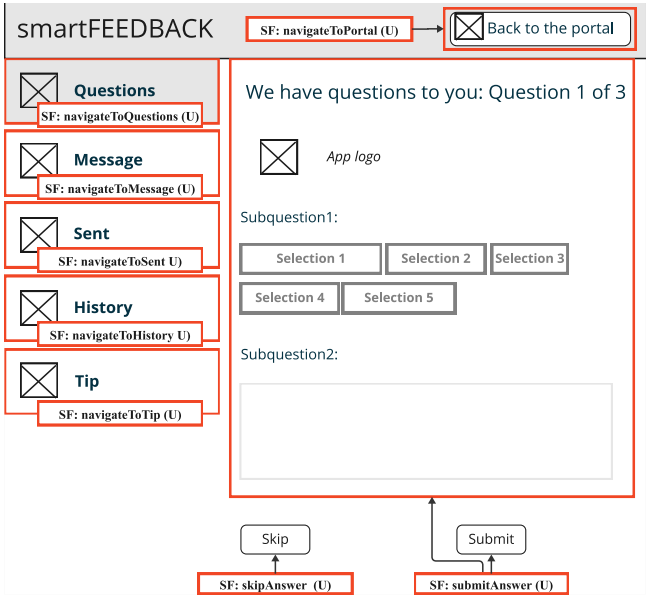


Figure B.2.2: Virtual window for W: MessageView (U)

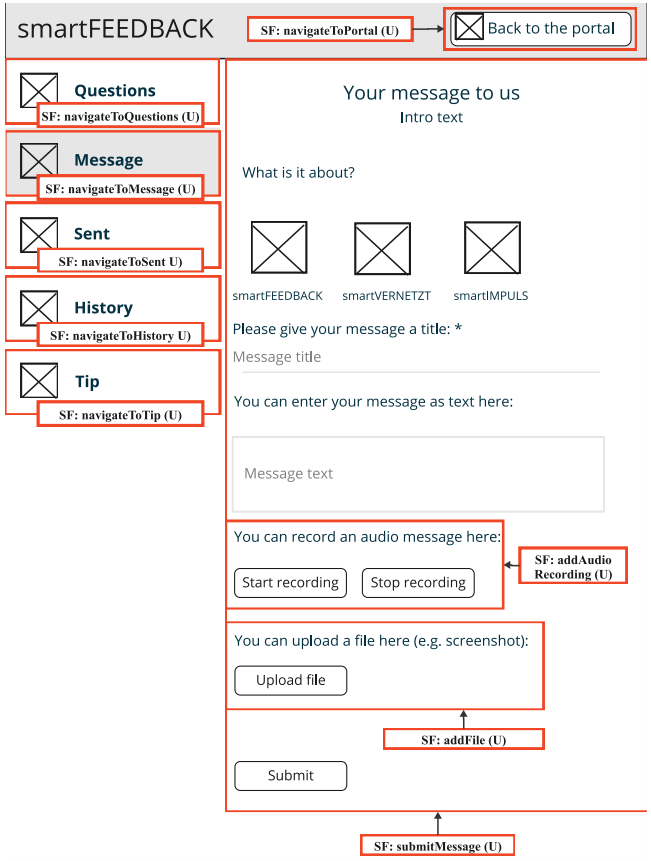


Figure B.2.3: Virtual window for W: SentView (U)

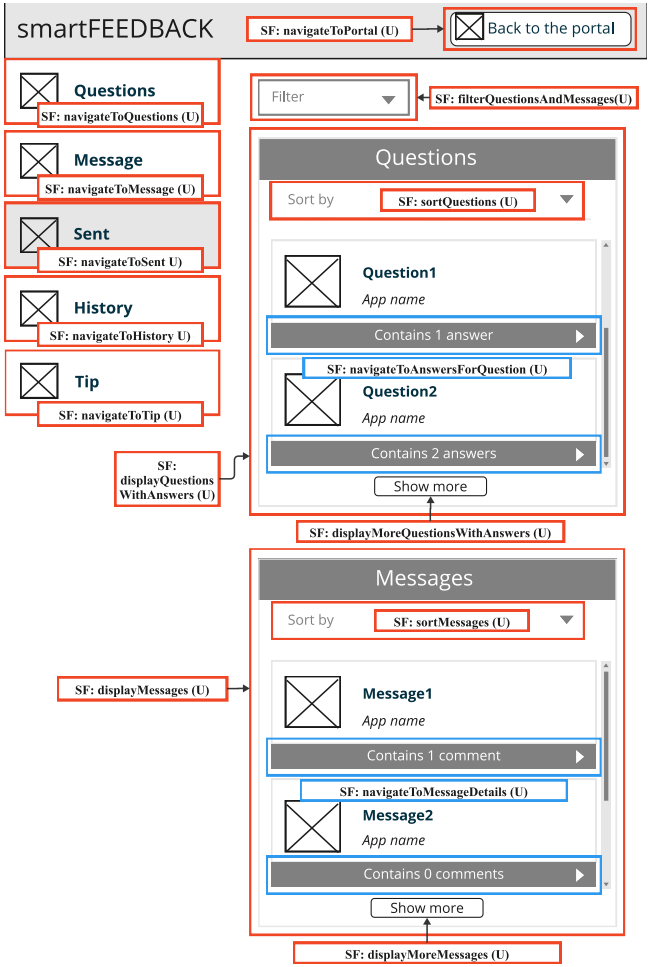


Figure B.2.4: Virtual window for W: DetailedQuestionView (U)

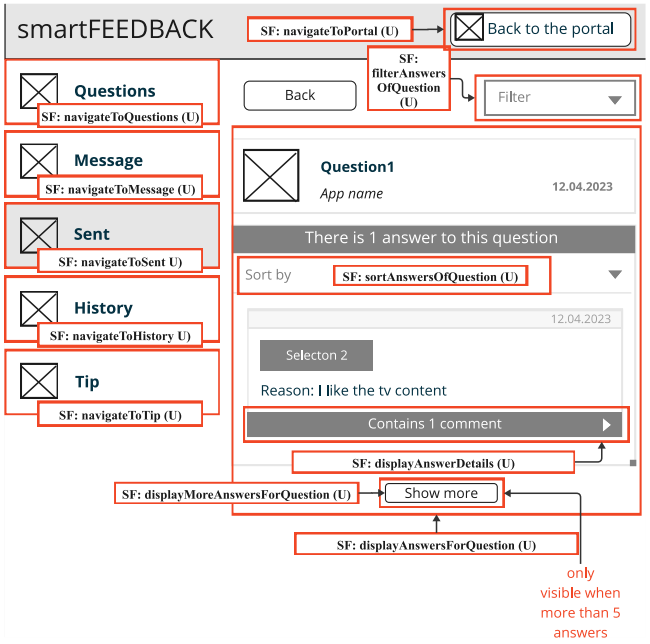


Figure B.2.5: Virtual window for W: DetailedAnwerView (U)

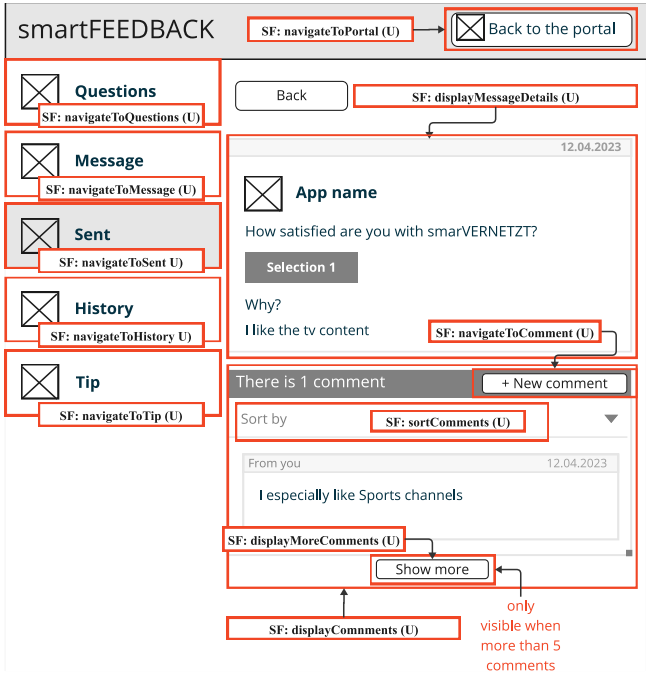


Figure B.2.6: Virtual window for W: DetailedMessageView (U)

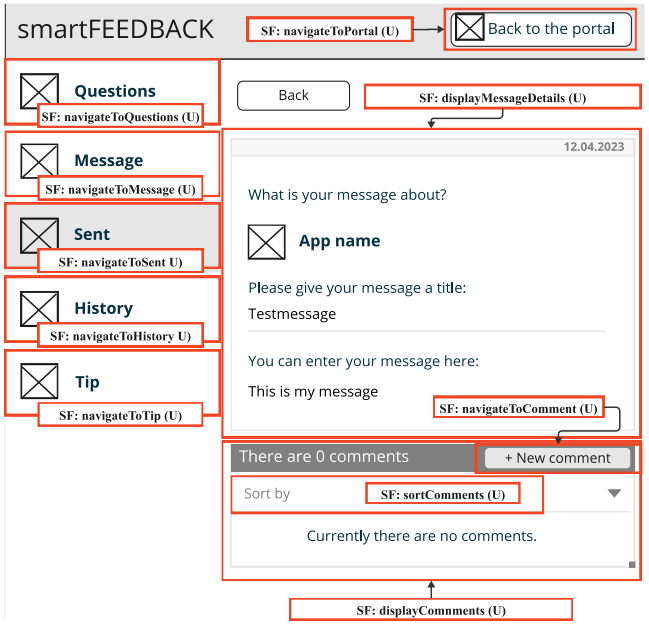


Figure B.2.7: Virtual window for
W: CommentView (U)

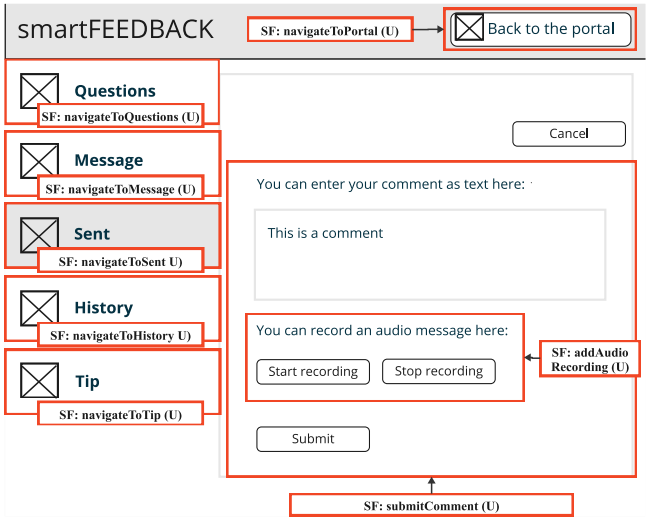


Figure B.2.8: Virtual window for
W: HistoryView (U)

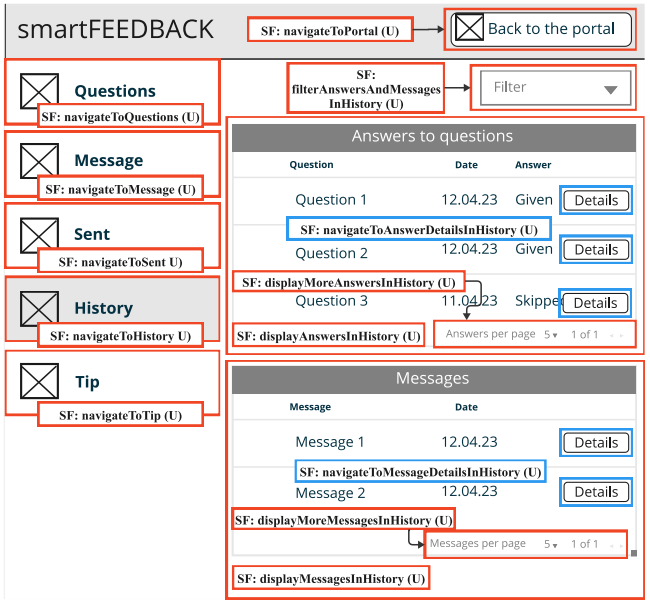


Figure B.2.9: Virtual window for
W: DetailedAnswerHistoryView (U)

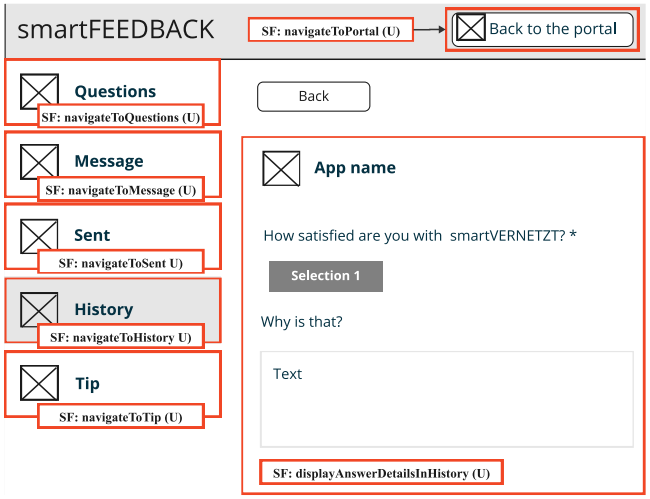


Figure B.2.10: Virtual window for
W: DetailedMessageHistoryView (U)

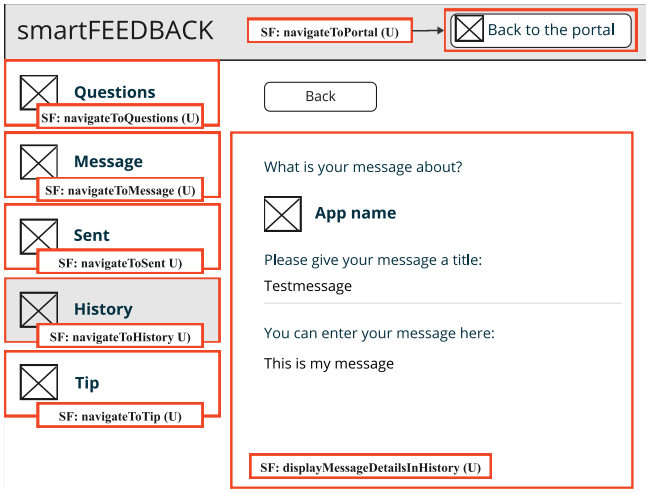
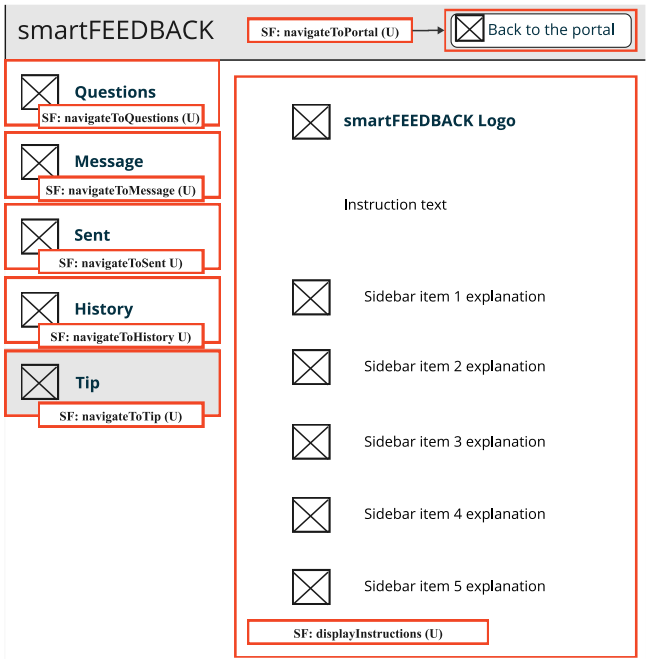


Figure B.2.11: Virtual window for
W: InstructionView (U)



Virtual windows for the REeng

Figure B.2.12: Virtual window for W: QuestionView (R)

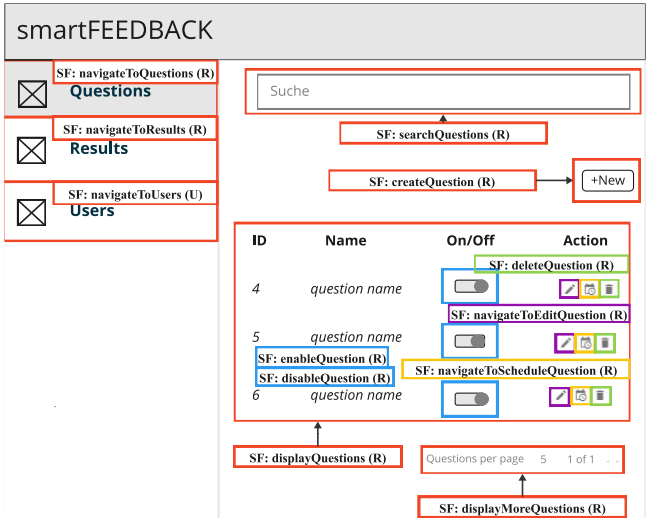


Figure B.2.13: Virtual window for W: CreateQuestionView (R)

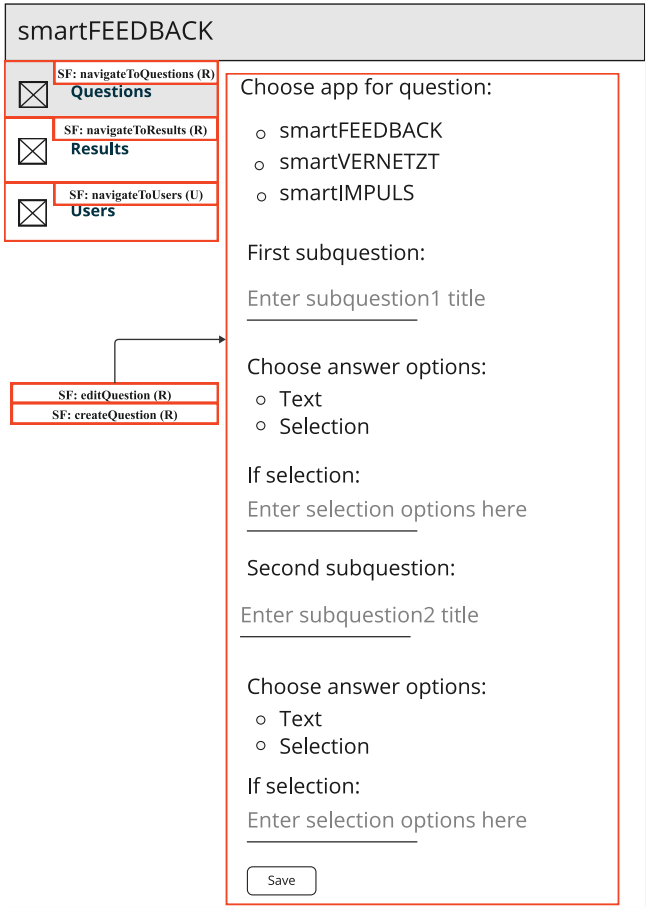


Figure B.2.14: Virtual window for W: ScheduleView (R)

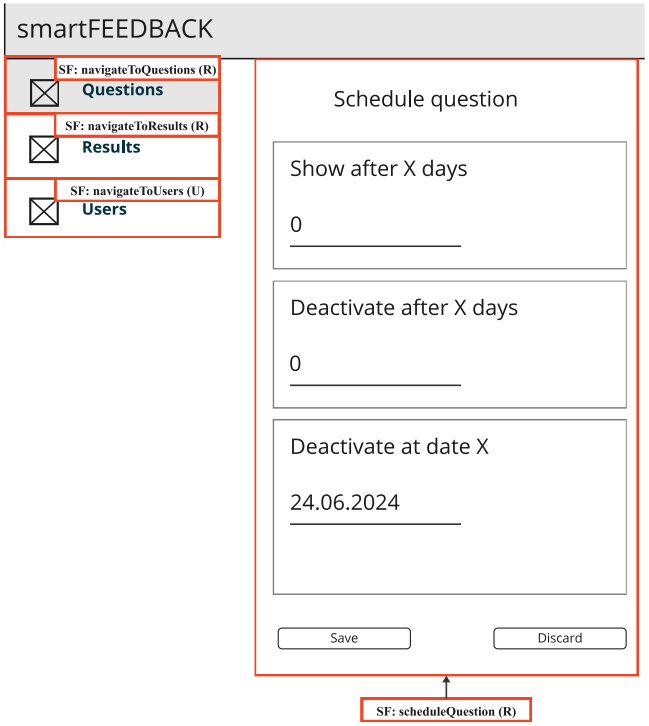


Figure B.2.15: Virtual window for W: ResultsView (R)

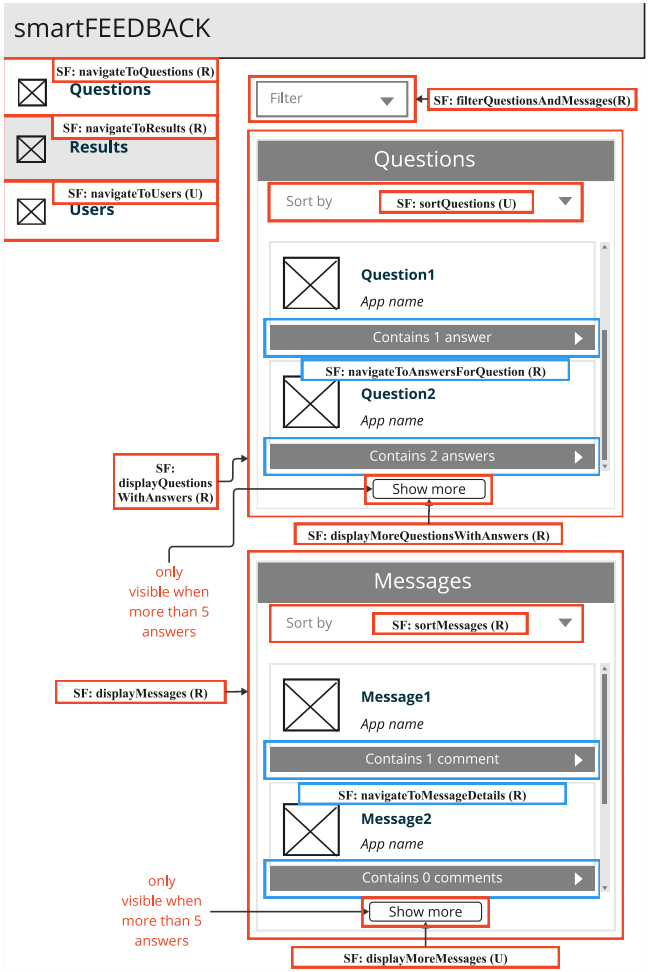


Figure B.2.16: Virtual window for W: DetailedQuestionView (R)

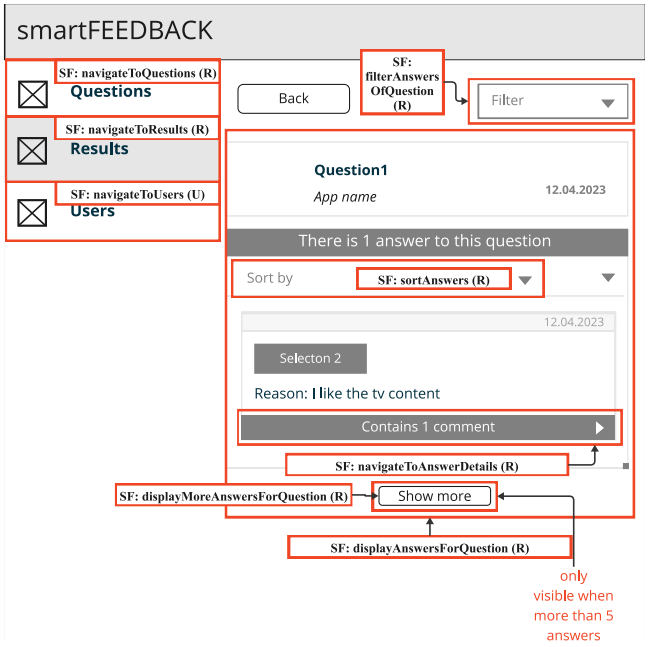


Figure B.2.17: Virtual window for W: DetailedAnswerView (R)

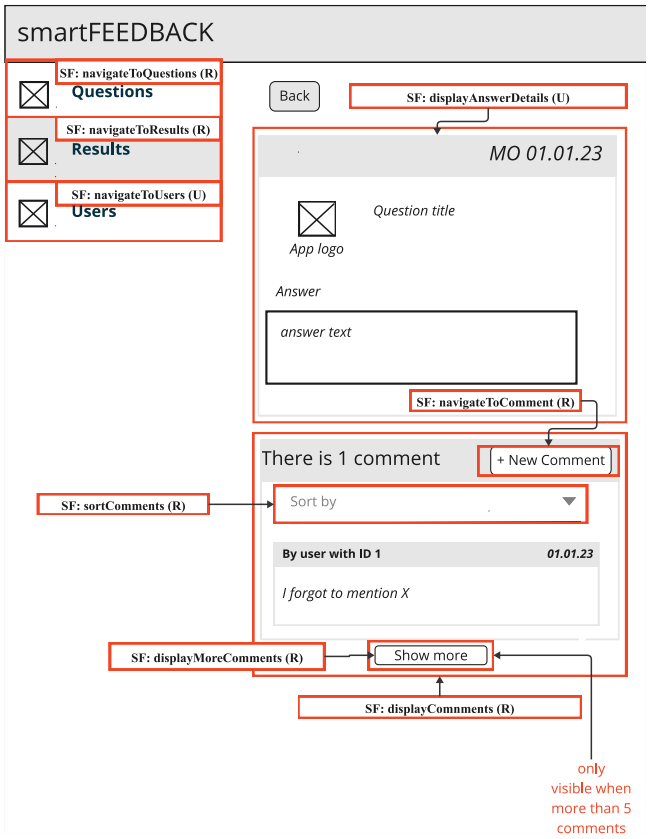


Figure B.2.18: Virtual window for W: DetailedMessageView (R)

