Integrating Implicit Feedback into Crowd Requirements Engineering – A Research Preview

Leon Radeck(✉) and Barbara Paech

Institute for Computer Science, Heidelberg University, 69120 Heidelberg, Germany
{radeck,paech}@informatik.uni-heidelberg.de

Abstract. [Context/Motivation] In crowd requirements engineering, users are asked specific questions (explicit pull feedback) to elicit requirements. Existing approaches collect explicit pull feedback by asking the same questions to all users. **[Problem]** Not all questions are meaningful for all users, e.g. regarding a functionality they have not yet used. Furthermore, without knowing the user behaviour giving rise to the feedback, it is difficult to understand the reasons for the feedback. These reasons are important for deriving requirements. **[Principal ideas]** Our idea is to use the user behaviour (implicit feedback) to adapt the collection of explicit pull feedback and the derivation of requirements. We embed this collection of explicit pull feedback into a novel approach that makes use of a rich palette of discussion elements from crowd-based requirements engineering to motivate user participation and to support requirements derivation. **[Contribution]**. To our best knowledge, this is the first approach that combines the collection of implicit feedback and explicit feedback with discussion elements from crowd-based requirements engineering. We sketch our approach and our research and evaluation plan regarding the application of the approach in the context of the interdisciplinary and large-scale research project SMART-AGE with around 500 users.

Keywords: Requirements engineering · Crowd · User feedback · Implicit feedback

1 Introduction

User feedback is essential for the continuous development of software, because it contributes substantially to the elicitation of requirements. Traditional methods of collecting user feedback, such as interviews or workshops, are only feasible with a limited number of users as they are very time-consuming. As the number of users increases, the use of (semi-) automated methods becomes more relevant. These methods do not require the presence of the persons involved, but can be performed remotely and by many stakeholders at the same time [5]. Crowd-based requirements engineering (CrowdRE) is an umbrella term for such approaches to gather and analyse feedback from a large number of users, also called "crowd", to derive validated user requirements [4]. Collecting

A. Ferrari and B. Penzenstadler (Eds.): REFSQ 2023, LNCS 13975, pp. 283–292, 2023.
https://doi.org/10.1007/978-3-031-29786-1_20

explicit push feedback (feedback intentionally pushed by the user) is one main approach in CrowdRE [15]. Another main approach in CrowdRE is to request users to give explicit pull feedback about the product by asking them questions (e.g. "How satisfied are you with product?", "What are your ideas on how to improve the product?") [15]. The problem is that not every question is always suitable for every user. Users who receive a question about a functionality that they have not used yet, cannot answer the question. In addition, it is difficult for requirements engineers to understand the reasons for the feedback without knowing the user behaviour, which led to the feedback. Our idea is to integrate the capture and analysis of the user behaviour (implicit feedback) into CrowdRE to adapt the collection of explicit feedback and the derivation of requirements.

In this paper, we present our approach CREII (Crowd-based Elicitation with Integrating Implicit Feedback) that employs the tool SMARTFEEDBACK (SF) to tailor the collection of explicit pull feedback to an individual user's usage behaviour. To motivate user participation and to allow for requirements derivation, SF further integrates various established discussion elements from CrowdRE [15]. Users can comment and vote answers in real-time by the use of a discussion platform, they can classify and prioritize explicit feedback, they can indicate a representative sentiment for their feedback and they can discuss about requirements (see Table 1). We also describe the application of CREII in the context of the interdisciplinary and large-scale research project SMART-AGE. In particular, we describe preliminary research in the form of a pilot test, the research plan for the employment of CREII in the main study and our ideas for evaluation. This paper is organized as follows: Sect. 2 gives a brief overview of SMART-AGE and the relevant terminology. Section 3 presents related work. Section 4 describes our approach, and Sect. 5 presents how the approach is applied in SMART-AGE, in particular preliminary research, our research and evaluation plan.

2 Project SMART-AGE and Terminology

SMART-AGE. Recent findings on the role and potential of apps for older adults' quality of life are encouraging. Major examples for these apps are solutions that address social engagement and networking, health and disease prevention, and training and fitness. The 5-year project "Smart Aging in Community Contexts: Testing Intelligent Assistive Systems for Self-regulation and Co-regulation under Real-Life Conditions" (SMART-AGE) is in its core a complex intervention trial aimed at evaluating different constellations of apps. These apps are used by around 500 study participants from two communities in Southwestern Germany (Heidelberg and Mannheim). In this project, we gather feedback for three tablet-based apps, namely (1) an app promoting social networking and social participation (SMARTVERNETZT), (2) an app providing health advice focusing on major areas of older adults' health and functioning (SMARTIMPULS) and, (3) an app to tailor the collection of explicit pull feedback to an individual user's usage behaviour (SMARTFEEDBACK). The gathered user feedback is privacy relevant, as both, voice messages in the explicit feedback and interaction data in the implicit feedback may allow identifying a person. Therefore, we obtain declarations of consent from the study participants. We describe our vision of a CrowdRE process adapted to the needs of older adults and the challenges in implementing our approach in the context of SMART-AGE in more detail in [11].

Terminology. In the following, we distinguish 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*) [8]. Explicit push feedback are either messages or comments. Explicit pull feedback are answers to questions. Implicit pull feedback are usage data (e.g. clicks on user interface). *Private* feedback are messages and answers from the users, which are only visible to the researchers. *Shared* feedback are answers from the users that are visible to everyone. Implicit push feedback (e.g. comments made by users about the apps during a conversation, that is not happening over SF) is not considered in our approach, as users are instructed to give their feedback via SF only.

3 Related Work

According to the mapping study [13], in approaches up to 2017 mainly explicit