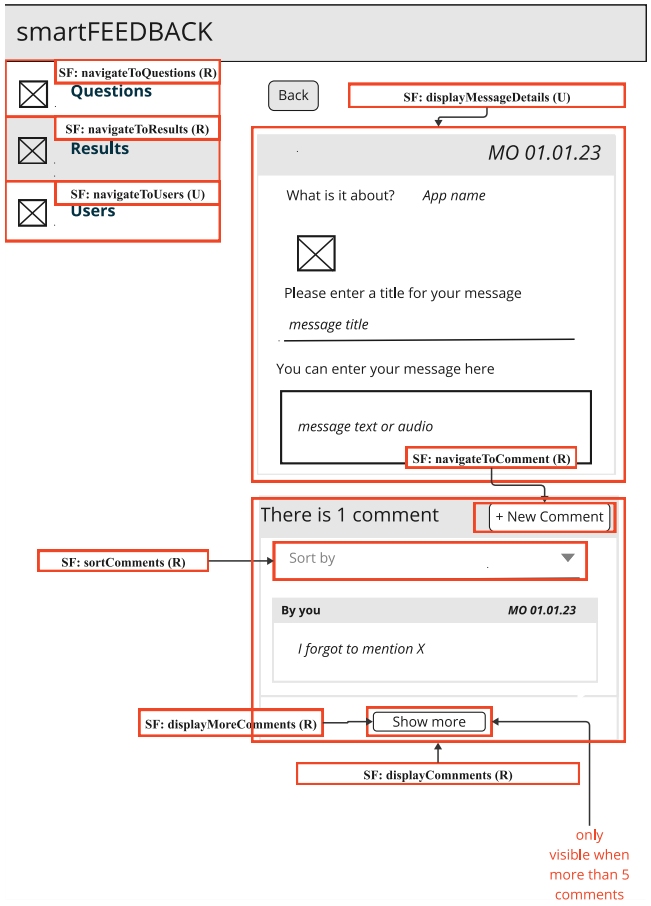


Figure B.2.19: Virtual window for W: CommentView (R)

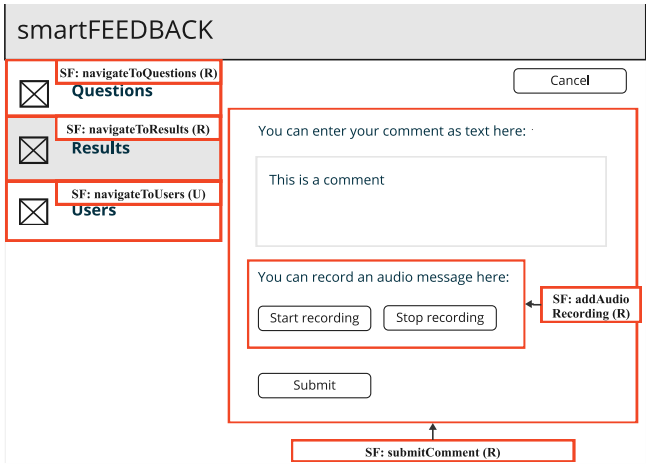
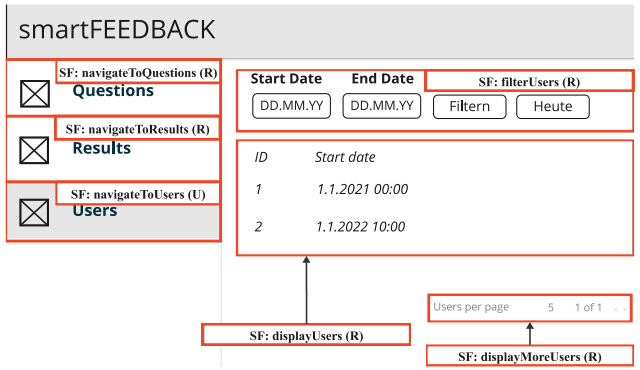


Figure B.2.20: Virtual window for W: UserView (R)



Screenshots for the user

Figure B.2.21: Screenshot for implemented virtual window of W: QuestionView (U)

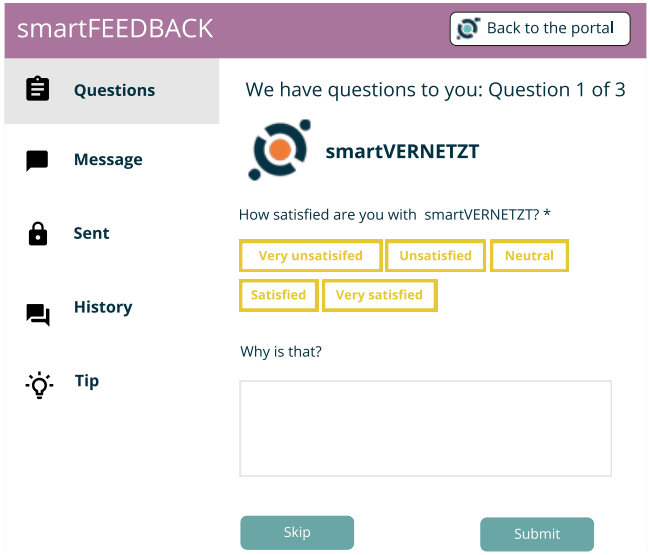


Figure B.2.22: Screenshot for implemented virtual window of W: MessageView (U)

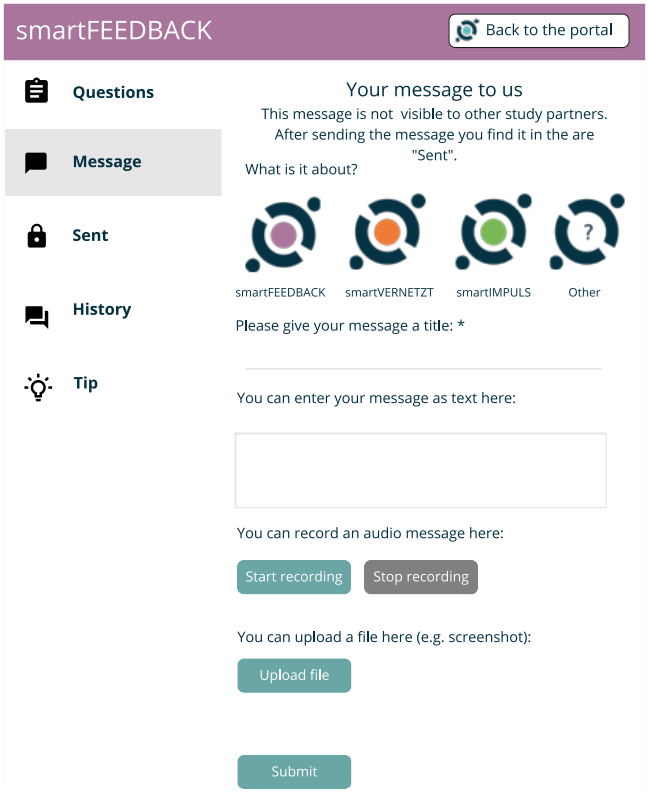


Figure B.2.23: Screenshot for implemented virtual window of W: SentView (U)

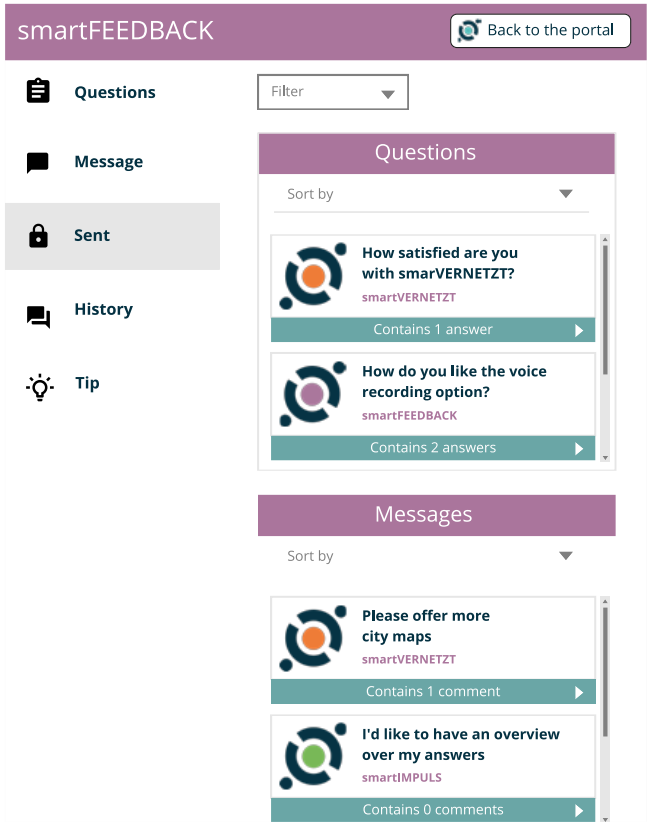


Figure B.2.24: Screenshot for implemented virtual window of: DetailedQuestionView (U)

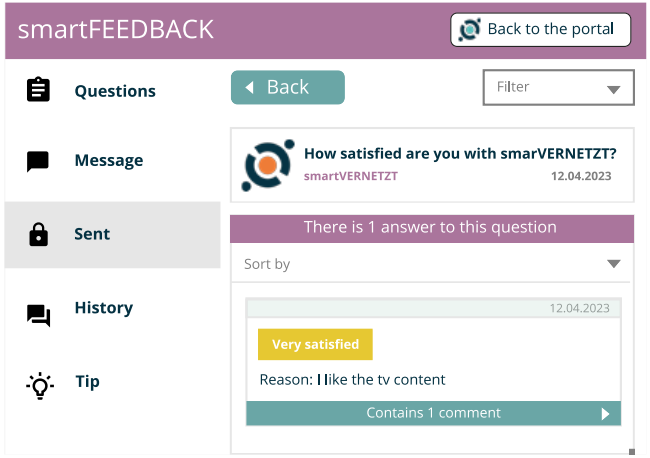


Figure B.2.25: Screenshot for implemented virtual window of: DetailedAnswerView (U)

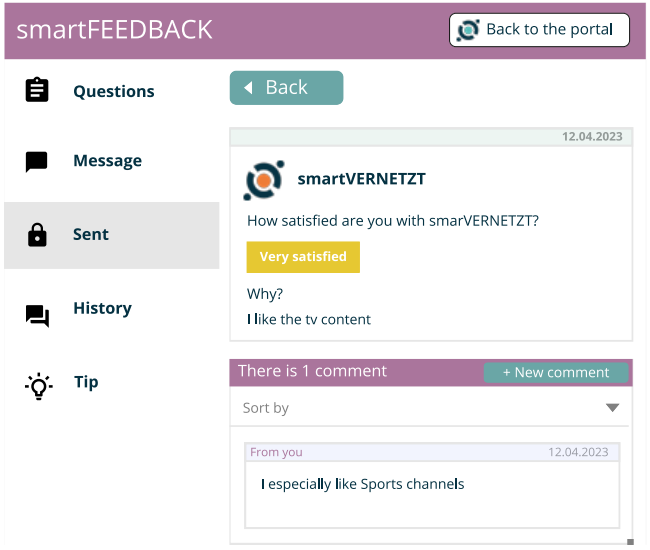


Figure B.2.26: Screenshot for implemented virtual window of W: DetailedMessageView (U)

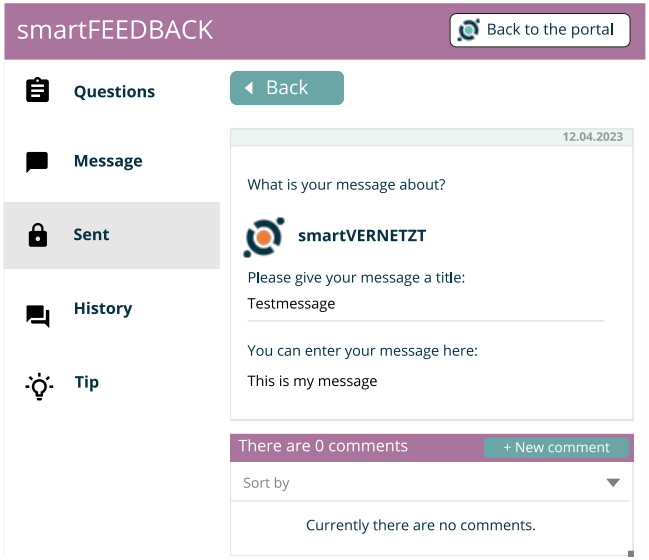


Figure B.2.27: Screenshot for implemented virtual window of W: CommentView (U)

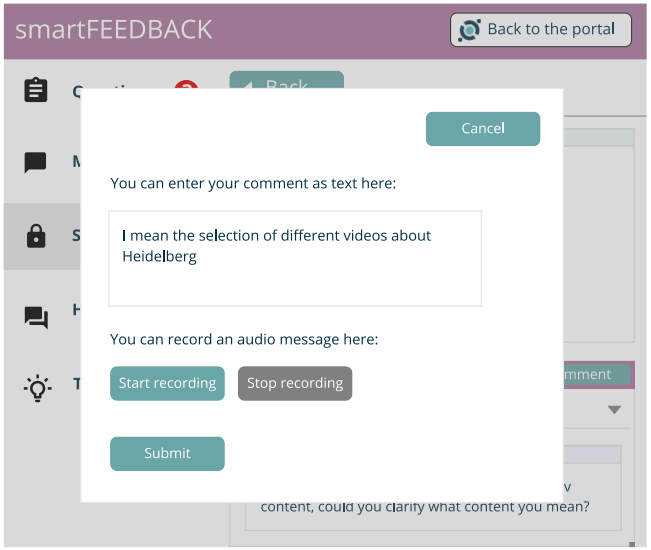


Figure B.2.28: Screenshot for implemented virtual window of W: HistoryView (U)

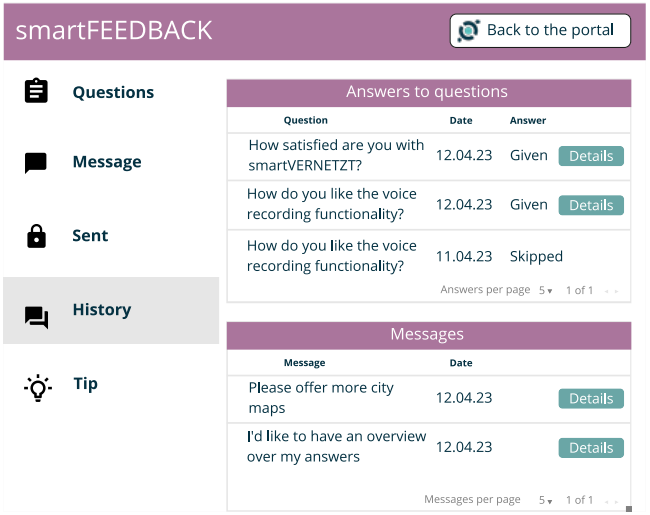


Figure B.2.29: Screenshot for implemented virtual window of W: DetailedAnswerHistoryView (U)

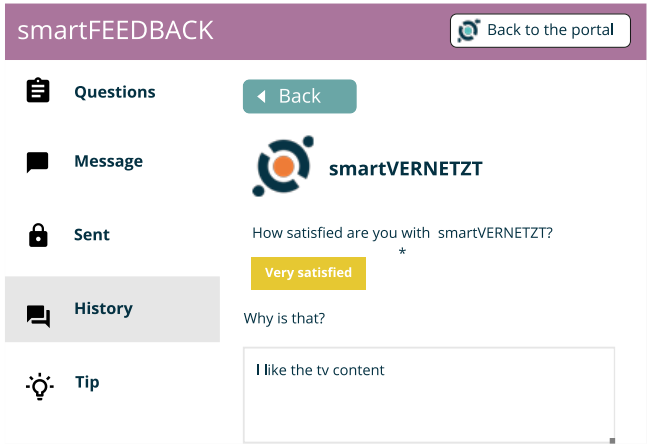


Figure B.2.30: Screenshot for implemented virtual window of W: DetailedMessageHistoryView (U)

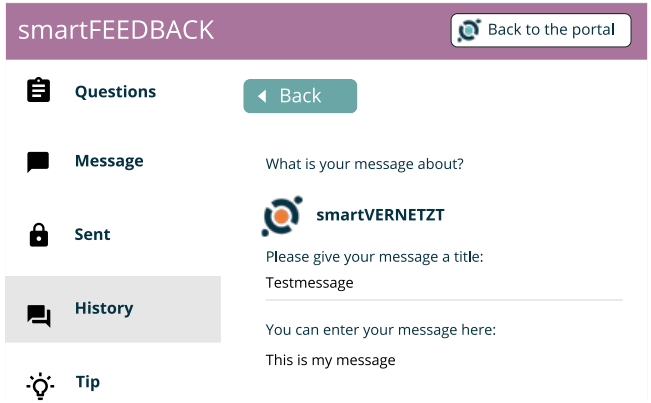
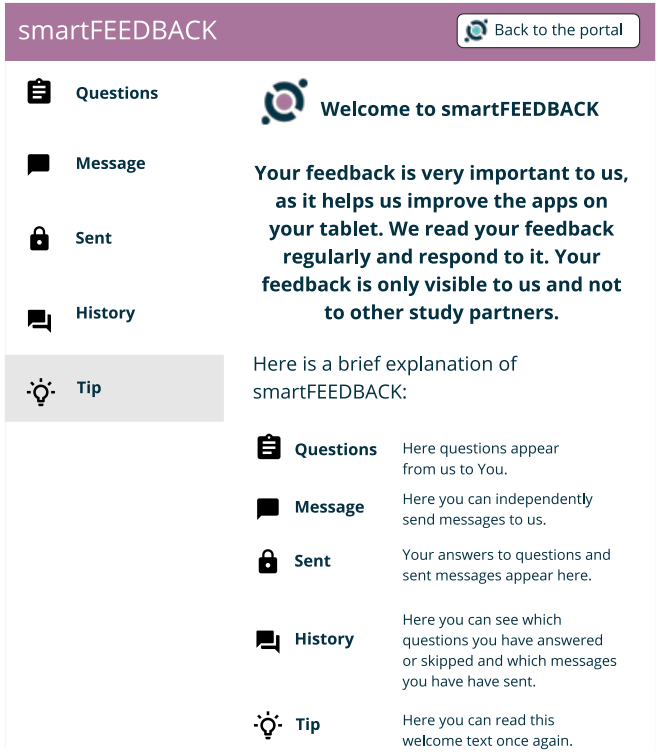


Figure B.2.31: Screenshot for implemented virtual window of W: InstructionView (U)



Screenshots for the REeng

Figure B.2.32: Screenshot for implemented virtual window of W: QuestionView (R)

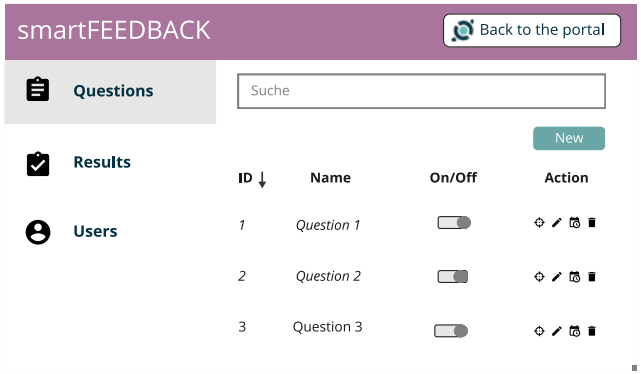


Figure B.2.33: Screenshot for implemented virtual window of W: CreateQuestionView (R)

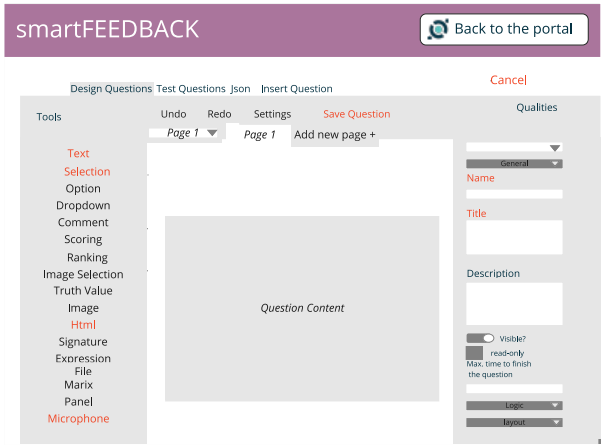


Figure B.2.34: Screenshot for implemented virtual window of W: ScheduleQuestionView (R)

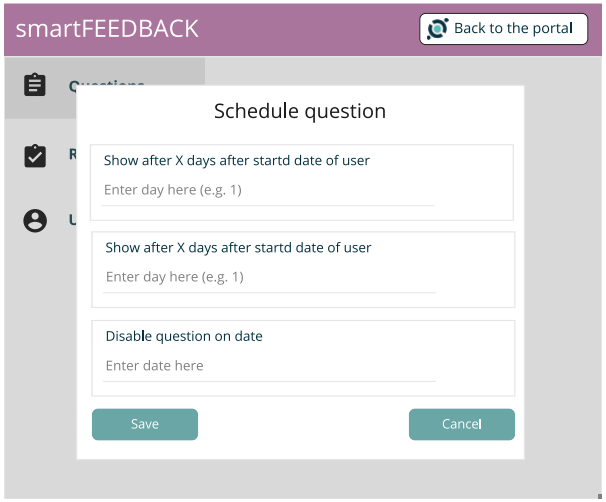


Figure B.2.35: Screenshot for implemented virtual window of W: ResultsView (R)

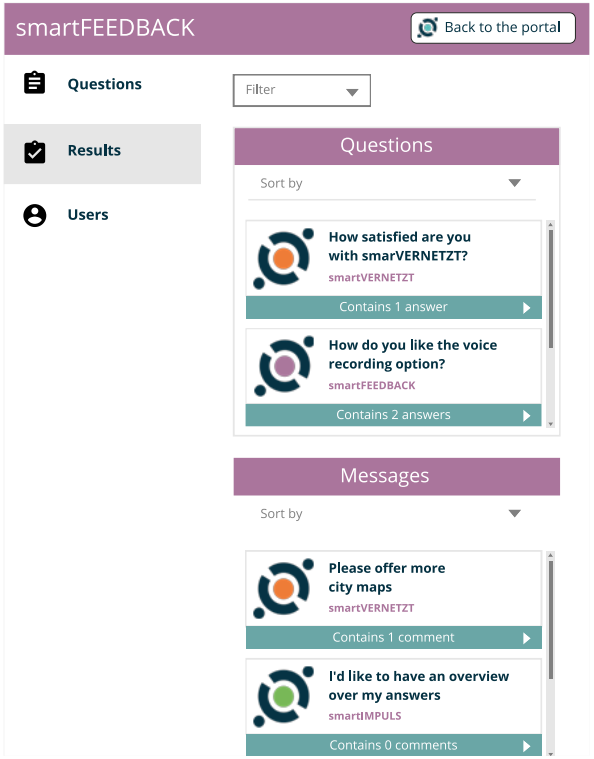


Figure B.2.36: Screenshot for implemented virtual window of W: DetailedQuestionView (R)

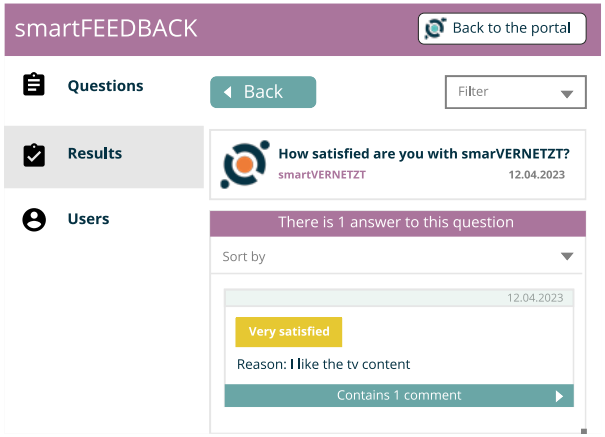


Figure B.2.37: Screenshot for implemented virtual window of W: DetailedAnswerView (R)

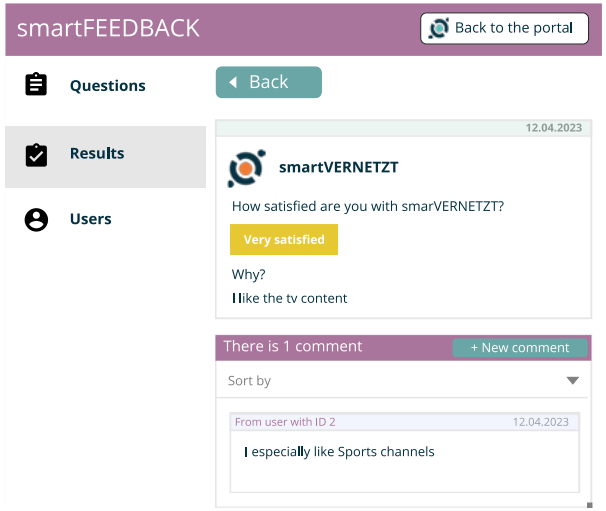


Figure B.2.38: Screenshot for implemented virtual window of W: DetailedMessageView (R)

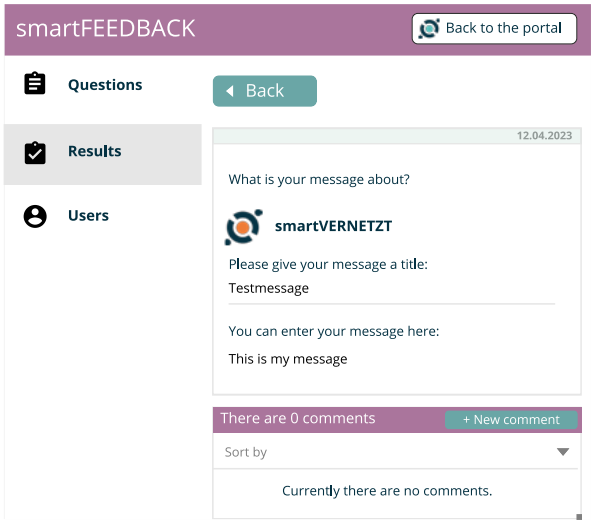


Figure B.2.39: Screenshot for implemented virtual window of W: CommentView (R)

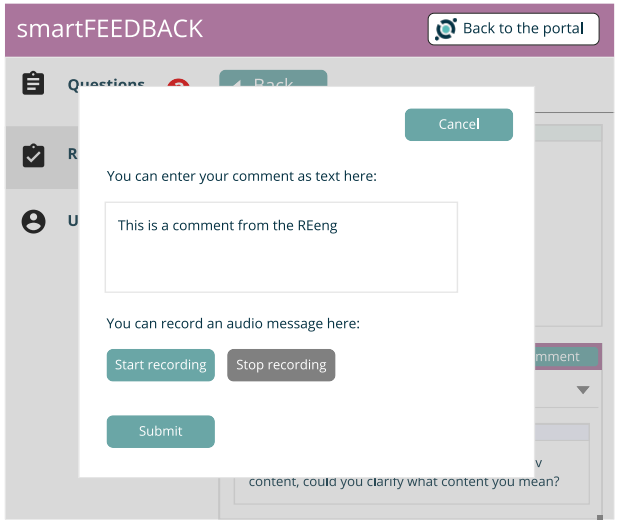
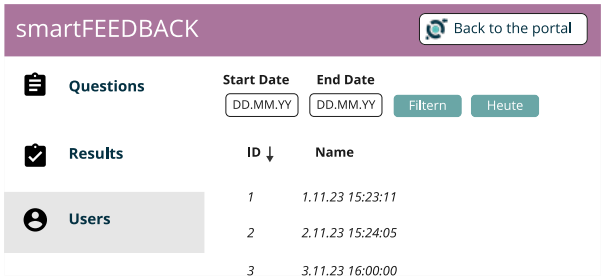


Figure B.2.40: Screenshot for implemented virtual window of W: UserView (R)



B.4 FUQ

Round 1 iteration 1

Figure B.2.41: FUQ with id FF1

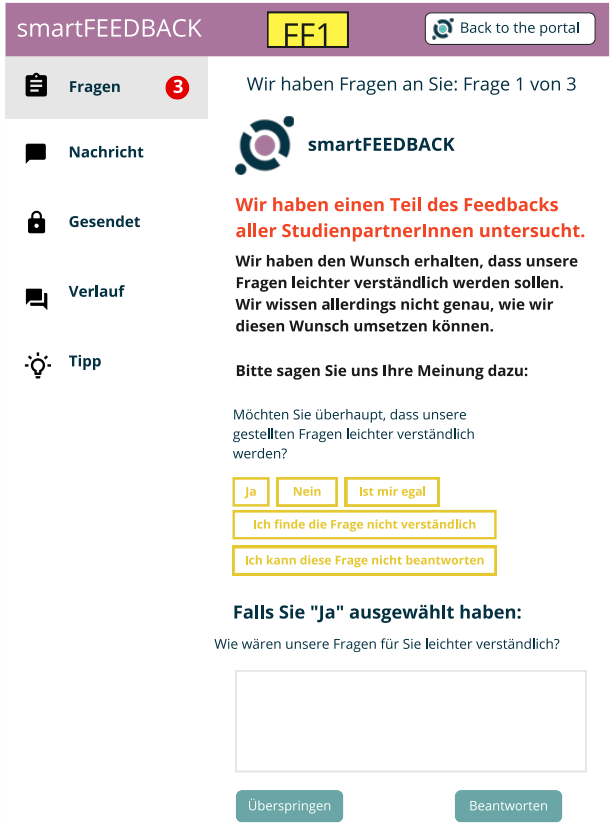


Figure B.2.42: FUQ with id FF2

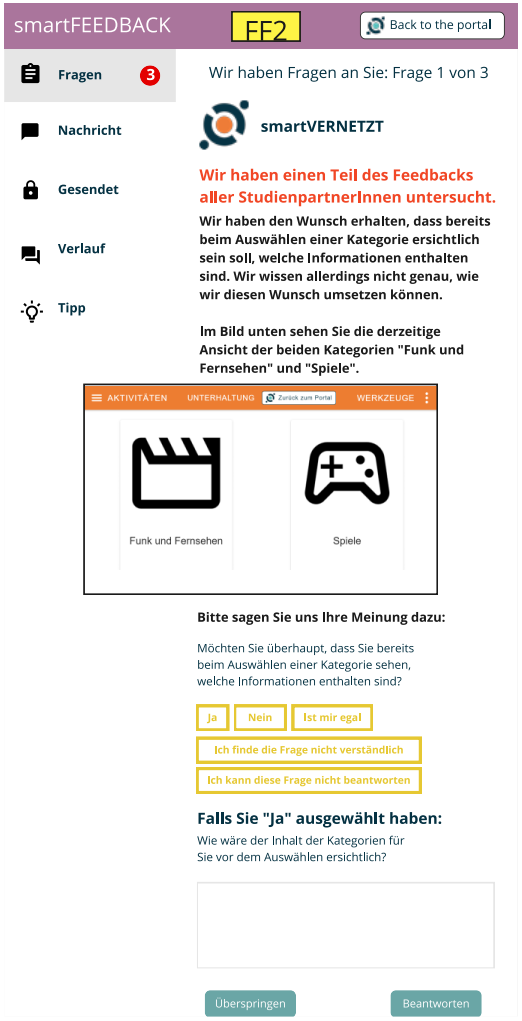


Figure B.2.43: FUQ with id FF3

Figure B.2.44: FUQ with id FF4

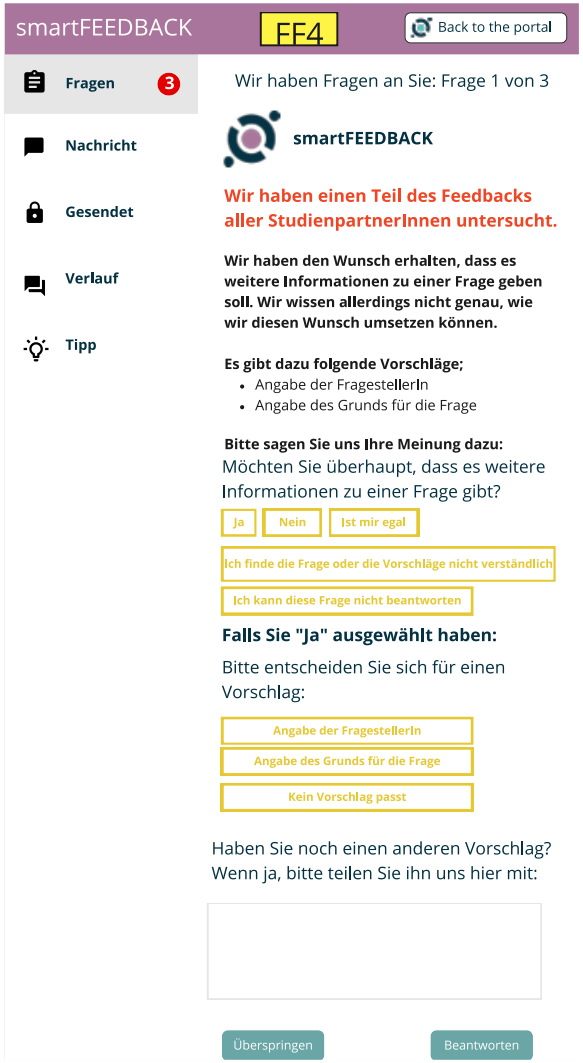


Figure B.2.45: FUQ with id FF5

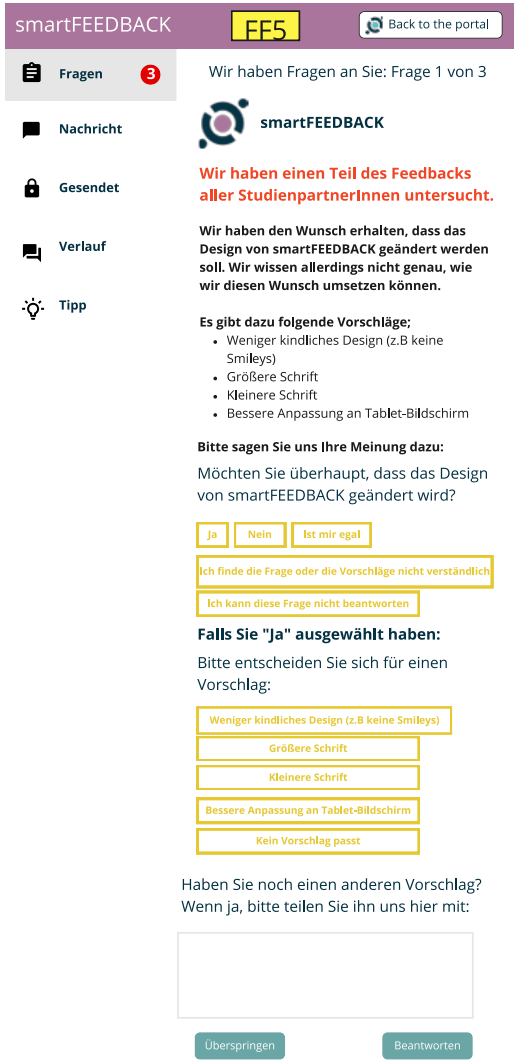


Figure B.2.46: FUQ with id FF6

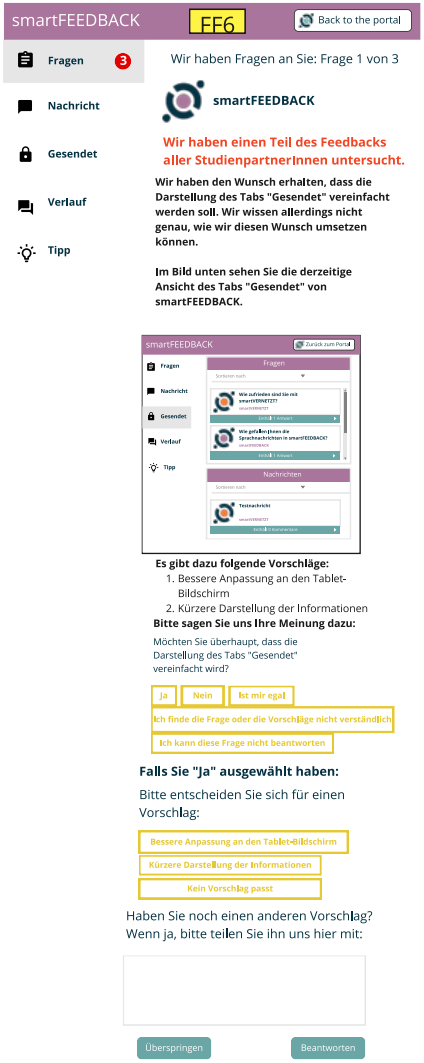


Figure B.2.47: FUQ with id FF7

Figure B.2.48: FUQ with id FF8

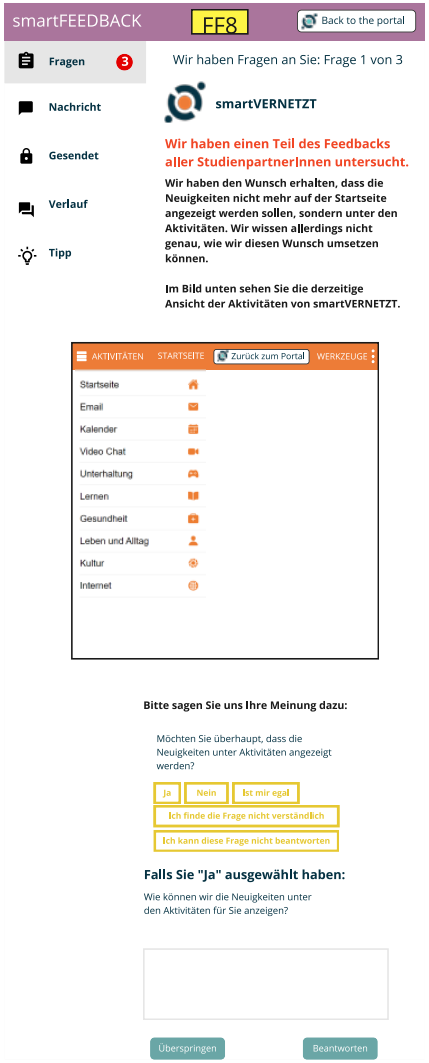


Figure B.2.49: FUQ with id FF9

Figure B.2.50: FUQ with id FF10

Figure B.2.51: FUQ with id FF11

Figure B.2.52: FUQ with id AV1

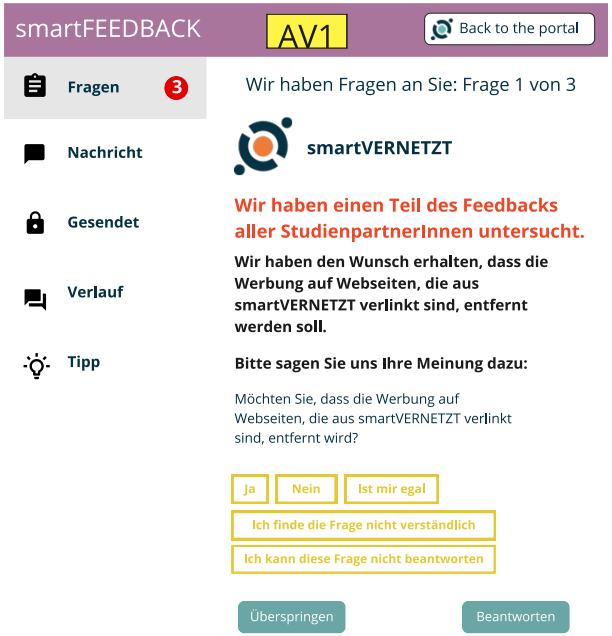


Figure B.2.53: FUQ with id AV2

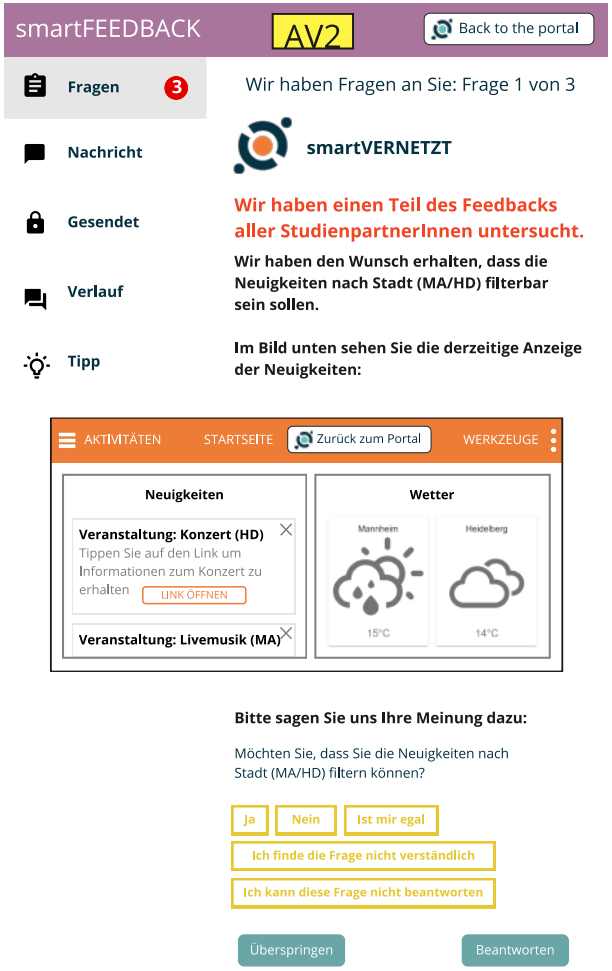


Figure B.2.54: FUQ with id AV3

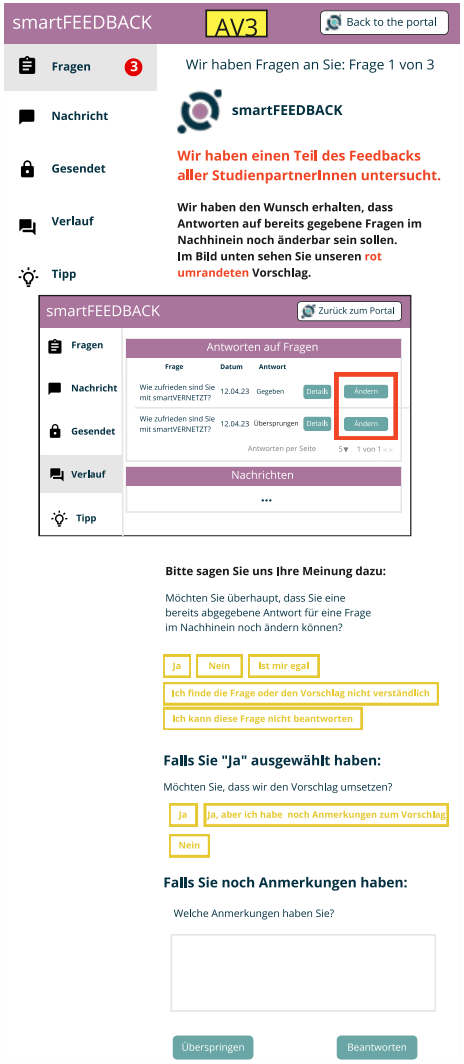


Figure B.2.55: FUQ with id AV4

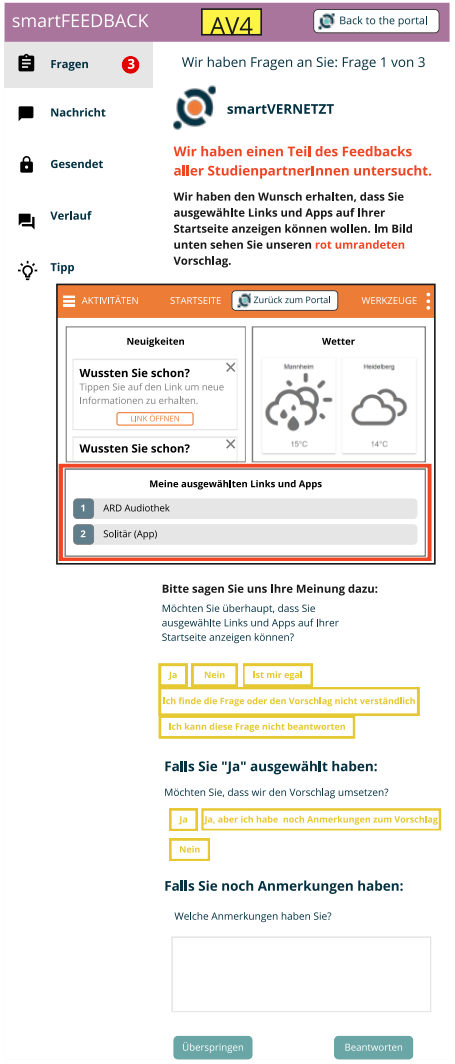


Figure B.2.56: FUQ with id AV5

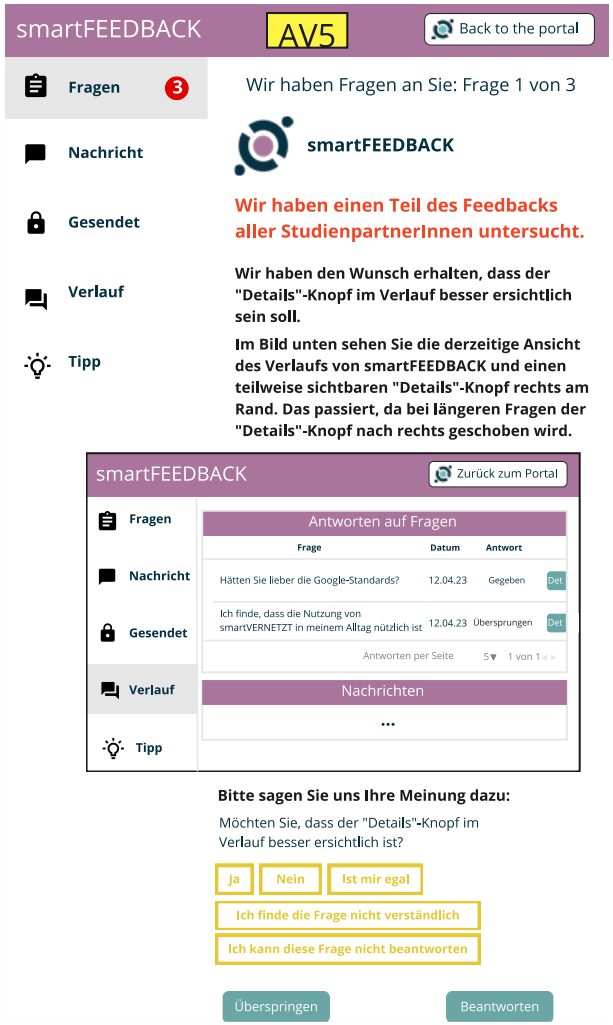


Figure B.2.57: FUQ with id AV6

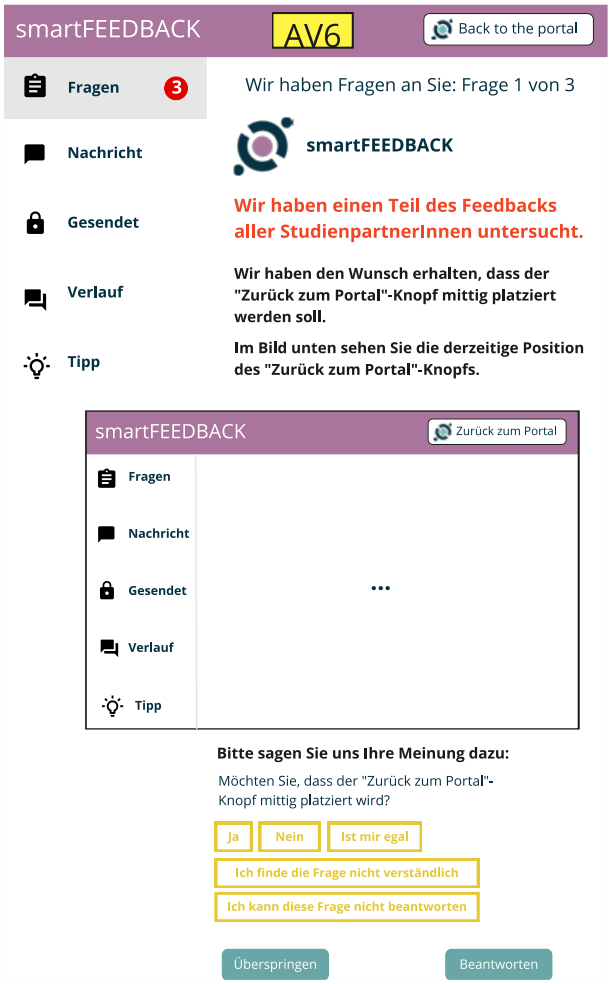


Figure B.2.58: FUQ with id AV7

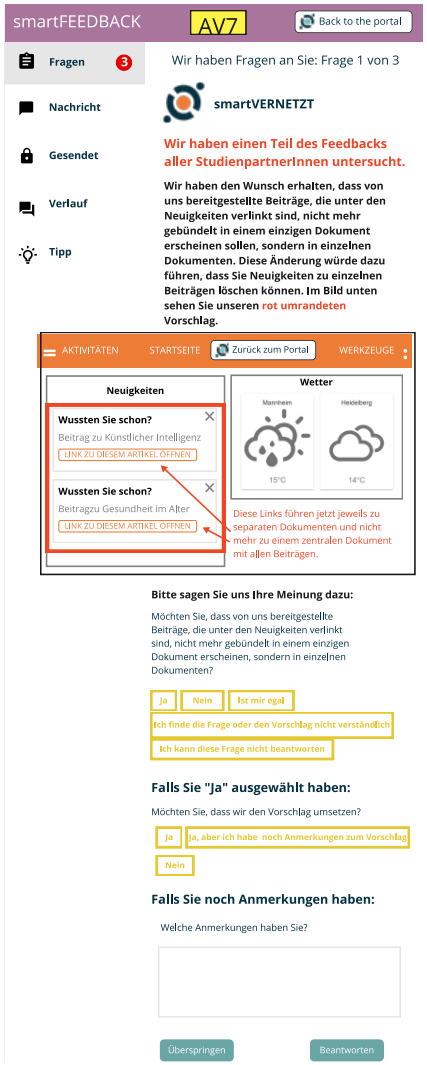
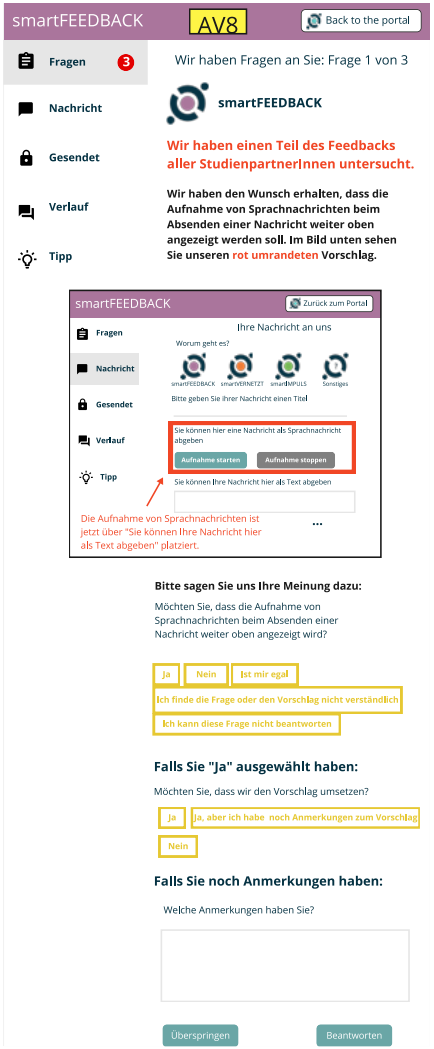


Figure B.2.59: FUQ with id AV8



FUQ of round 1 iteration 2

Figure B.2.60: FUQ with id FF1_FF

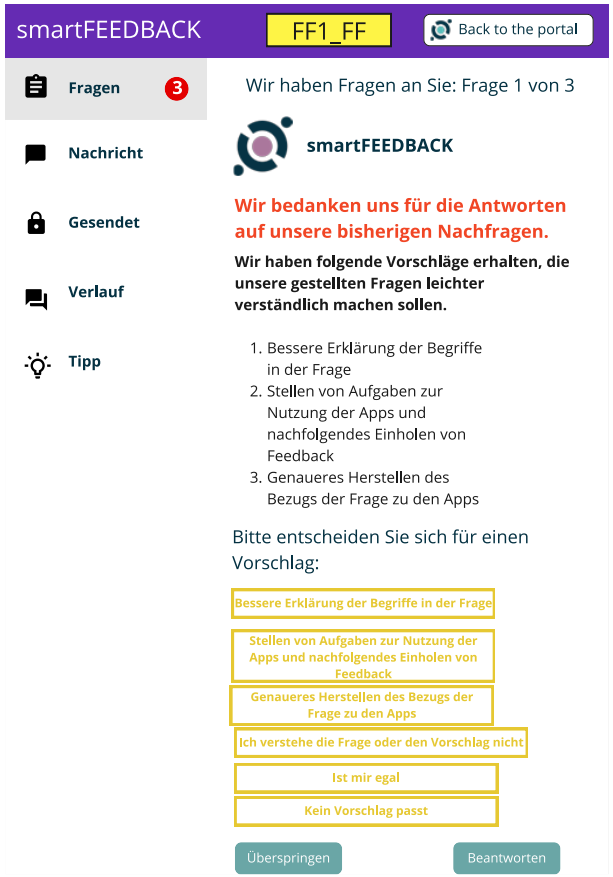


Figure B.2.61: FUQ with id FF2_FF

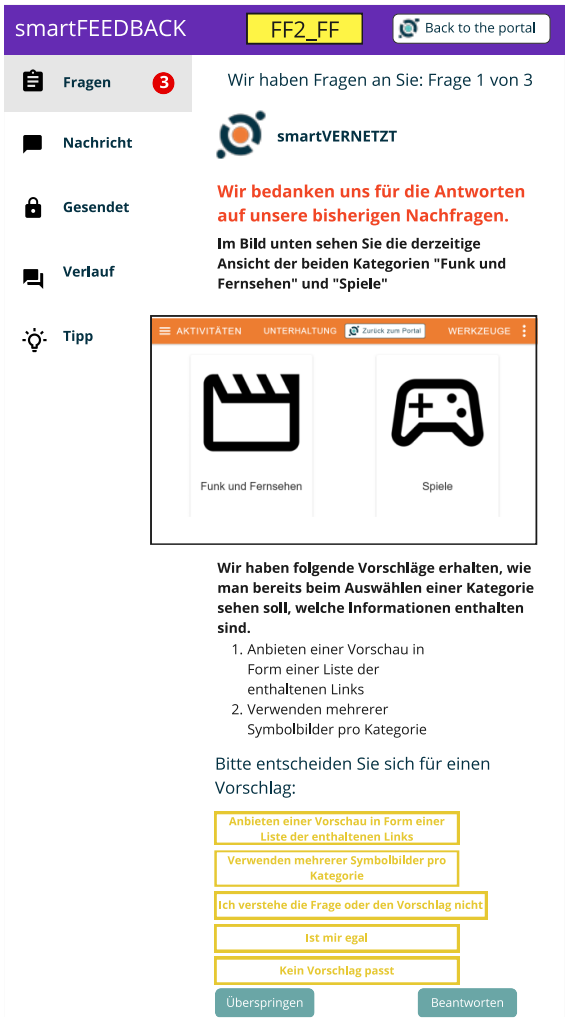


Figure B.2.62: FUQ with id FF3_AV

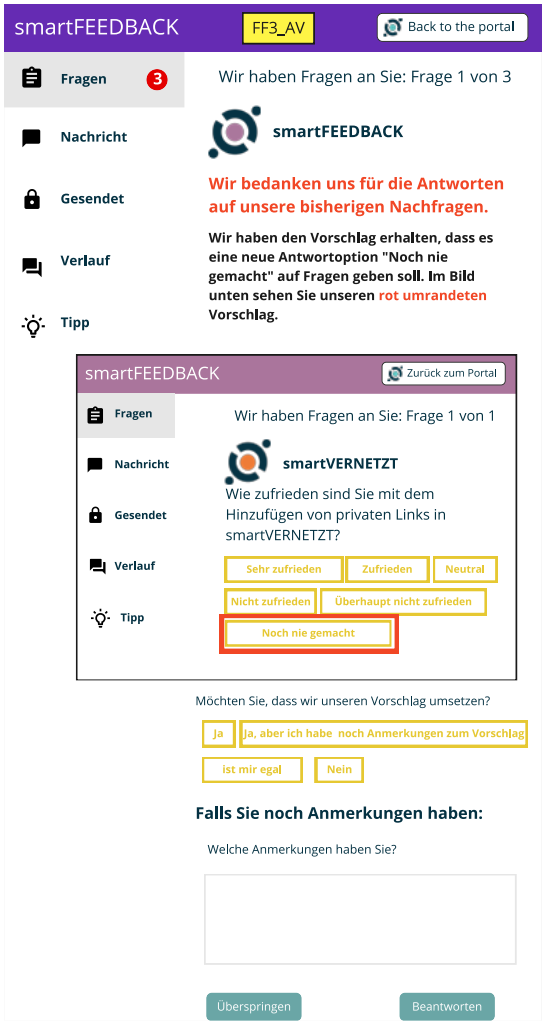


Figure B.2.63: FUQ with id FF4_AV

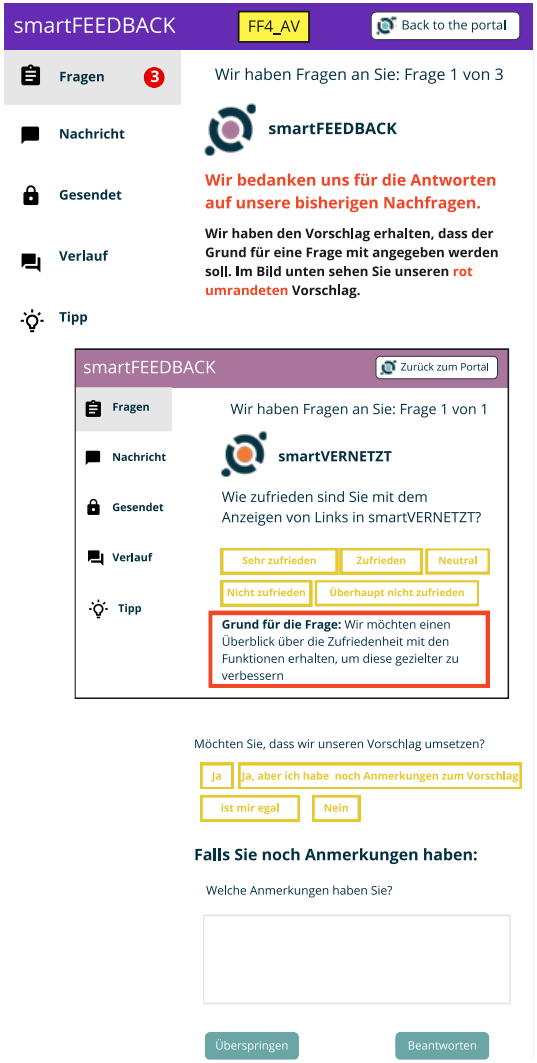


Figure B.2.64: FUQ with id FF5_AV

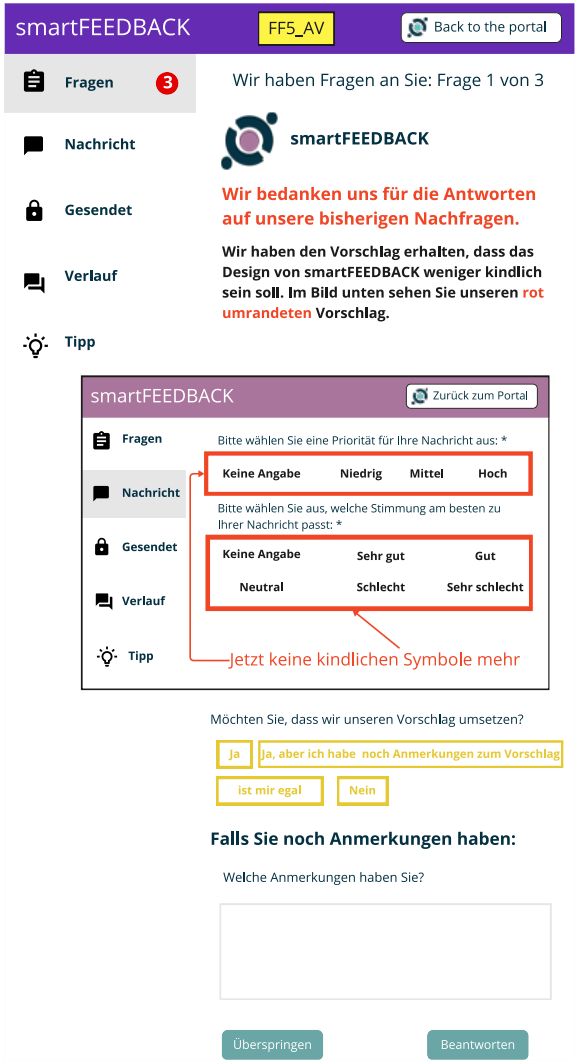


Figure B.2.65: FUQ with id FF6_AV

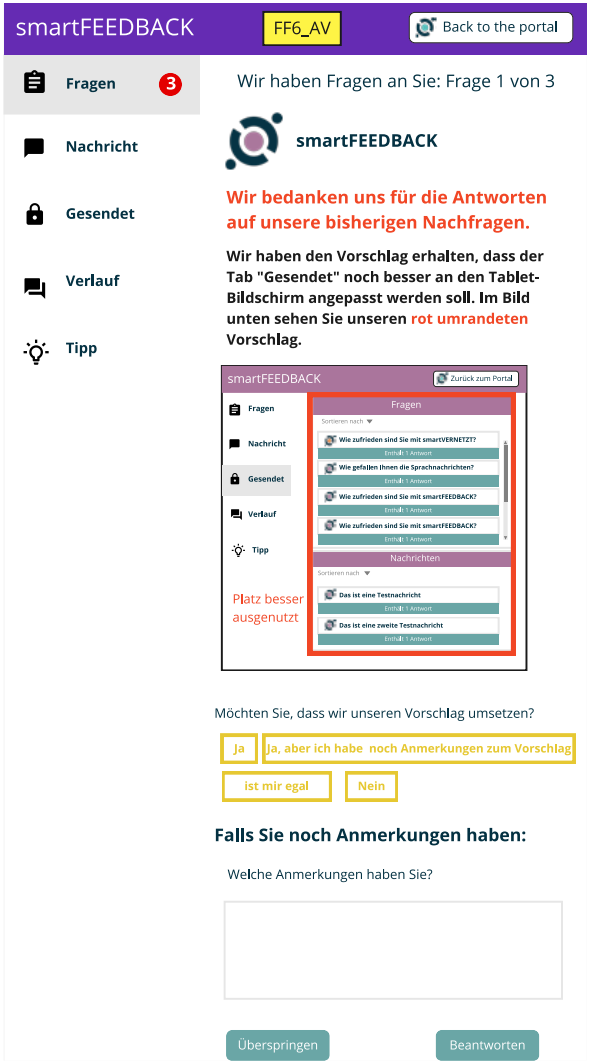


Figure B.2.66: FUQ with id FF7_AV

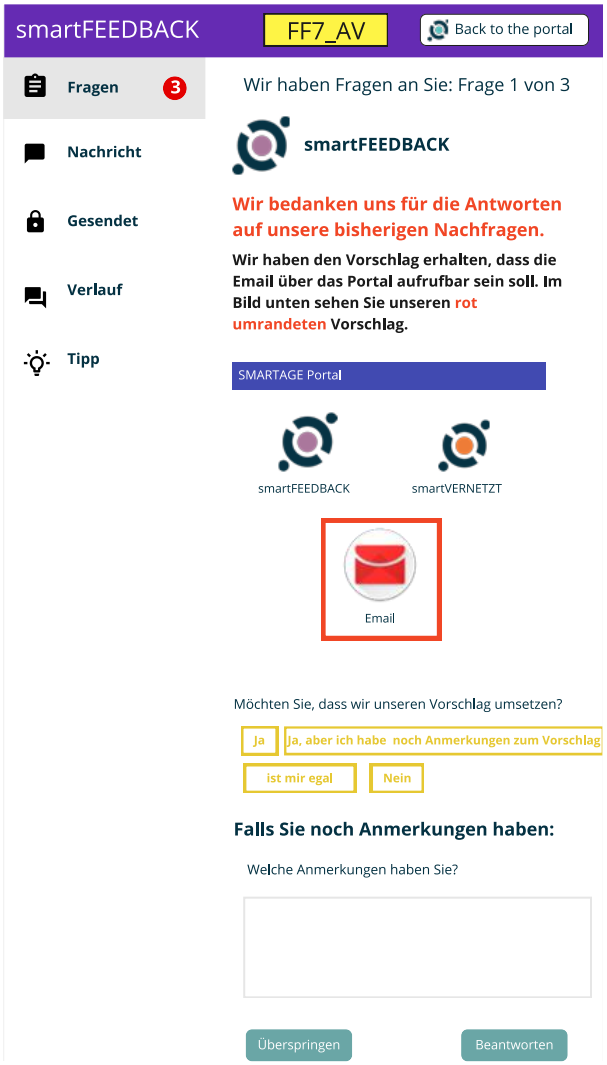


Figure B.2.67: FUQ with id FF8_AV

Figure B.2.68: FUQ with id FF9_FF

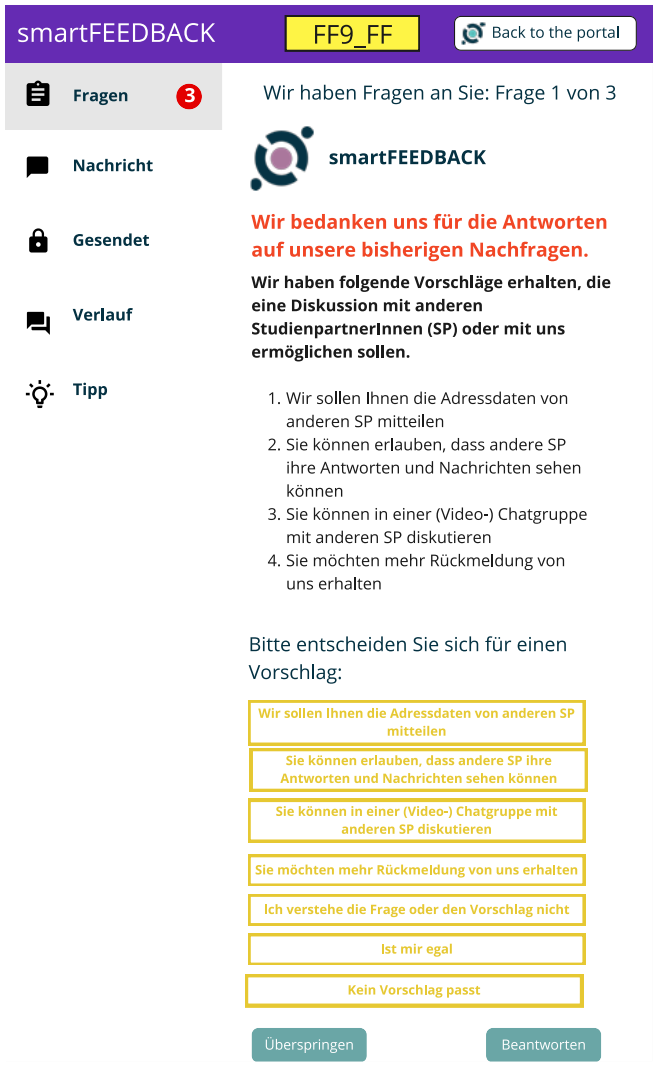


Figure B.2.69: FUQ with id FF10_AV

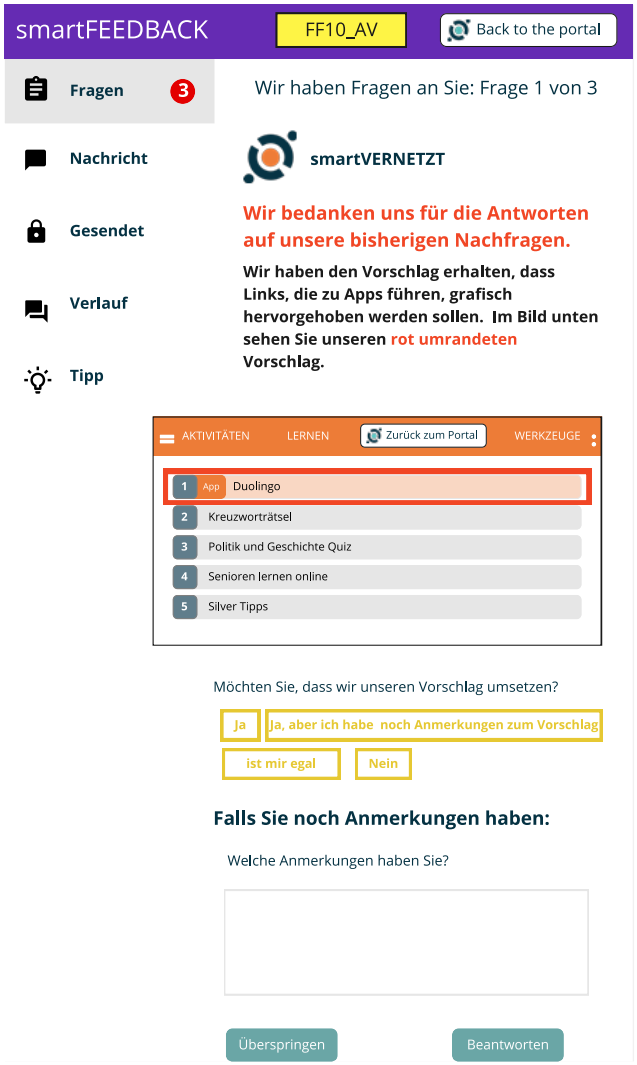
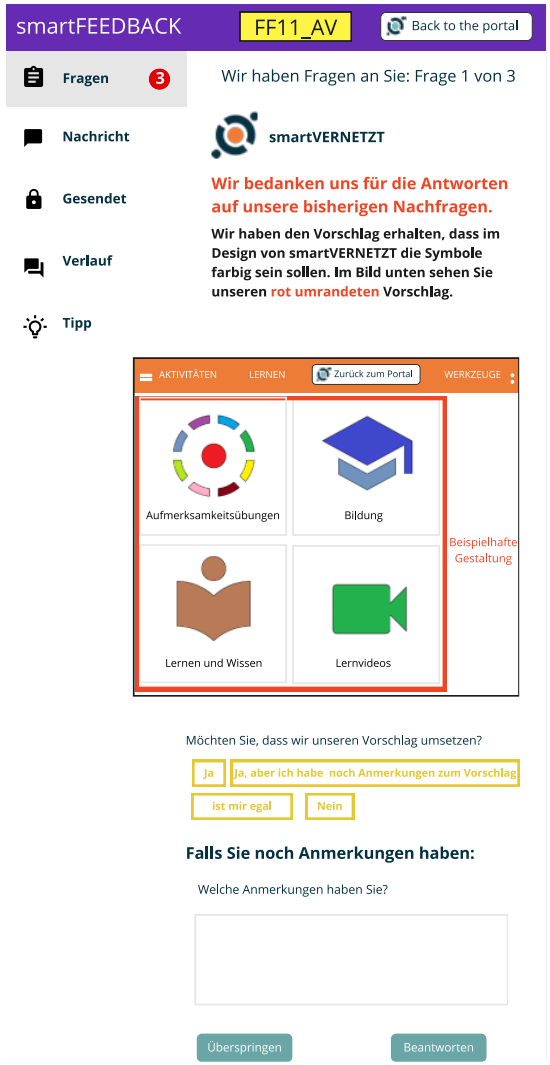


Figure B.2.70: FUQ with id FF11_AV



FUQ of round 2 iteration 1

Figure B.2.71: FUQ with id 2_FF1

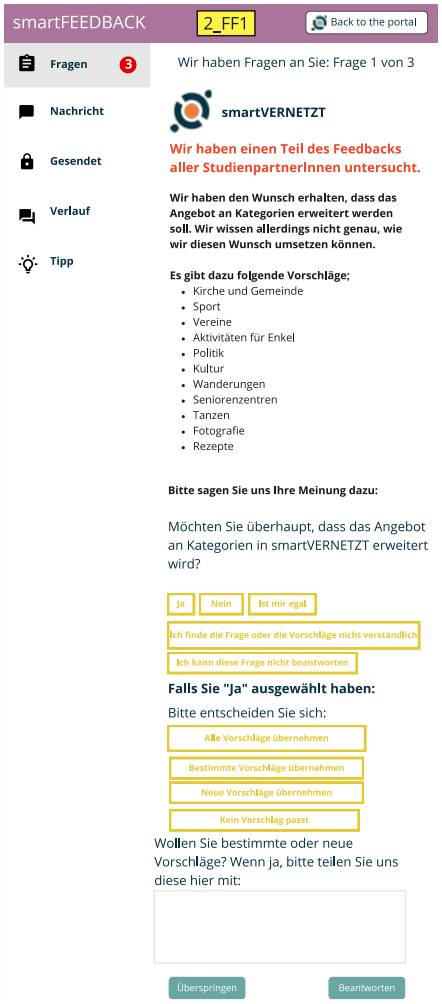


Figure B.2.72: FUQ with id 2_FF2

Figure B.2.73: FUQ with id 2_FF3

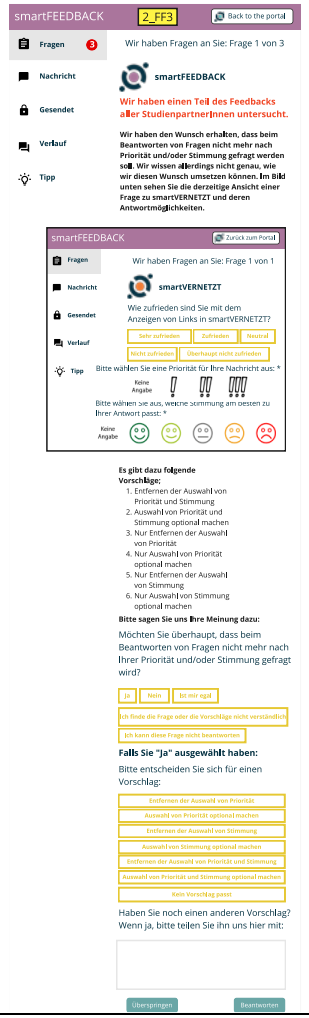


Figure B.2.74: FUQ with id 2_FF4

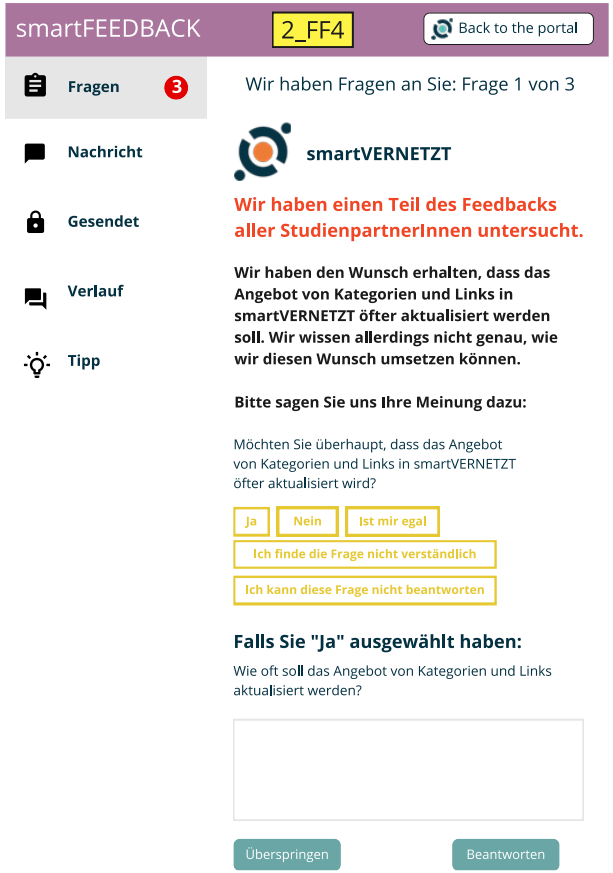


Figure B.2.75: FUQ with id 2_FF5

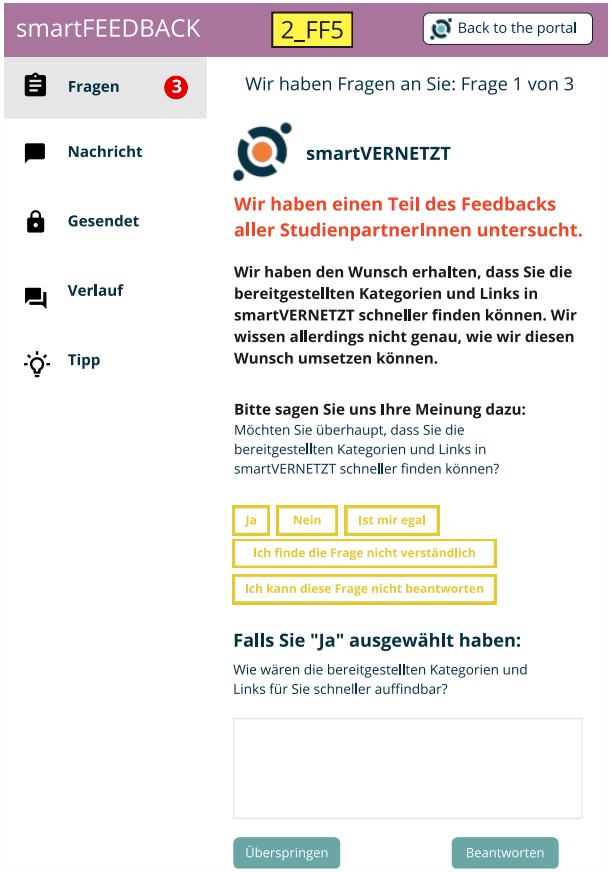


Figure B.2.76: FUQ with id 2_FF6

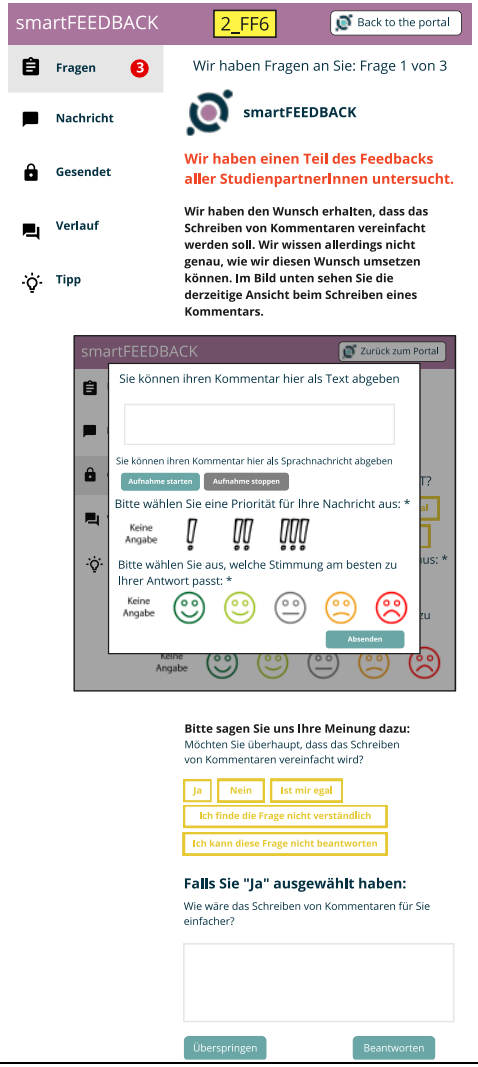


Figure B.2.77: FUQ with id 2_FF7

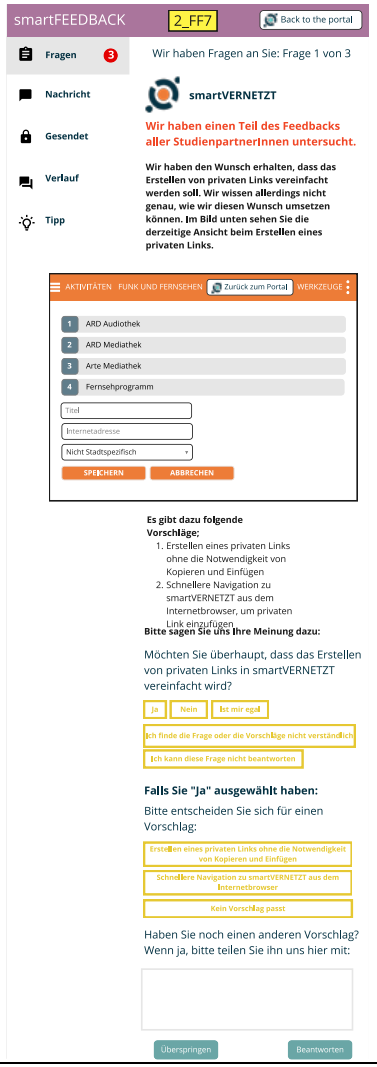


Figure B.2.78: FUQ with id 2_FF8

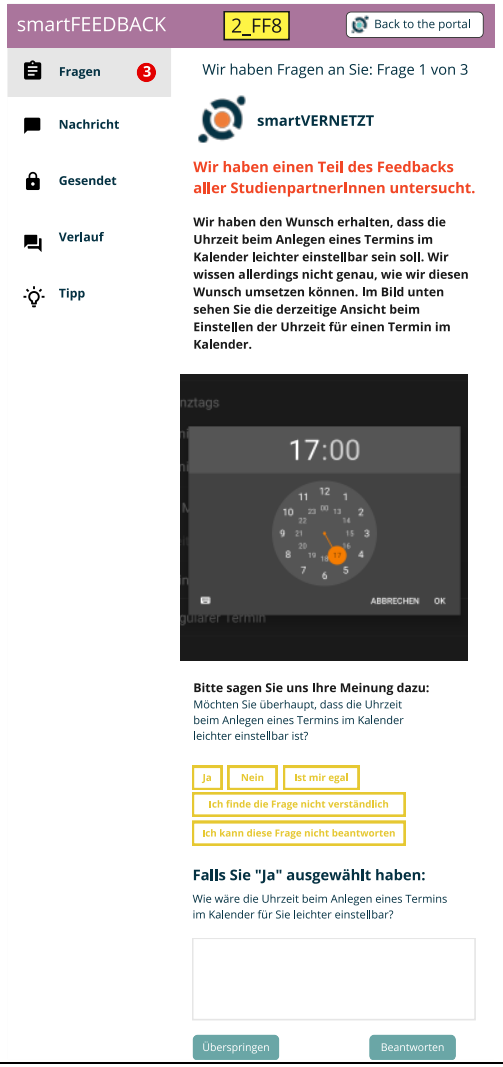


Figure B.2.79: FUQ with id 2_FF9

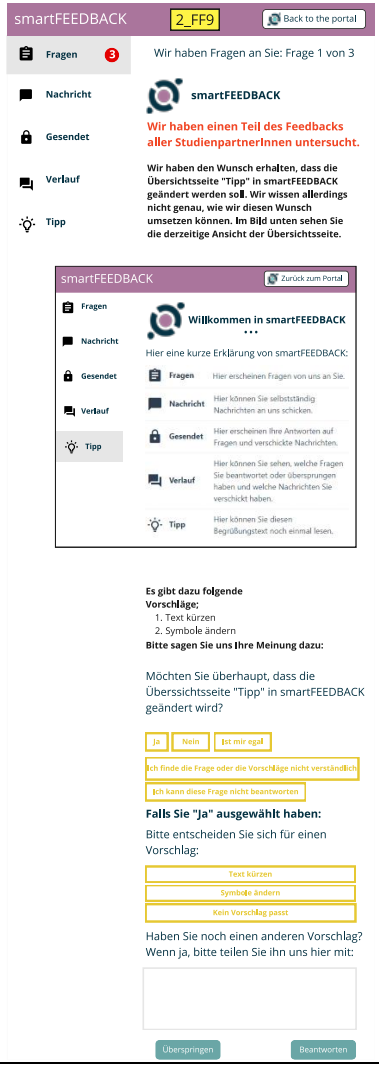


Figure B.2.80: FUQ with id 2_FF10

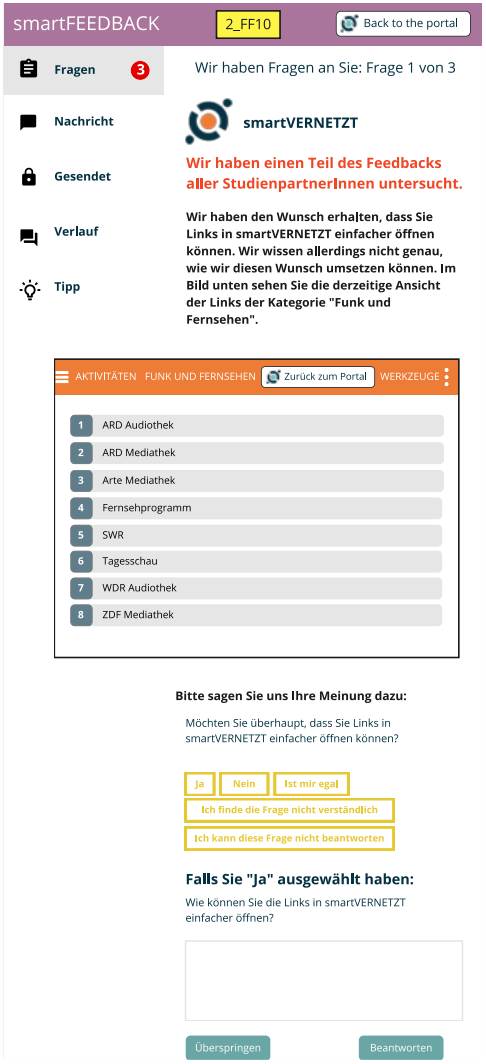


Figure B.2.81: FUQ with id 2_FF11

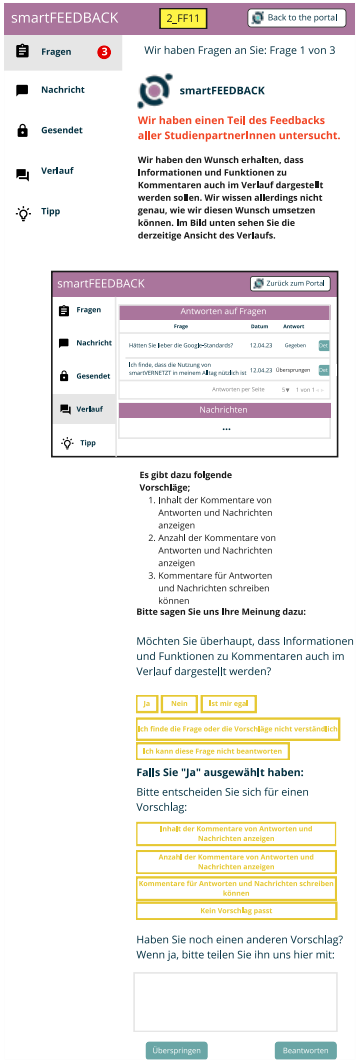


Figure B.2.82: FUQ with id 2_AV1

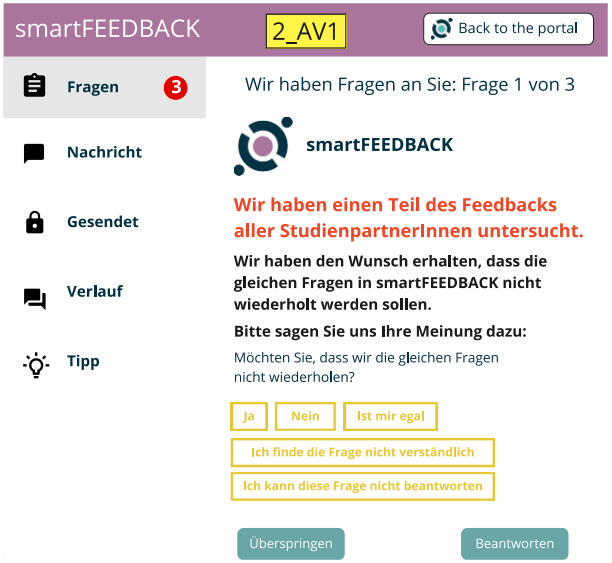


Figure B.2.83: FUQ with id 2_AV2

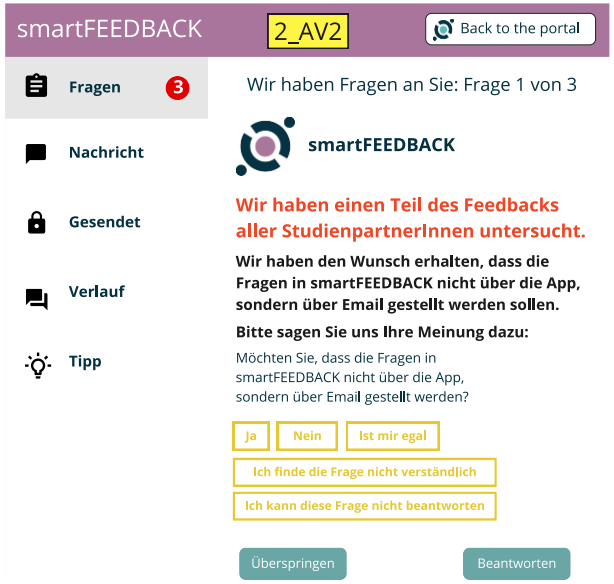


Figure B.2.84: FUQ with id 2_AV3

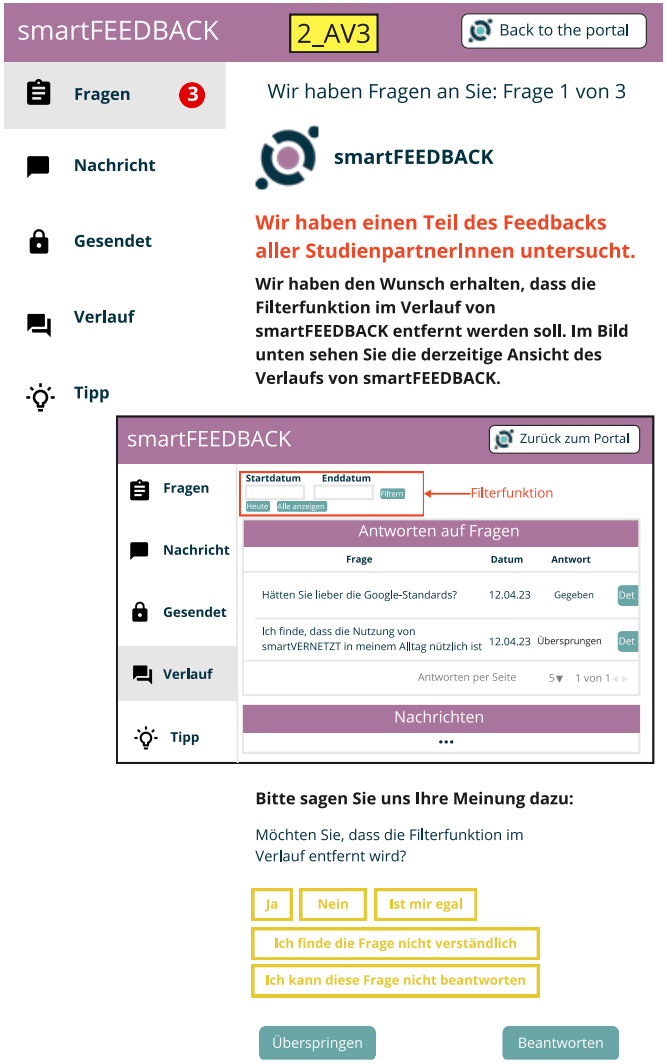


Figure B.2.85: FUQ with id 2_AV4

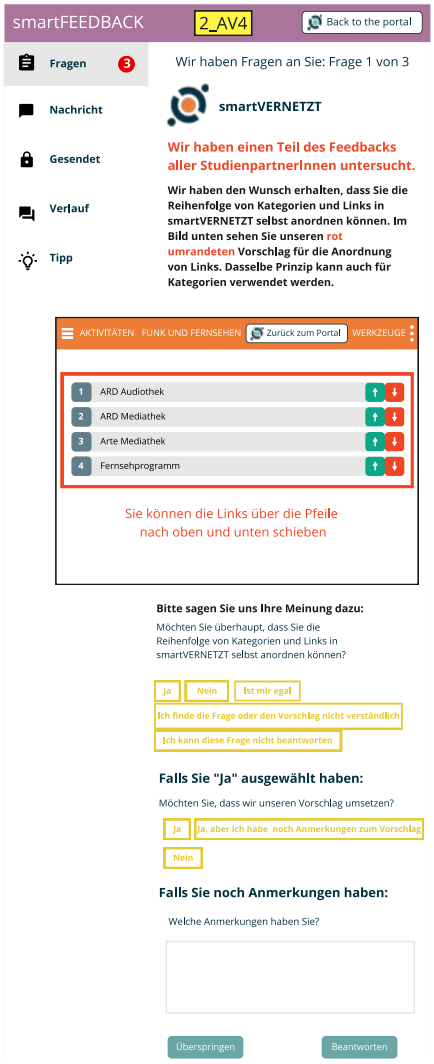


Figure B.2.86: FUQ with id 2_AV5

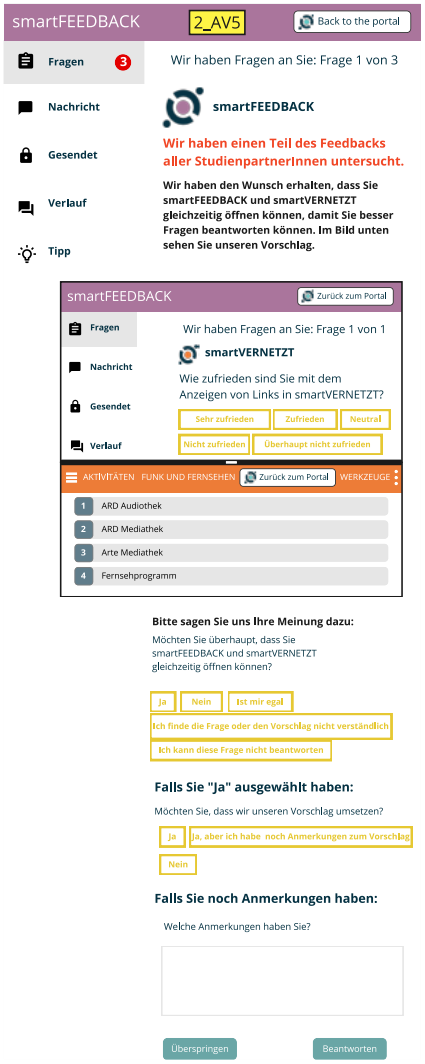


Figure B.2.87: FUQ with id 2_AV6

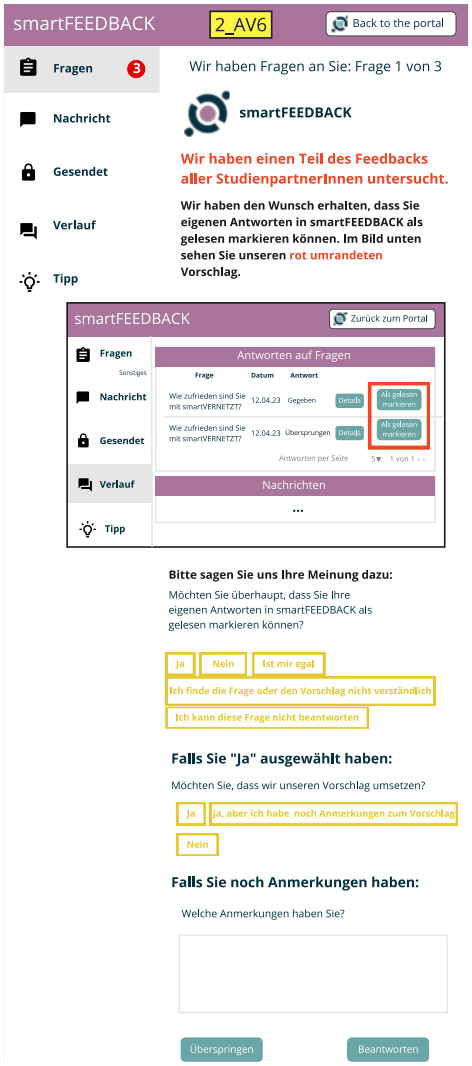


Figure B.2.88: FUQ with id 2_AV7

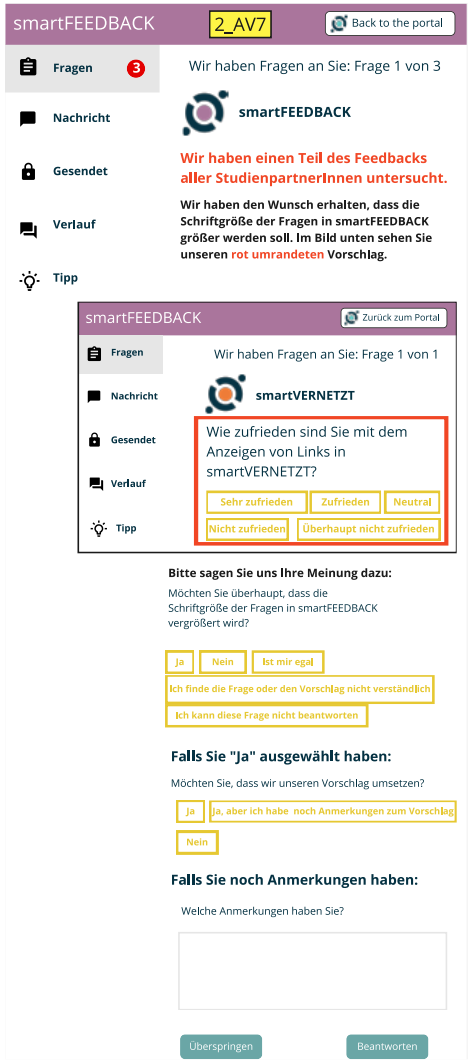
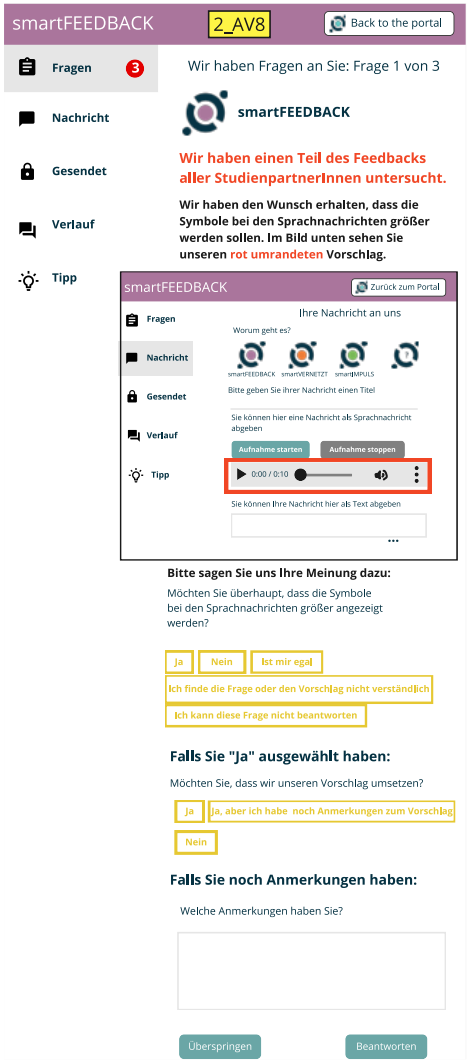


Figure B.2.89: FUQ with id 2_AV8



FUQ of round 2 iteration 2

Figure B.2.90: FUQ with id 2_FF1_AV

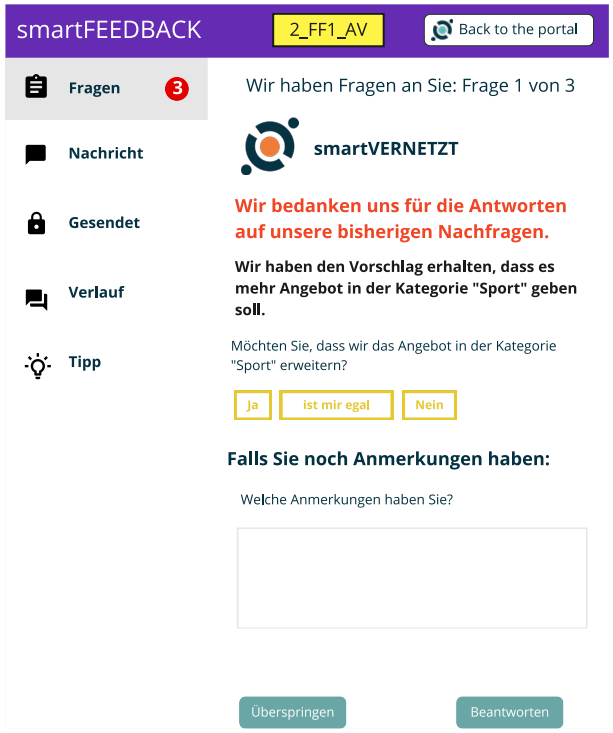


Figure B.2.91: FUQ with id 2_FF2_AV

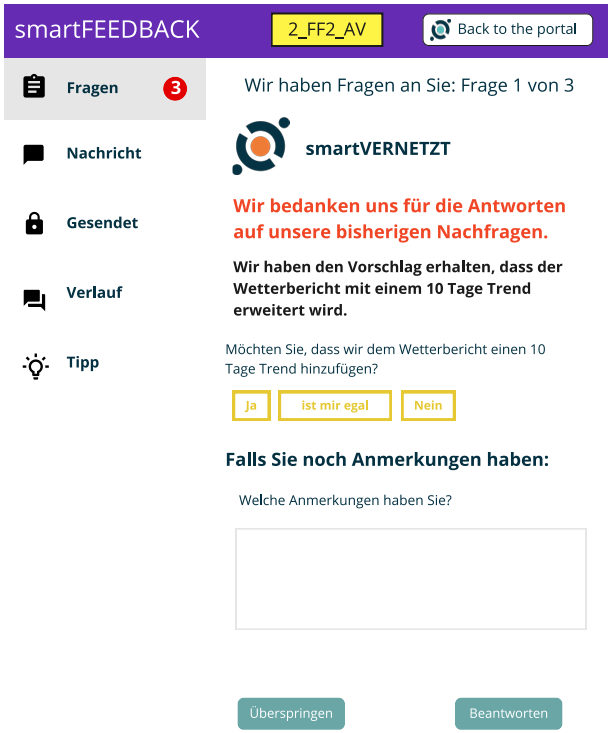


Figure B.2.92: FUQ with id 2_FF3_AV

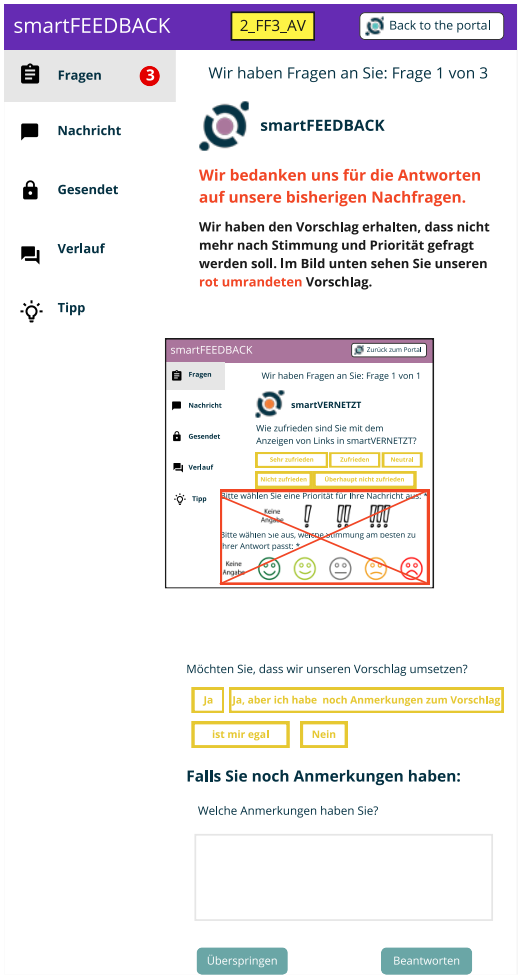


Figure B.2.93: FUQ with id 2_FF4_FF

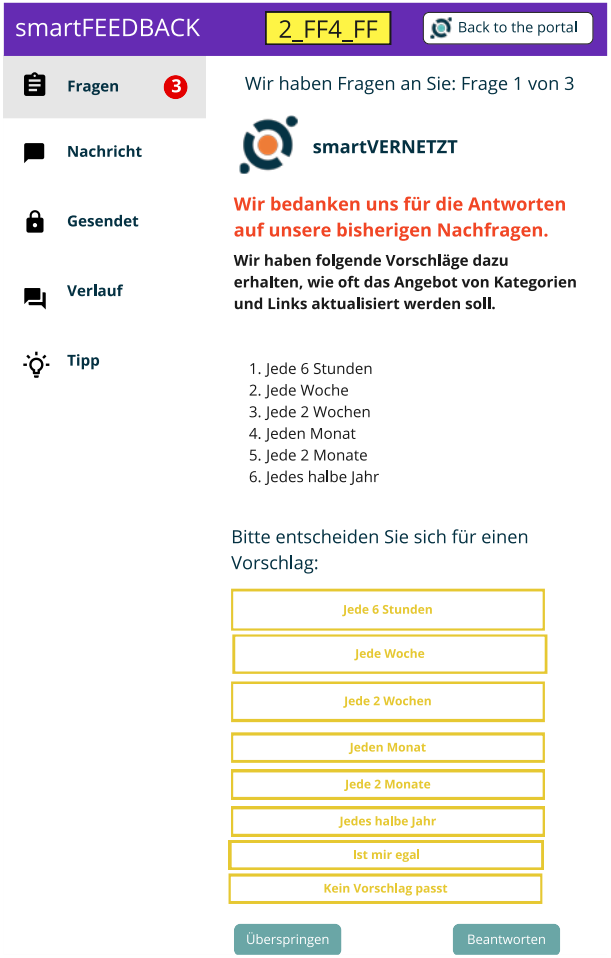


Figure B.2.94: FUQ with id 2_FF5_AV

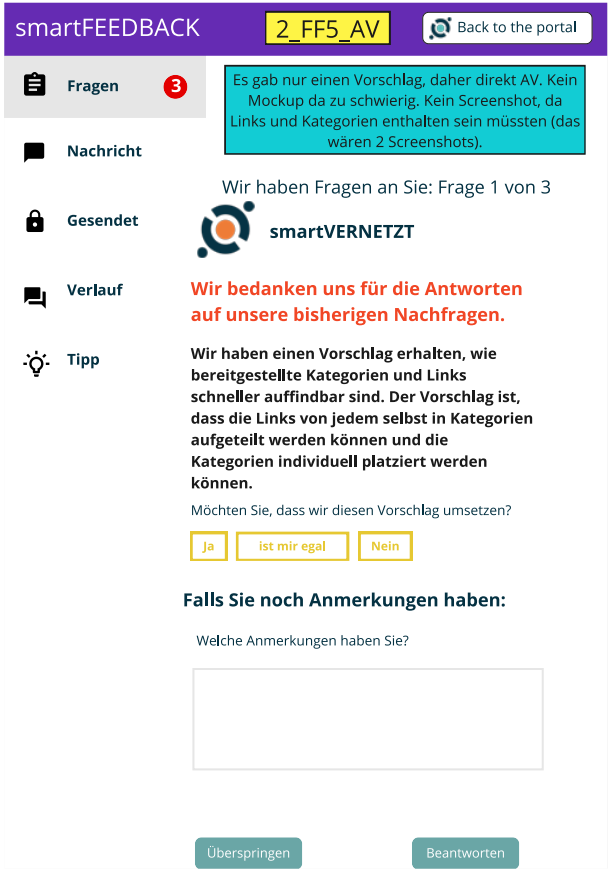


Figure B.2.95: FUQ with id 2_FF6_AV

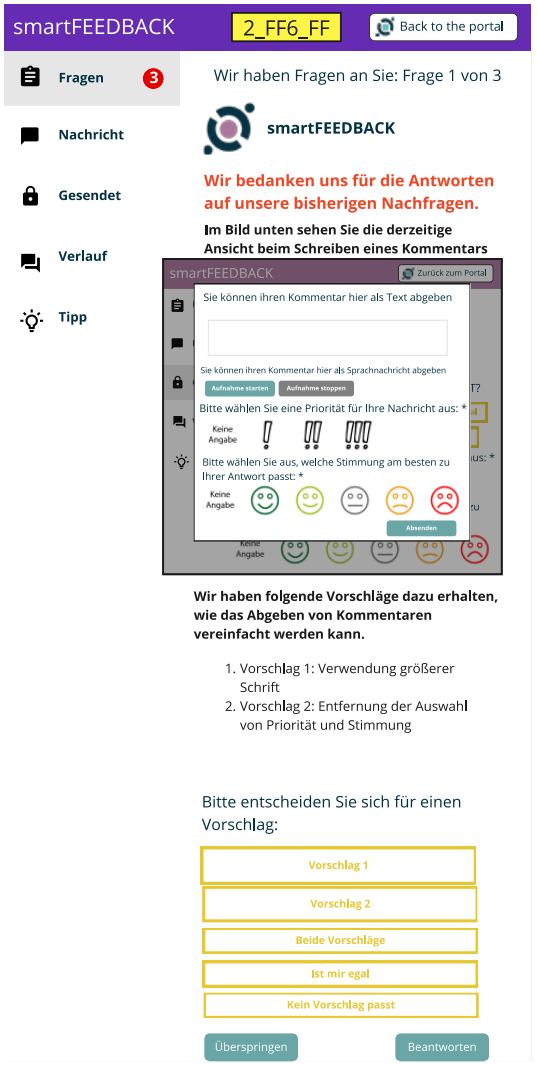


Figure B.2.96: FUQ with id 2_FF7_AV

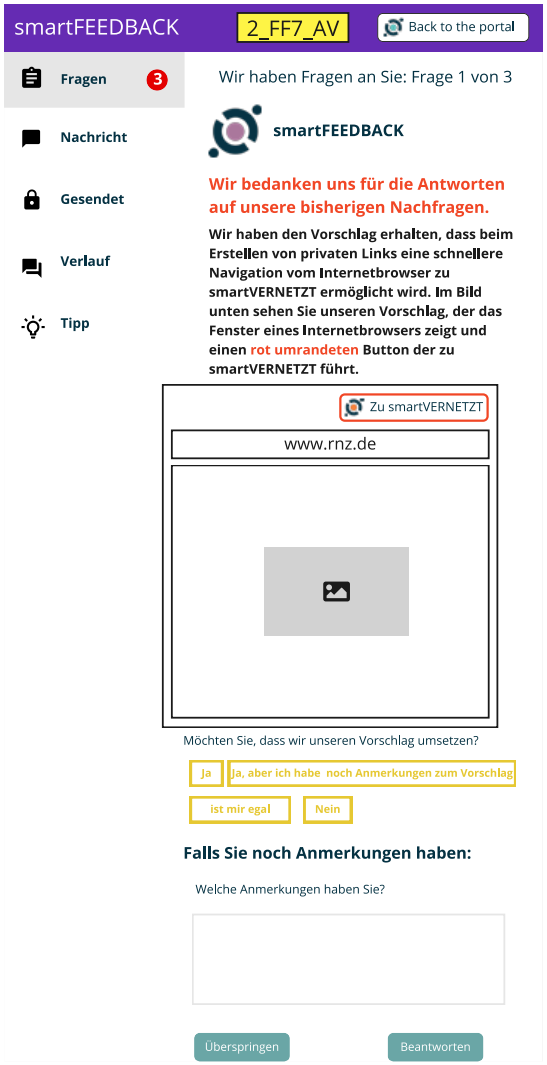


Figure B.2.97: FUQ with id 2_FF8_FF

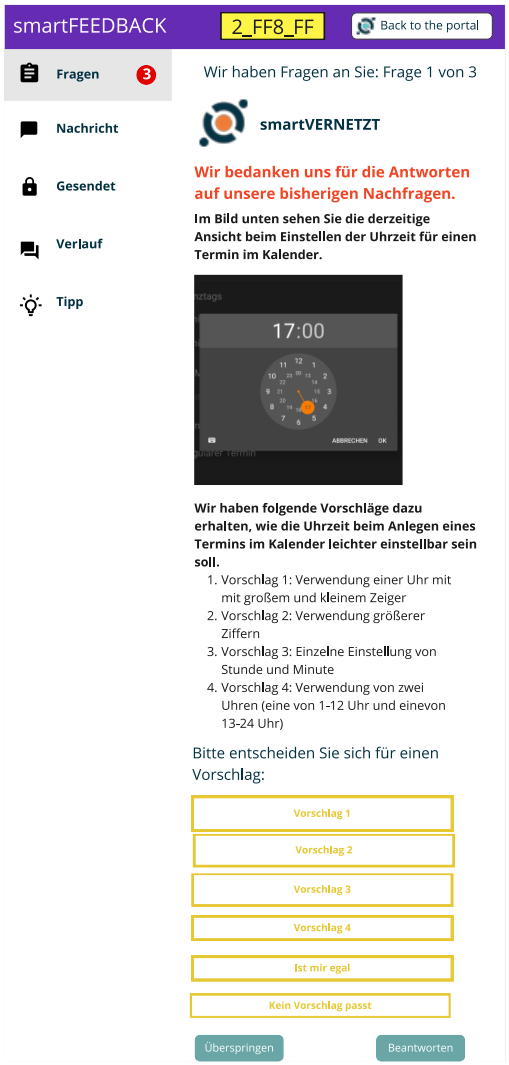


Figure B.2.98: FUQ with id 2_FF9_AV

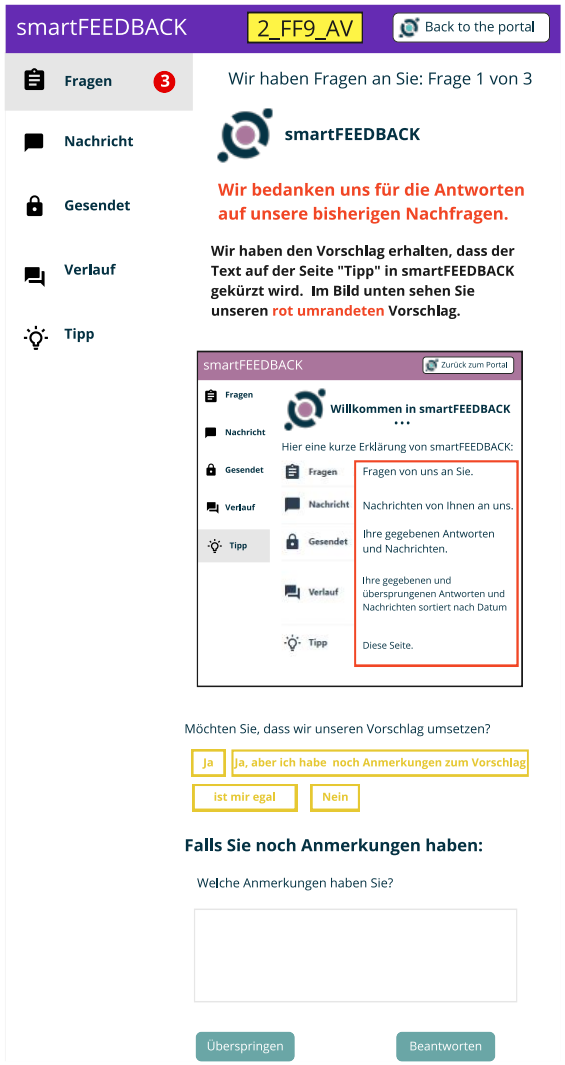
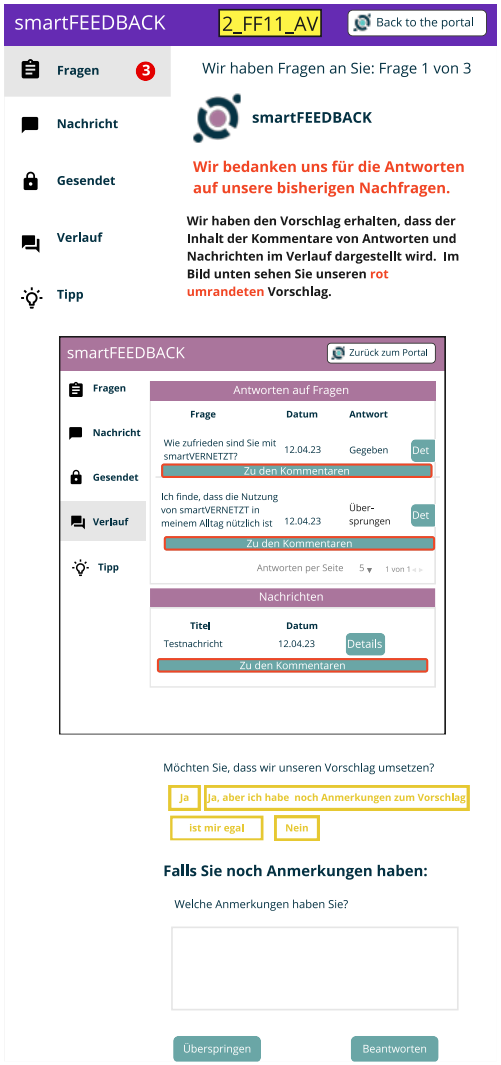


Figure B.2.99: FUQ with id 2_FF11_AV



B.5 Handbook for SF

We describe the features that are used by the REeng in Section "Features for the REeng" and the features that are used by the users in Section "Features for the user". We start with the features of the REeng, because the question structure is explained, which is necessary to understand when answering questions. We use screenshots of the implemented virtual windows of Section 5.1.4 to explain the features. We don't explain each system function in detail. For a complete description of all system functions, see Table B.3.1 and Table B.3.2. To see how each individual system function is related to the user interface, see the virtual windows in Section 5.1.4.

Features for the REeng

We explain how the REeng can manage questions in Section "Managing questions (UT1S1(R))". We describe how the REeng can manage given feedback in Section "Managing given feedback (UT1S2(R))".

Managing questions (UT1S1(R))

Figure B.5.1 shows a screenshot for the implemented virtual window of workspace W: QuestionView (R). The REeng has the option to navigate to the question overview through ❶ and to the overview of received feedback through ❷. The REeng sees all existing questions in the area ❻. The questions are shown along with their ID, their name, their on/off status (whether they are activated or not) and with different action buttons. The REeng can enable and disable a question through the toggle switch in the on/off column. With the ✎ action button, the REeng can edit a question, with the 🕓 action button the REeng can schedule a question and with the 🗑 action button the REeng can delete a question. The REeng can search through the questions with ❸. The REeng can sort the columns of the question table with the arrows next to the table captions (❺). The column data can be sorted ascending and descending. The REeng can create a new question with ❹.

Figure B.5.1: Screenshot for the implemented virtual window of workspace W: QuestionView (R)

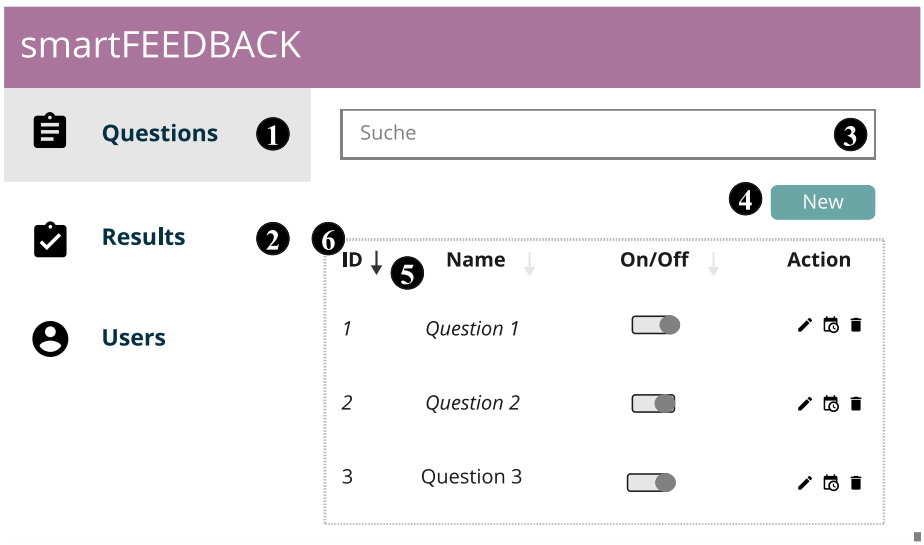
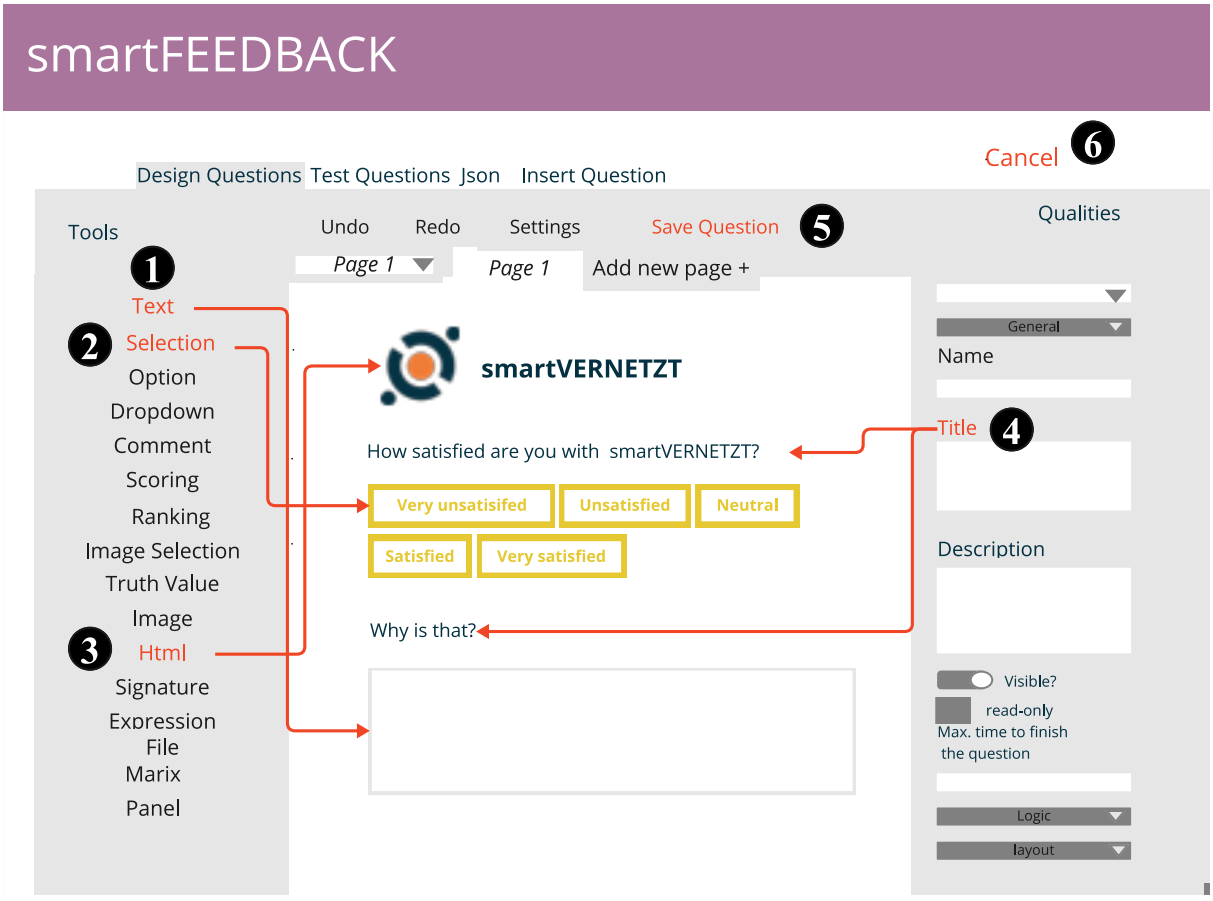


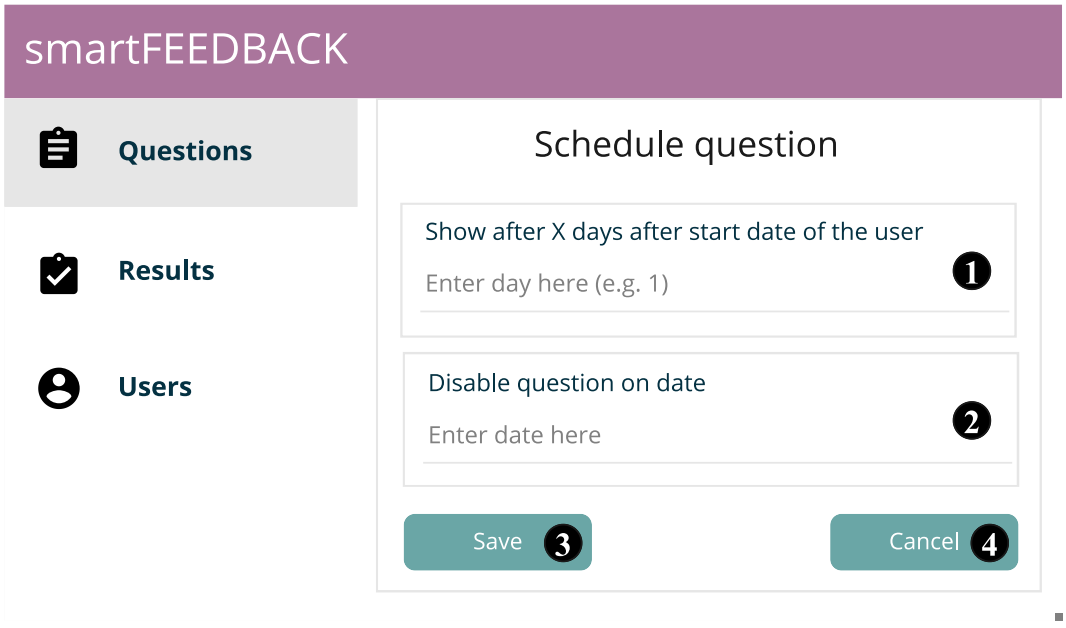
Figure B.5.2 shows a screenshot for the implemented virtual windows of workspace W: CreateQuestionView (R). The REeng can see this screen after pressing the ✎ action button or after pressing the "new" button that is shown in Figure B.5.1. On the left side different tools are shown including text, selection, options, dropdowns, html etc. These tools can be dragged and dropped in the middle. To support our process we only need the answer options that are highlighted in red (text (❶), selection (❷) and html (❸). The text tool generates a question with a title and a text input field as answer option. The selection tool also generates a question with a title, but with different selection options that the user can choose from to answer the question. The html allows arbitrary html code to be used. We use the html tool to design the top part of a question where we display the application logo and name. The REeng can use the title (❹) to give questions a title and ❺ to save the question. The REeng can cancel the question creation with ❻. All other functionalities in Figure B.5.2 that are not red can still be used for example to adapt the process in the future through questions with e.g. image selection but they are not needed to support the version of the process that is presented in this thesis. We still leave the functionalities visible, to remind the REeng that the question creation is very flexible. We explain in Section 5.2.1.2 how the functionality is implemented.

Figure B.5.2: Screenshot for the implemented virtual window of workspace W: QuestionView (R)



After the question is saved it appears in the list of questions of Figure B.5.1. The REeng can then schedule when the question should be asked to the users. For this, the REeng presses the 🕓 action button in Figure B.5.1 for the specific question. The screen Figure B.5.3 is shown to the REeng.

Figure B.5.3: Screenshot for the implemented virtual window
of workspace W: ScheduleQuestionView (R)

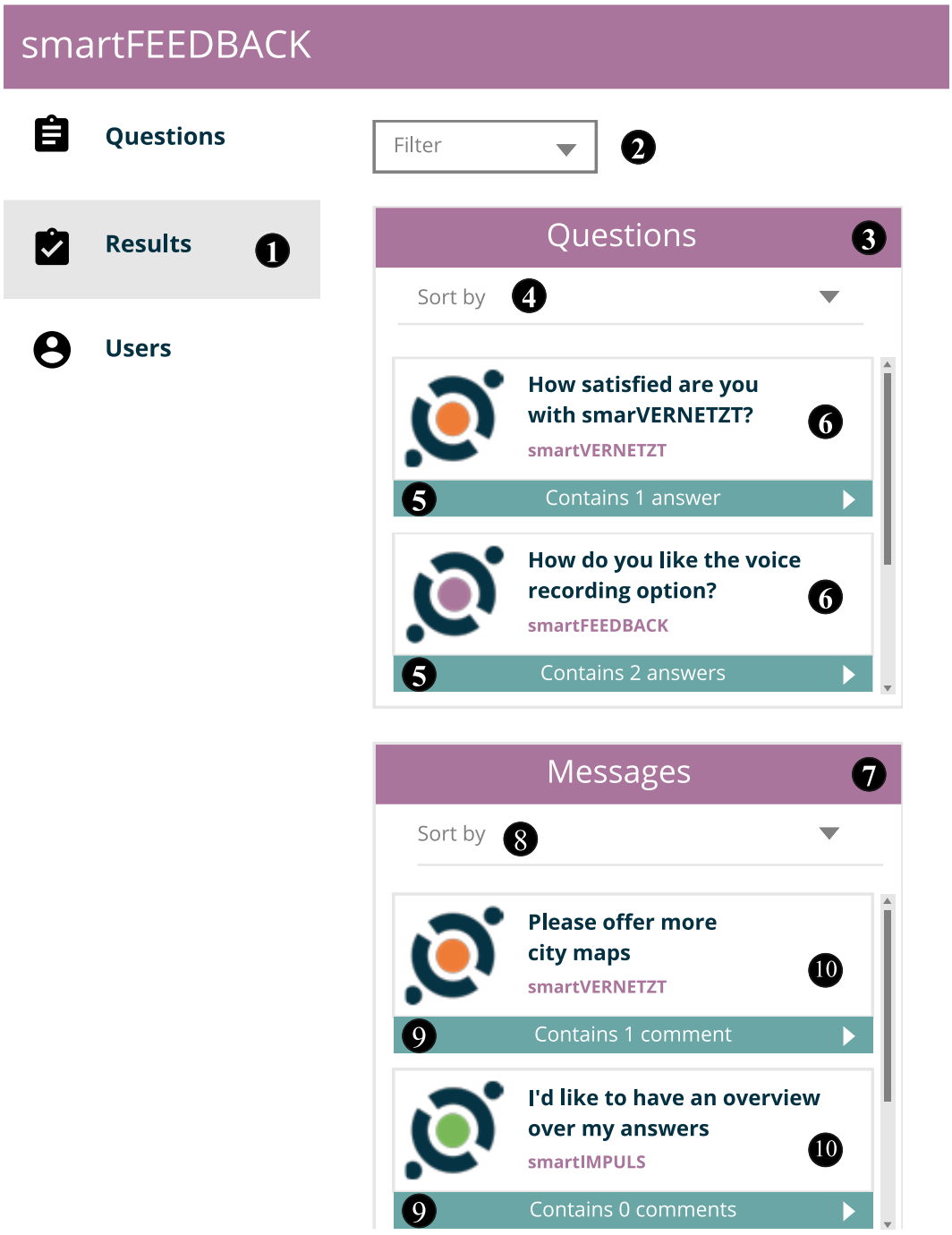


The REeng can either use one of the two options (❶ or ❷) to schedule a question. With ❶ the REeng can show the question after a number of days relative to the day when the user started using SF. When "1" is entered, that means that the question is displayed one day after the users' start date to the user. The REeng can also input multiple days such as "1, 100". This means that the questions appears one day after the users' start date and then again 100 days after the start date. If the user doesn't answer the question before the day 100, the question will not be asked twice on day 100, but it will just remain open. With ❷ the REeng can disable a question at a specific date for all users. This means that when the date is reached, no user will be able to receive the question anymore. We use ❶ to schedule our questions of type "scheduled" and a combination of ❶ and ❷ to ask FUQ only to users who at least have 7 days of usage and to disable our FUQ at a specific date for all users. The REeng can save the schedule of a question with ❹ and the schedule can be canceled with ❺. After saving or cancellation the REeng is redirected to back to Figure B.5.1.

Managing given feedback (UT1S2(R))

Figure B.5.4 shows a screenshot for the implemented virtual window of workspace W: QuestionView (R). The REeng can see this screen after pressing "Results" (❶) in the sidebar. The REeng can see all questions that contain at least one answer in ❸.

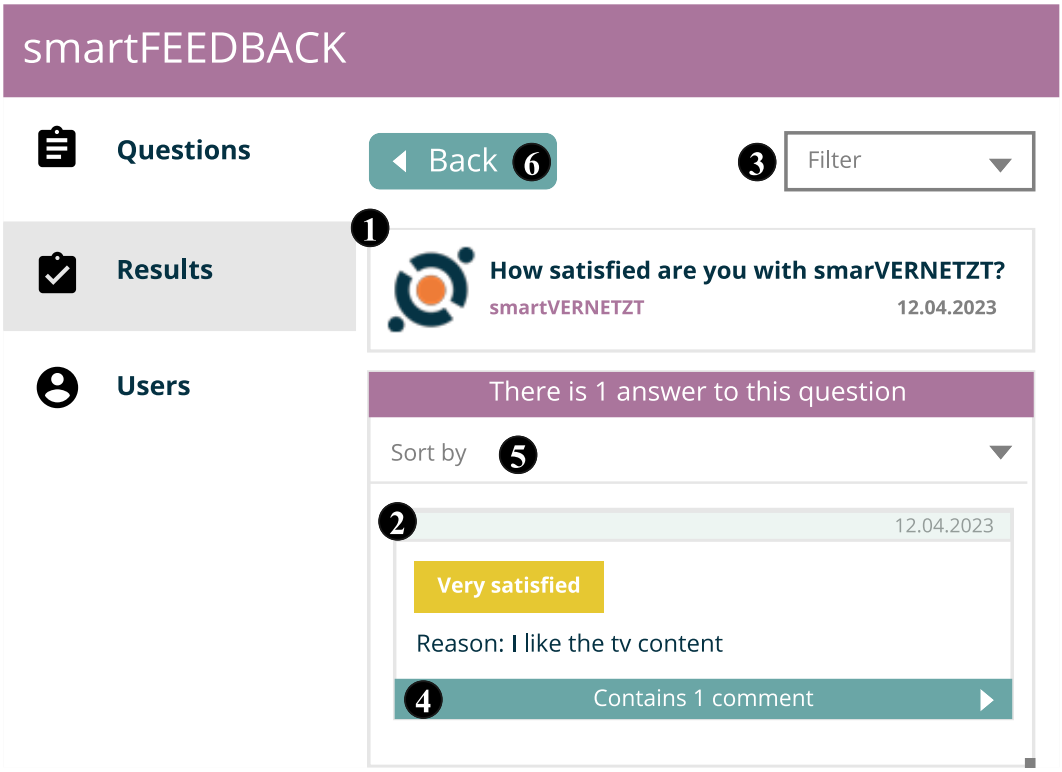
Figure B.5.4: Screenshot for the implemented virtual window of workspace W: ResultsView (R)



The questions are shown with application logo, title and application name (❻). The questions can be sorted with ❹ by the number of answers of the users, by the date of the last answer and by the date of the last comment. This helps the REeng to identify popular questions and questions that are currently answered or commented. The messages (❼) are also shown with application logo, title and application name (❿). The REeng can sort the messages by creation date and by creation date of last comment to identify new messages or comments quickly. The REeng can filter questions and messages simultaneously through ❷. The REeng can filter by date range (for questions the date of the last answer and for messages the date of their creation). The date ranges

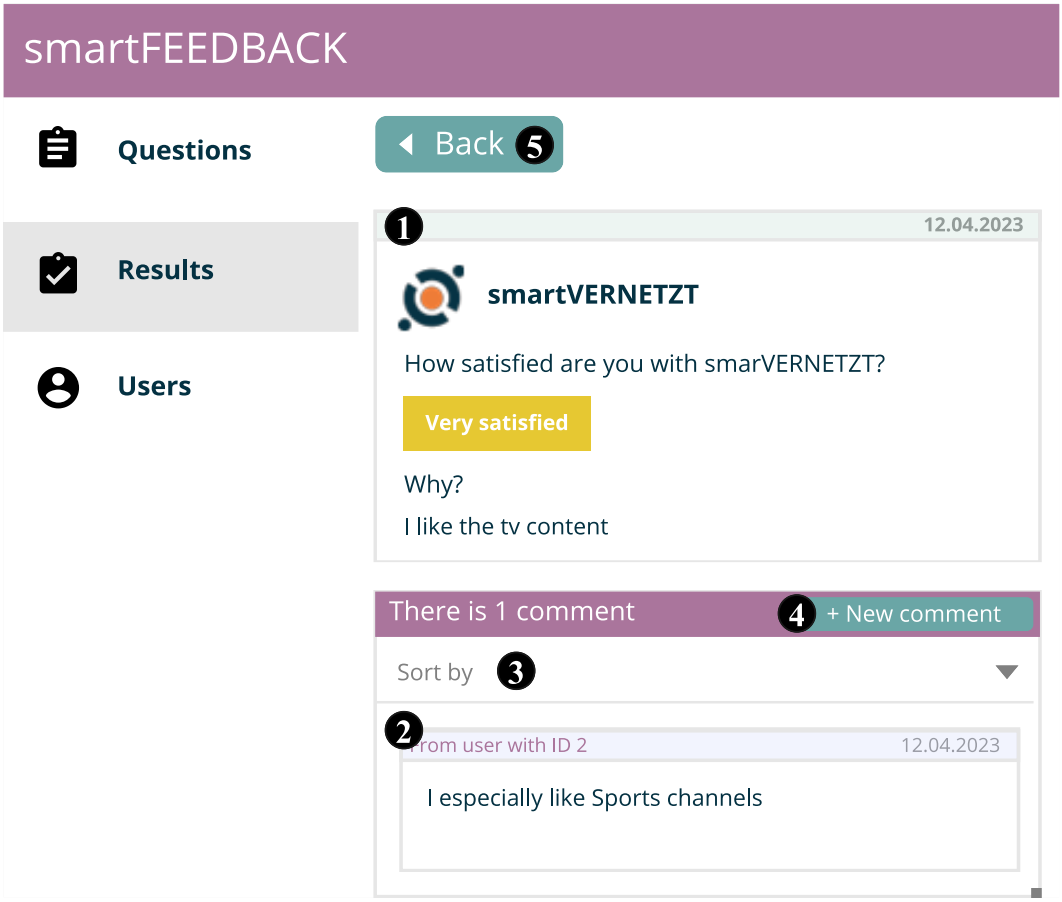
are last week, this week and today. This allows the REeng to filter for relevant feedback when reviewing the feedback. The REeng can also filter questions and messages for a specific app only. This is helpful, when the REeng wants to inspect the feedback for only one app. The REeng can navigate to all answers of a questions with ❺. After navigation the REeng is presented with the screen in Figure B.5.5.

Figure B.5.5: Screenshot for the implemented virtual window of workspace W: DetailedQuestionView (R)



The question for which the answers are presented is shown at the top (❶) in Figure B.5.5. The question is presented with the application logo and name, as well as the title of the question and the day when the question was last answered. The answers of the question are shown at ❷. There is only one answer in this case. The answer contains the selected answer option and a reason in freetext below. The answers are shown without the question title to save space, because the question title is always the same for each answer. The creation date of the answer is shown in the top right corner. The REeng can filter and sort the answers with ❸ and ❺. The REeng can filter by creation date of the answer (last week, this week, today). The REeng can sort the answers by creation date as well. With ❻ the REeng can navigate back to Figure B.5.4. The button ❹ shows the number of comments of the answer and allows the REeng to navigate to the details of an answer. Figure B.5.6 shows how the details of an answer are presented.

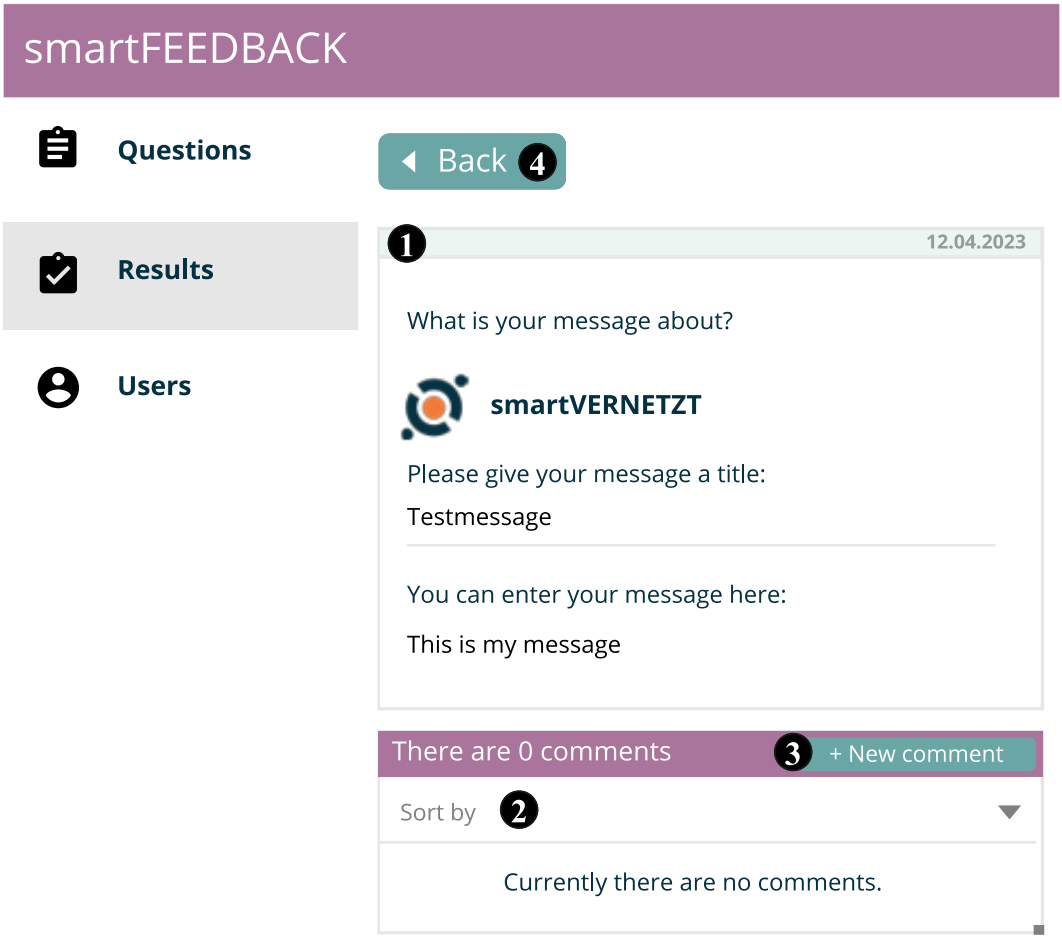
Figure B.5.6: Screenshot for the implemented virtual window of workspace W: DetailedAnswerView (R)



In Figure B.5.6 the answer is shown in ❶. The answer is shown with the application logo and name, as well as the question title. The comments of the answer are shown in ❷ with the author of the comment (e.g. user with ID 2) in the left upper corner and the creation date of the comment in the right upper corner. The content of the comment is shown below as text. If the comment is given as a voice message, the voice message appears and can be played. The comments differ in color depending on whether they are given by users or by the REeng, to make it easier for the REeng to identify the comment authors. When a comment is written by the REeng, the author of the comment is shown as "By us". The REeng can sort the comments through ❸ by the date of the creation of the comment. The REeng can navigate back to Figure B.5.5 through ❺. The REeng has the option to create a new comment through ❹. In the context of SMARTAGE, the REeng doesn't comment on answers to questions, because this would not be feasible due to the large number of answers. The REeng only comments on messages from users that are very urgent and require a reply to avoid causing dissatisfaction to the user.

To view the details of a message, the REeng can press on ❾ Figure B.5.4. The details of a message are then shown in Figure B.5.7.

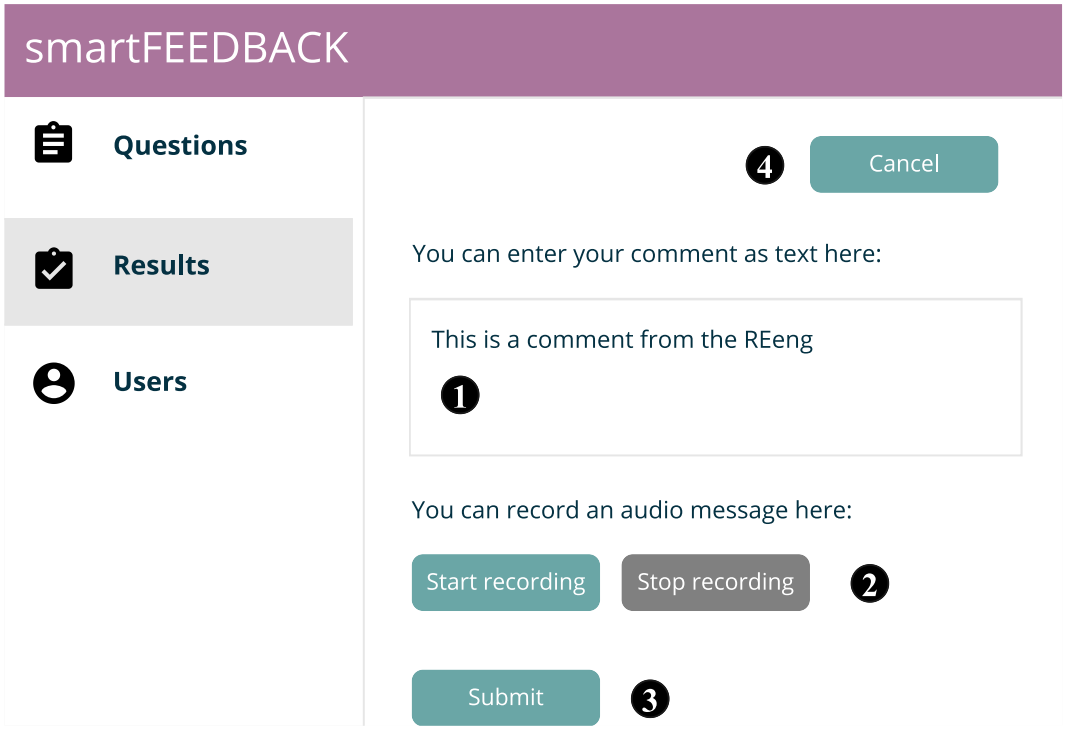
Figure B.5.7: Screenshot for the implemented virtual window
of workspace W: DetailedMessageView (R)



The message details are shown in ❶. The message is displayed with the associated app logo and name, as well as the title of the message and content. If the content is textual, the text is displayed and if it is a voice message, then the voice message is displayed and can be played. The creation date of the message is displayed in the right upper corner. The comments of the message are shown below ❷. If there are no comments, the message "Currently there are no comments" is shown. If there are comments, the comments are shown in the same way as in Figure B.5.6. With the "Sort by" function next to ❷, the REeng can sort the comments by creation date. With ❸ the REeng can create a new comment and with ❹ the REeng can navigate back to Figure B.5.4.

When creating a comment, the screen in Figure B.5.8 is shown. The REeng can enter the comment in the text field of ❶ or as a voice message through ❷. To record a voice message, the "Start recording" button is pressed and after speaking the "Stop recording" button is pressed. The comment can be submitted with ❸ and the REeng can cancel the comment submission through ❹. When the comment submission is canceled, the REeng sees Figure B.5.7 again.

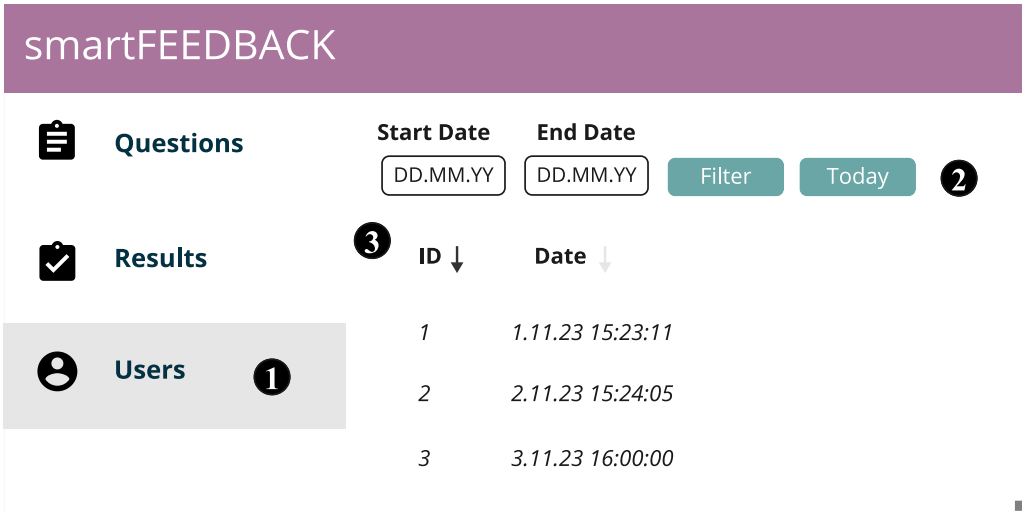
Figure B.5.8: Screenshot for the implemented virtual window of workspace W: CommentView (R)



Viewing users (UT1S3(R))

Figure B.5.9 shows a screenshot for the implemented virtual window of workspace W: UserView (R). The REeng can view this screen by navigating to "Users" 1 in the sidebar.

Figure B.5.9: Screenshot for the implemented virtual window of workspace W: UserView (R)



In ❸ the REeng can see the users of smartFEEDACK with their IDs and their date of first use. The REeng can sort the headers of the table ascending and descending by pressing the arrow next to the header. The REeng can filter the users with ❷. A start date can be entered together with an end date, followed by a press on the button "Filter". This filters the users based on the start date, which has to be between the two dates. With a press on "Today", the REeng can quickly filter for users who start the use of SF at that day.

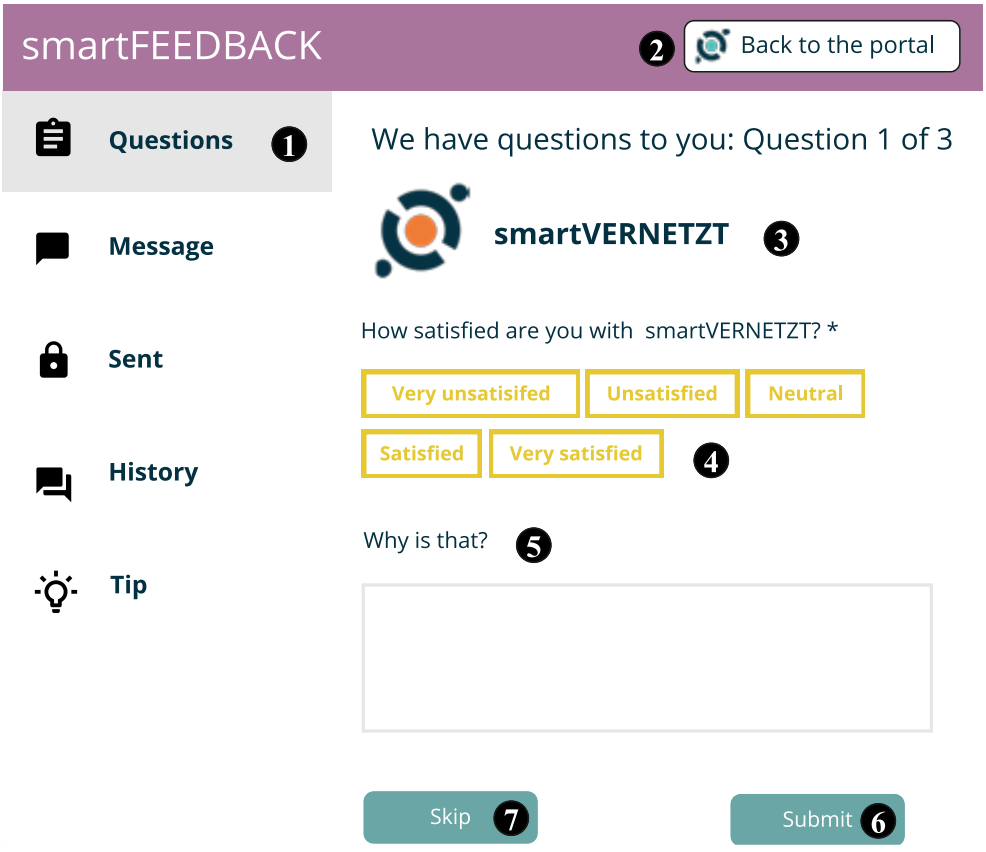
Features for the user

We explain how the user can submit feedback to questions in "Submitting feedback (UT1S1(U))". We describe how the user can manage given feedback in "Managing existing feedback (UT1S2(U))".

Submitting feedback (UT1S1(U))

The user can submit feedback through answers to questions, messages and comments. The submission of answers works through the screen in Figure B.5.10. The user can navigate to this screen through pressing the button "Users" in the sidebar (❶).

Figure B.5.10: Screenshot for the implemented virtual window of workspace W: QuestionView (U)

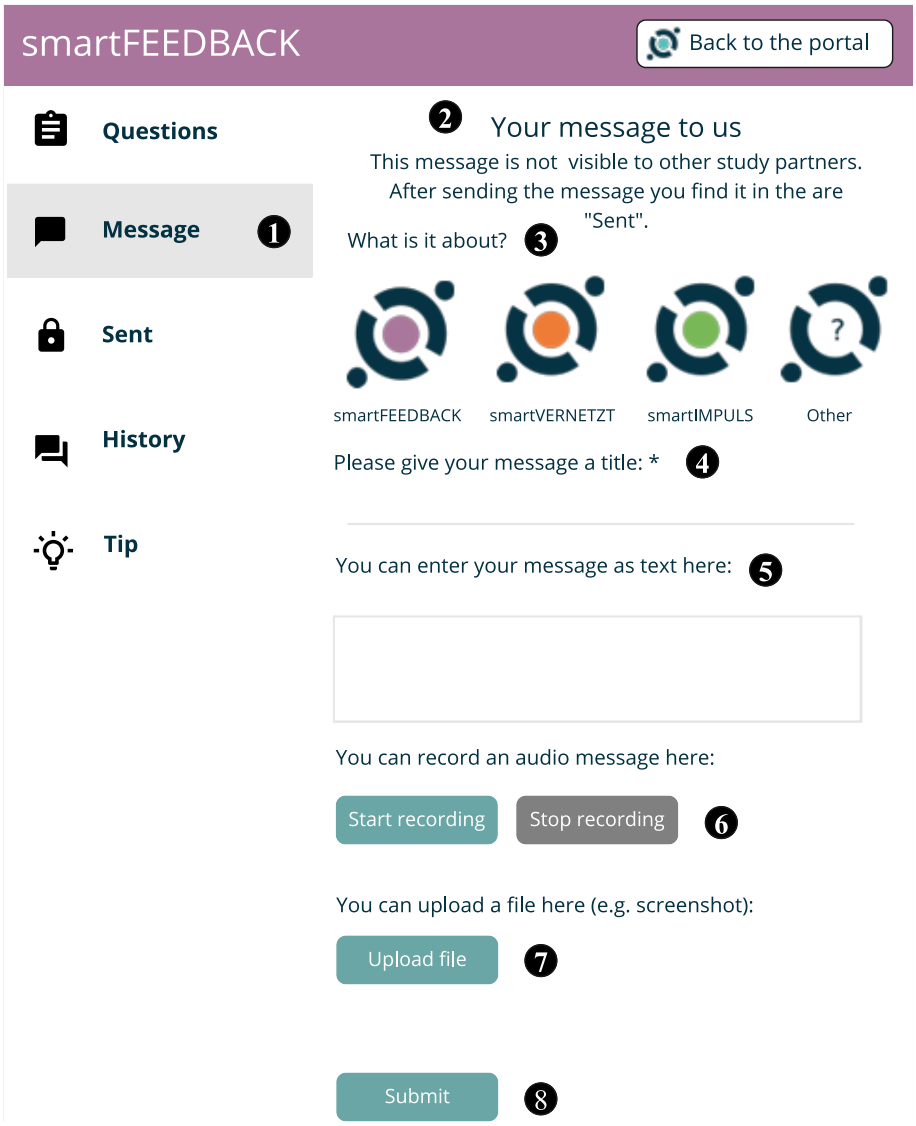


The user sees how many questions are open by the caption "We have questions to you: Question n of N" where n is the current question and N is the number of questions that is open. Each question is displayed individually and is shown with the application logo and name (❸) as well

as with all its subquestions and corresponding answer options (❹, ❺). The user can either skip the answer for a question (❼), which means that no answer is given and the next question is shown or the user addresses the input fields (❹ - selection and ❺ - text) and submit the answer (❻). To submit the answer, the user must at least give answers to questions that are marked with a "*" (e.g. in Figure B.5.10 the user must at least select an answer option for the question "How satisfied are you with SV?" to be able to submit the answer). When no more questions are open, a confetti animation is shown to the user.

The submission of messages is conducted through the screen in Figure B.5.11.

Figure B.5.11: Screenshot for the implemented virtual window of workspace W: MessageView (U)



The user can navigate to the screen by pressing the button "Message" in the sidebar (❶). The caption ❷ serves as an explanation for the user, that the message is not shown to other users that the message can be found by navigating to "Sent" in the sidebar after submission. To submit the message, the user must choose an app or the category "other". The category "other" is used when the message is not about the app SV, SF or SI. The user must enter a title for the message through

❹. The user can provide the message content either through text (❺) or through a voice message (❻). The recording of a voice message works by pressing the "start recording" button and then pressing the "stop recording" button after speaking. The user also has the option to upload a file as an appendix to the message (❼). This is useful when the user needs to provide a screenshot or another file as a reference. The user can submit the message by pressing the "Submit" button (❽). After submission a text appears that the message was sent successfully and the user is presented with the screen of Figure B.5.11 again.

The submission of comments to answers and message is conducted by pressing the "Sent" button in the sidebar and then proceeding exactly like described in Section "Managing given feedback (UT1S2(R))" from the perspective of the REeng.

When the user has problems to remember which of the sidebar buttons are relevant for submitting feedback, he/she can navigate to the screen in Figure B.2.31 in the appendix through the "Tip" button in the sidebar. The screen in the figure explains all sidebar buttons.

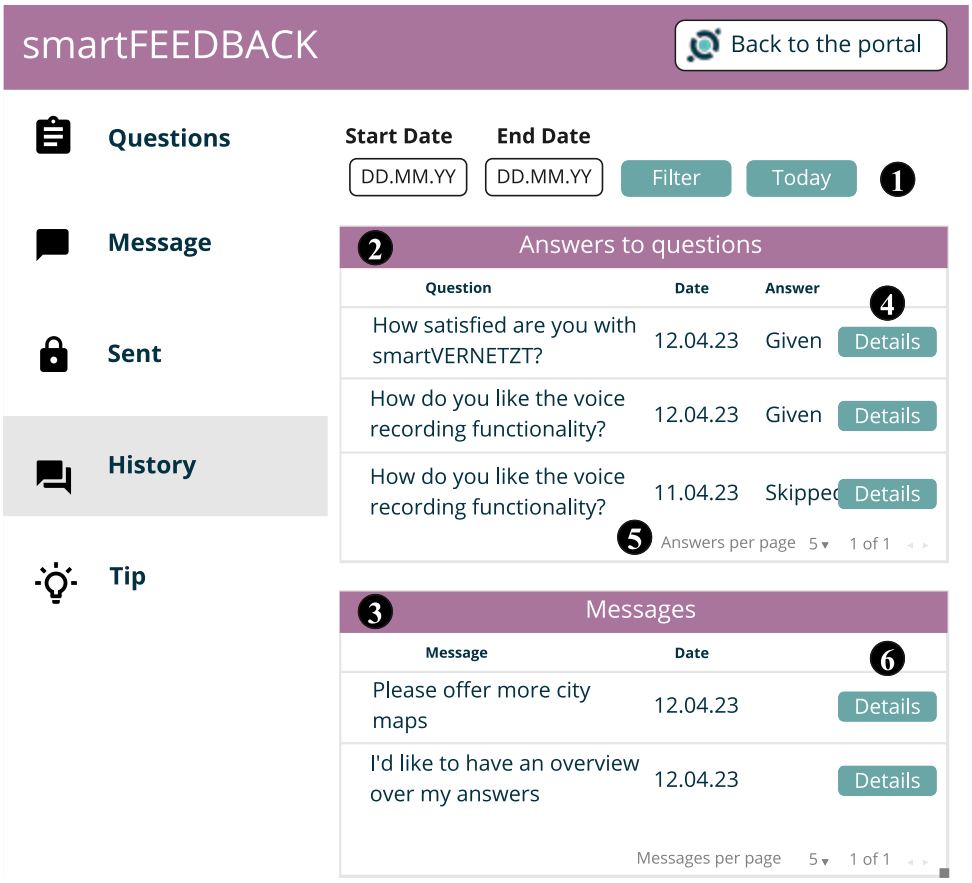
Managing existing feedback (UT1S2(U))

The user manages her/his own feedback through the two links "Sent" and "History" in the sidebar.

The "Sent" button redirects the user to the screen in Figure B.2.23 in the appendix. This screen looks exactly the same for the user and the REeng, with the exception that the screenshot of the REeng shows different sidebar buttons and that the REeng sees questions, answers and messages of all users compared to the user who only sees questions, answers and messages that are relevant for her/him. This is why we do not explain the management of existing feedback through the "Sent" button in this Section, as we already describe this from the perspective of the REeng. To see a complete overview over all screens from the perspective of the user, see Figure B.2.24 in the appendix for a screenshot of the implemented virtual window for W: DetailedQuestionView (U), Figure B.2.26 in the appendix for a screenshot of the implemented virtual window for W: DetailedMessageView (U) and Figure B.2.27 in the appendix for a screenshot of the implemented virtual window for W: CommentView (U).

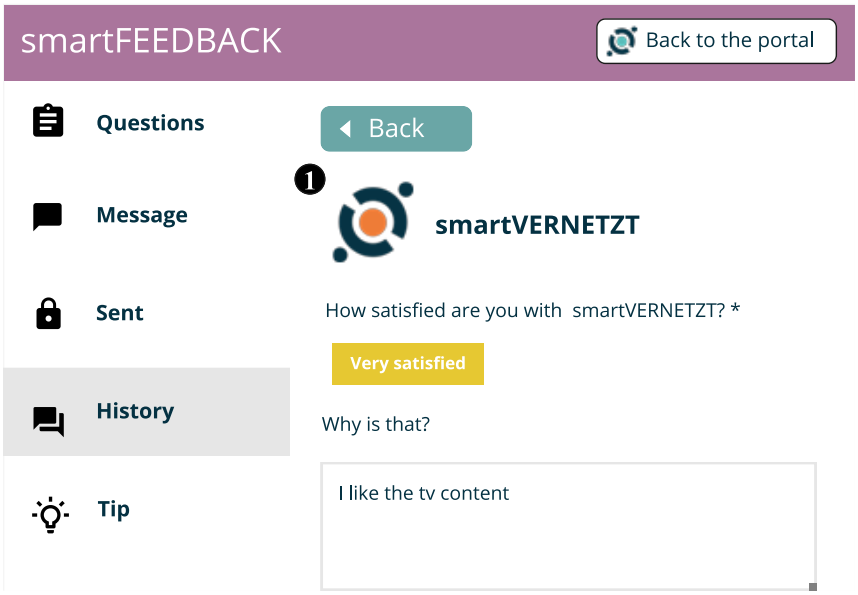
The "History" button redirects the user to the screen in Figure B.5.12. In the history the user sees an overview over all answers (❷) and messages (❸). Compared to the "Sent" screen, where the answers are grouped by questions, in the history the user sees the answers in their chronological order independent of their question. Each answer is listed with the question name, the date when the answer was given or skipped and the info whether the answer was given or skipped. The user sees always five answers and more answers can be loaded through ❺. This is the same for the messages. The user can filter the answers and messages simultaneously through ❶. A start date and end date can be entered and with a press on the button "Filter" the answers and messages are filtered based on their creation date which must be between the start and end date. With a press on the button "Today" the user can quickly filter for answers and messages which where created on that day.

Figure B.5.12: Screenshot for the implemented virtual window of workspace W: HistoryView (U)



The user can view an answer in detail by pressing the button “Details” (4) which takes the user to Figure B.5.13.

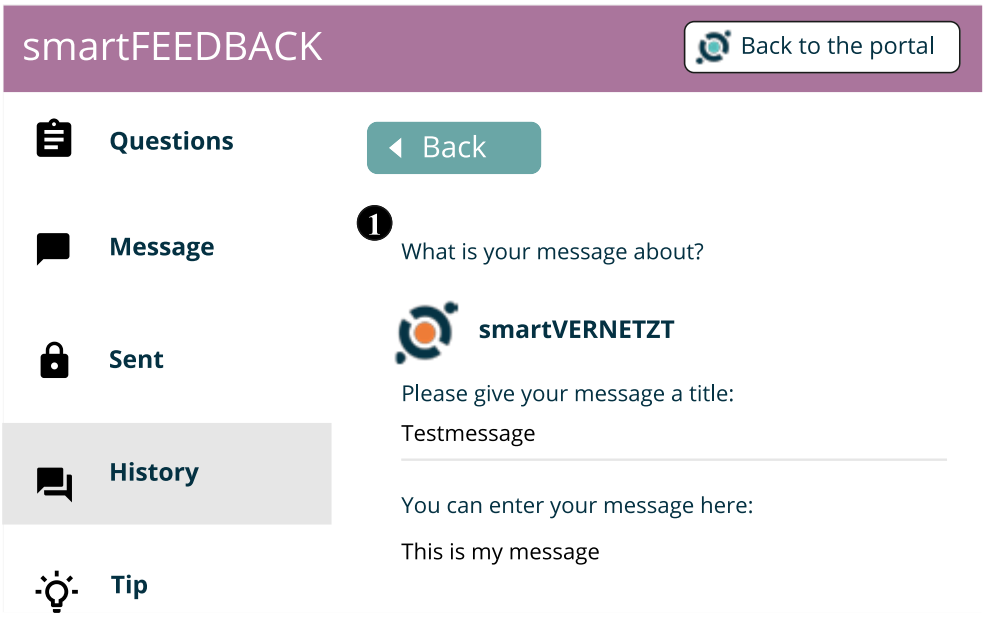
Figure B.5.13: Screenshot for the implemented virtual window of workspace W: DetailedAnswerHistoryView (U)



The user can view the answer details below ❶. The answer is displayed with the application logo and name, as well as with all subquestions their titles and the selection or text that was given by the user. The user can navigate back through the “Back” button above ❶.

The user can view a message in detail by pressing the button “Details” (❻) in Figure B.5.12 which takes the user to Figure B.5.14.

Figure B.5.14: Screenshot for the implemented virtual window of workspace W: DetailedMessageHistoryView (U)



The user can view the message details below ❶. The message is displayed with the application logo and name, as well as with its title and content. The user can navigate back to Figure B.5.12. through the “Back” button above ❶.

C Supplementary material for the treatment validation

C.1 Derived requirements

Table C.1: Derived changes to existing requirements and new requirements

Readable ID	Screenshot	Change	New	Application	Change or new requirement
AV1	Figure B.2.52	X		SV	Change *SF: displayLink* When opening links block advertisements automatically.
AV2	Figure B.2.53		X	SV	Create *SF: filterNews* The user can filter the news by the city “Mannheim” and “Heidelberg”.
AV3	Figure B.2.54		X	SF	Create: *SF: editSubmittedAnswer (U)* The user can edit an already submitted answer.
AV4	Figure B.2.55		X	SV	Create: *SF: customizeStartPage (U)* The user can customize the start page by adding links and applications to it.
AV5	Figure B.2.56	X		SF	Change: *W: HistoryView (U)* The button that allows the navigation to the details of an answer and a message should be visible clearly and not be hidden behind scroll pane.
AV6	Figure B.2.57	X		SF	Change: *W: HeaderView (U)* The button the allows navigation to the portal should be centered.
AV7	Figure B.2.58	X		SV	Change: *SF: displayNews* When displaying news, each news link should direct the user to an individual page where only this news is displayed and no other news.
AV8	Figure B.2.59	X		SF	Change: *W: MessageView (U)* The placement of the audio recording should be above the placement of the text input.

FF3_AV	Figure B.2.62	X		SF	Change: SF: *submitAnswer (U)* When asking a question, there should be an answer option "I never did this", so that the user can signalize that his/her experience is not sufficient to answer the question.
FF4_AV	Figure B.2.63	X		SF	Change: *W: QuestionView (U)* When displaying a question to the user, there should be information about why we ask the question.
FF5_AV	Figure B.2.64	X		SF	Change: *W: MessageView (U)* When submitting a message, no 'childish' symbols such as smileys should be displayed.
FF6_AV	Figure B.2.65	X		SF	Change: *W: SentView (U)* The design of the Sent View should be fitted to the size of the tablet screen.
FF7_AV	Figure B.2.66			P	Change: This change addresses a requirement from the SMART-AGE portal, which we did not define. The change suggests that the email application should be openable from the portal.
FF8_AV	Figure B.2.67	X		SV	Change: *W: HomeView* The news should be displayed in an activity "News" in the activity sidebar.
FF10_AV	Figure B.2.69	X		SV	Change: *W: LinkView* Links that lead to apps should highlighted so that it is clear that the lead to apps.
FF11_AV	Figure B.2.70	X		SV	Change: *W: CategoryView* The symbols of the categories should be colored and not black.
2_AV1	Figure B.2.82	X		SF	Change: *SF: displayQuestion (U)* A question should only be asked once and it should not be repeated in the exact form.
2_AV2	Figure B.2.83	X		SF	Change: *SF: displayQuestion (U)* A question should be asked over email.
2_AV3	Figure B.2.84	X		SF	Change: The *SF: filterHistory (U)* should be removed.
2_AV4	Figure B.2.85		X	SV	Create: *SF: changeOrderOfLinks* The order of the links should be changeable by the user.

2_AV5	Figure B.2.86			P	Change: This change addresses a requirement from the SMART-AGE portal, which we did not define. The portal should allow the users to open both SF and SV simultaneously.
2_AV6	Figure B.2.**87**		X	SF	Create: SF: *markAnswerAsRead (U)* The user can mark an answer as read.
2_AV7	Figure B.2.88	X		SF	Change: W: QuestionView (U) The font size of the questions should be increased.
2_AV8	Figure B.2.89	X		SF	Change: W: MessageView (U) The size of the symbols of the audio recording should be increased.
2_FF1_AV	Figure B.2.90	X		SV	Change: *SF: displayLinks* The number of links in the category "sport" should be increased.
2_FF2_AV	Figure B.2.91	X		SV	Change: *SF: displayWeather* The weather report should include a 10 day forecast.
2_FF3_AV	Figure B.2.92	X		SF	Change: *SF: displayQuestion (U)* There should be no questions that ask about the priority or sentiment of the user.
2_FF5_AV	Figure B.2.**94**		X	SV	Create: *SF: customizeCategories* The user can group links into categories by himself/herself. The position of the categories can also be chosen by the user.
2_FF7_AV	Figure B.2.96	X		SV	Change: *SF: addPersonalLink* A personal link can be added through the browser with one click.
2_FF9_AV	Figure B.2.98	X		SF	Change: *SF: displayInstructions (U)* The instruction text of the app should be shortened.
2_FF11_AV	Figure B.2.99	X		SF	Change: *W: HistoryView (U)* The History View should include the comments of answer and messages.

C.2 Characteristics

Table C.2: Characteristics of users that are collected through questionnaire.

Description	Values
u_age (Age)	
What is your age?	E.g. 70
u_gender (Gender)	

What is your gender?	1: male, 2: female
u_abitur (Education)	
Do you have a high school degree?	0: No high school degree 1: High school degree
u_swe (Self-efficacy (Jerusalem and Schwarzer, 2003)	
Number of questions: 10 Examples: "If I encounter resistance, I find ways and means to assert myself." “I always succeed in solving difficult problems when I put effort into them."	4: Completely agree 3: Somewhat agree 2: Hardly agree 1: Do not agree For one SP, we use the sum score of all answers.
u_mhdt (Media Use/Frequency of Technology Use) Self-designed questionnaire, based on: (Wagner and Zank, 2022)	
Number of questions: 11 Examples: "How often do you use a smartphone?" "How often do you use a... computer, PC, laptop, notebook, or netbook?"	0: Not available / I don’t know 1: Multiple times a day 2: Daily / almost daily 3: At least once a week 4: At least once a month 5: Less often 6: Never For one SP, we use the sum score of all answers.
u_huadi (Frequency and Type of Internet Use) Self-designed questionnaire, based on: (Vogel et al., 2020a)	
Number of questions: 23 The internet offers a variety of usage possibilities. Therefore, we are interested in how often you use the internet for the following purposes: Examples: “Contact with friends, acquaintances, and relatives (e.g., WhatsApp, Telegram, Signal, video calls like Skype)." “Searching for social contacts (e.g., friends, partners, like-minded people)”	0: I don’t use it 1: Daily 2: Several times a week 3: Once a week 4: One to three times a month 5: Less often 6: Never For one SP, we use the sum score of all answers.
u_mdpq (Mobile Device Proficiency Questionnaire (Roque and Boot, 2018))	
Number of questions: 16 When using a mobile device, I can: Examples: “Navigate through the on-screen menus using the	1: Never tried 2: Not at all 3: Not very easy

touchscreen." "Using the on-screen keyboard for typing."	4: Relatively easy 5: Very easy For one SP, we calculate the score according to (Roque and Boot, 2018)
u_techbio (Technology biography (Mollenkopf et al., 2000))	
Number of questions: 7 Examples: "I have always dealt a lot with technology in my life." "I have always been interested in owning the latest technological devices."	1: Does not apply at all 2: Applies slightly 3: Partially applies 4: Mostly applies 5: Applies very well For one SP, we calculate the average score from all answers
u_pus_peu (Perceived Usefulness & Perceived Ease of use) Self-designed questionnaire, based on: (Davis, 1985)	
Number of questions: 6 Examples: • Using IT helps me manage everyday life. • I find it easy to handle IT.	1: Strongly disagree 2: Disagree 3: Rather disagree 4: Neutral 5: Rather agree 6: Agree 7: Strongly agree For one SP, we calculate the average score from all answers
u_intc (Intention to (continue) use) Self-designed questionnaire, based on: (Bhattacherjee and Premkumar, 2004)	
Number of questions: 4 Examples: "I intend to (continue to) use IT in the future." "I assume that I will (continue to) use IT in the future."	1: Strongly disagree 2: Disagree 3: Rather disagree 4: Neutral 5: Rather agree 6: Agree 7: Strongly agree For one SP, we calculate the average score from all answers
u_peen (Perceived Enjoyment) Self-designed questionnaire, based on: (Davis, 1985)	
Number of questions: 3 Examples:	1: Strongly disagree 2: Disagree

"I enjoy using IT." "It is a pleasure for me to use IT."	3: Rather disagree 4: Neutral 5: Rather agree 6: Agree 7: Strongly agree For one SP, we calculate the average score from all answers

Table C.3: User usage characteristics

Variable	Description	Values
Usage time of SF	The usage time of SF in seconds	Value in seconds (e.g. 3600 for 1h)
Usage time of SV	The usage time of SV in seconds	Value in seconds (e.g. 3600 for 1h)
Number of starts of SF	The number of times SF was started	E.g. 100
Number of starts of SV	The number of times SV was started	E.g. 100

BIBLIOGRAPHY

AAPOR 2004. Standard definitions: Final dispositions of case codes and outcome rates for surveys. *American Association for Public Opinion Research.*

Altmeyer et al. 2018. Investigating Gamification for Seniors Aged 75+ *In*: *Proceedings of the Designing Interactive Systems Conference*. ACM, pp.453–458.

Bajic, D. and Lyons, K. 2011. Leveraging social media to gather user feedback for software development *In*: *Proceedings of the International Workshop on Web 2.0 for Software Engineering*. ACM, pp.1–6.

Berger, C. and Blauth, R. 1993. KANO's Methods For Understanding Customer-Defined Quality. *Center for Quality Management Journal*. **4**, pp.3–36.

Bhattacherjee, A. and Premkumar, G. 2004. Understanding Changes in Belief and Attitude Toward Information Technology Usage: A Theoretical Model and Longitudinal Test. *MIS Quarterly*. **28**(2), pp.229–254.

Brennan, R.L. and Prediger, D.J. 1981. Coefficient Kappa: Some Uses, Misuses, and Alternatives. *Educational and Psychological Measurement*. **41**(3), pp.687–699.

Brooke, J. 1995. SUS: A 'Quick and Dirty' Usability Scale. *Usability Evaluation In Industry.*, pp.207–212.

Bührer, M. 2021. *Supporting Social Interaction and Participation of Older Adults Through Technology (Master Thesis)*. Heidelberg University.

Chevalier, J.M. and Buckles, D.J. 2019. *Participatory Action Research: Theory and Methods for Engaged Inquiry* 2nd ed. Routledge.

Covell et al. 2012. Does the sequence of data collection influence participants' responses to closed and open-ended questions? A methodological study. *International Journal of Nursing Studies*. **49**(6), pp.664–71.

Czaja et al 2018. Improving Social Support for Older Adults Through Technology: Findings From the PRISM Randomized Controlled Trial. *Gerontologist*. **58**(3), pp.467–477.

Davis, F.D. 1985. *A technology acceptance model for empirically testing new end-user information systems : theory and results*. Massachusetts Institute of Technology.

El Emam, K. and Koru, G.A. 2008. A replicated survey of IT software project failures. *IEEE Software*. **25**(5), pp.84–90.

Fernandes et al. 2012. IThink : A game-based approach towards improving collaboration and participation in requirement elicitation. *Procedia Computer Science*. **15**, pp.66–77.

Fotrousi et al. 2018. The effects of requests for user feedback on Quality of Experience. *Software Quality Journal*. **26**(2), pp.385–415.

Gartner, S. and Schneider, K. 2012. A method for prioritizing end-user feedback for requirements engineering *In*: *Proceedings of ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. IEEE, pp.47–49.

Glinz et al. 2020. *Handbook for the CPRE Foundation Level according to the IREB Standard Education and Training for Certified Professional for Requirements Engineering (CPRE) Foundation Level Terms of Use*. IREB.

Groen et al. 2017. The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software*. **34**(2), pp.44–52.

Guzman et al. 2016. A Needle in a Haystack: What Do Twitter Users Say about Software? *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.96–105.

Haug et al. 2023. Scalable Design Evaluation for Everyone! Designing Configuration Systems for Crowd-Feedback Request Generation *In*: *Proceedings of Mensch und Computer*. Association for Computing Machinery, pp.91–100.

Jakobs, R. 2021. *Health assessment and recommendation for older adults (Master Thesis)*. Heidelberg University.

Jerusalem, M. and Schwarzer, R. 2003. Skala zur Allgemeinen Selbstwirksamkeitserwartung. *Open Test Archive*.

Johanssen et al. 2019. How do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering ? *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.153–164.

Kabbedijk et al. 2009. Customer Involvement in Requirements Management: Lessons from Mass Market Software Development *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.281–286.

Kifetew et al. 2021. Automating user-feedback driven requirements prioritization. *Information and Software Technology*. **138**(1), p.106635.

Kolpondinos and Glinz 2020. GARUSO: a gamification approach for involving stakeholders outside organizational reach in requirements engineering. *Requirements Engineering*. **25**(2), pp.185–212.

Kujala, S. 2008. Effective user involvement in product development by improving the analysis of user needs. *Behaviour & Information Technology*. **27**(6), pp.457–473.

Lai et al. 2014. A collaborative method for business process oriented requirements acquisition and refining *In*: *Proceedings of the International Conference on Software and System Process*. ACM, pp.84–93.

Laporti et al. 2009. Athena: A collaborative approach to requirements elicitation. *Computers in Industry*. **60**(6), pp.367–380.

Li et al. 2024. Unveiling the Life Cycle of User Feedback : Best Practices from Software Practitioners *In*: *Proceedings of the International Conference on Software Engineering*. ACM, pp.1–13.

Lim et al. 2021. Data-Driven Requirements Elicitation: A Systematic Literature Review. *SN Computer Science*. **2**(1).

Lohmann et al. 2009. A Web Platform for Social Requirements Engineering. *Software Engineering Workshopband (GI).*, pp.309–315.

Maalej et al. 2009. When users become collaborators *In*: *Proceedings of the SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM, pp.981–990.

Malgaonkar et al. 2022. Prioritizing user concerns in app reviews – A study of requests for new features, enhancements and bug fixes. *Information and Software Technology*. **144**, p.106798.

Memmer et al. 2024a. Pilot Testing of a German Version of the 'PRISM' App to Promote Social

Participation in Aging: Initial Implementation. *Zeitschrift für Medien und Altern*.

Memmer et al. 2024b. SMART-AGE Study Protocol: A Complex Intervention to Increase Social Participation, Physical Fitness and Health Awareness Among Older Adults. *BMC Trials*.

Menkveld et al. 2019. User story writing in crowd requirements engineering: The case of a web application for sports tournament planning *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.174–179.

Mollenkopf et al. 2000. Technik im Haushalt zur Unterstützung einer selbstbestimmten Lebensführung im Alter. *Zeitschrift fur Gerontologie und Geriatrie*. **33**(3), pp.155–168.

Van Oordt, S. and Guzman, E. 2021. On the Role of User Feedback in Software Evolution: A Practitioners' Perspective *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.221–232.

Oriol et al. 2018. FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.217–227.

Paech, B. and Kohler, K. 2004. Task-Driven Requirements in Object-Oriented Development *In*: *Perspectives on Software Requirements*. Springer, pp.45–67.

Panichella et al. 2015. How can I improve my app? Classifying user reviews for software maintenance and evolution *In*: *Proceedings of the International Conference on Software Maintenance*. IEEE.

Radeck et al. 2022. Understanding IT-related Well-being, Aging and Health Needs of Older Adults with Crowd-Requirements Engineering *In*: *Proceedings of the IEEE International Conference on Requirements Engineering*. IEEE, pp.57–64.

Radeck, L. and Paech, B. 2024. Channeling the Voice of the Crowd: Applying Structured Queries in User Feedback Collection *In*: *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer Nature Switzerland, pp.284–301.

Radeck, L. and Paech, B. 2023. Integrating Implicit Feedback into Crowd Requirements Engineering – A Research Preview *In*: *Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality*. ACM, pp.283–292.

Renzel et al. 2013. Requirements Bazaar: Social requirements engineering for community-driven innovation *In*: *Proceedings of the International Requirements Engineering Conference*. IEEE, pp.326–327.

Rietz, T. 2019. Designing a conversational requirements elicitation system for end-users *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.452–457.

Roque, N.A. and Boot, W.R. 2018. A New Tool for Assessing Mobile Device Proficiency in Older Adults: The Mobile Device Proficiency Questionnaire. *Journal of Applied Gerontology*. **37**(2), pp.131–156.

Saphira, M. and Rusli, A. 2019. Towards a gamified support tool for requirements gathering in Bahasa Indonesia *In*: *Proceedings of the International Conference on New Media Studies*. IEEE, pp.201–206.

Sardi, L., Idri, A. and Fernández-Alemán, J.L. 2017. A systematic review of gamification in e-Health. *Journal of Biomedical Informatics*. **71**, pp.31–48.

Sauro, J. and Lewis, J.R. 2012. *Quantifying the User Experience: Practical Statistics for User Research*.

Morgan Kaufmann.

Scherbatschenko, D. 2023. *Ableitung von Anforderungen aus Nutzerfeedback (Master Thesis)*. Heidelberg University.

Seyff et al. 2010. End-user requirements blogging with iRequire *In*: *Proceedings of the International Conference on Software Engineering*., pp.285–288.

Sharma, R. and Sureka, A. 2018. CRUISE: A platform for crowdsourcing Requirements Elicitation and evolution *In*: *Proceedings of the International Conference on Contemporary Computing*. IEEE, pp.1–7.

Sheskin, D. 2004. Handbook of Parametric and Nonparametric Statistical Procedures. *Technometrics*. **46**(3), pp.369–370.

Shih, T.H. and Xitao, F. 2008. Comparing response rates from web and mail surveys: A meta-analysis. *Field Methods*. **20**(3), pp.249–271.

Snijders et al. 2015. REfine: A gamified platform for participatory requirements engineering *In*: *Proceedings of the International Workshop on Crowd-Based Requirements Engineering*. IEEE, pp.1–6.

Stade et al. 2017. Providing a user forum is not enough: First experiences of a software company with CrowdRE *In*: *Proceedings of the International Requirements Engineering Conference Workshops*. IEEE, pp.164–169.

Stanmore, E. 2021. Developing, Testing, and Implementing a Falls Prevention and Healthy Aging App (Keep-On-Keep-Up) for Older Adults. *Innovation in Aging*. **5**, p.514.

Tizard et al. 2020. Voice of the Users: A Demographic Study of Software Feedback Behaviour *In*: *Proceedings of the International Conference on Requirements Engineering*. IEEE, pp.55–65.

Tizard et al. 2021. Voice of the users: an extended study of software feedback engagement. *Requirements Engineering*. **27**, pp.293–315.

Vijayan et al. 2017. Collaborative requirements elicitation using elicitation tool for small projects *In*: *Proceedings of the International Conference on Signal Processing, Communication, Power and Embedded System*. IEEE, pp.340–344.

Vogel et al. 2020a. German Ageing Survey (DEAS). *Encyclopedia of Gerontology and Population Aging*., pp.1–9.

Vogel et al. 2020b. Leveraging the internal crowd for continuous requirements engineering - Lessons learned from a design science research project *In*: *Proceedings of the European Conference on Information Systems*. AIS eLibrary.

Wagner, M. and Zank, S. 2022. Abschlussbericht: Lebensqualität und Wohlbefinden hochaltriger Menschen in NRW (Folgebefragung NRW80+). *Cologne Center for Ethics, Rights, Economics, and Social Sciences of Health*.

Wehrmaker, T., Gärtner, S. and Schneider, K. 2012. ConTexter feedback system *In*: *Proceedings of the International Conference on Software Engineering*. IEEE, pp.1459–1460.

Wieringa, R.J. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer.

Wouters et al. 2022. Crowd-based requirements elicitation via pull feedback: method and case studies. *Requirements Engineering*. **27**, pp.429–455.

Wouters et al. 2021. CrowdRE in a Governmental Setting : Lessons from Two Case Studies *In*:

Proceedings of the International Conference on Requirements Engineering. IEEE, pp.312–322.

Wüest et al. 2019. Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use *In*: *Proceedings of the International Conference on Requirements Engineering: Foundation for Software Quality*. Springer, pp.209–225.

Xu, X., Du, H. and Lian, Z. 2022. Discussion on regression analysis with small determination coefficient in human-environment researches. *Indoor Air*. **32**(10), pp.1–13.

Yang et al. 2008. WikiWinWin: A Wiki based system for collaborative requirements negotiation *In*: *Proceedings of the International Conference on System Sciences*. IEEE, pp.1–10.

INDEX OF TABLES

INDEX OF FIGURES

INDEX OF LISTINGS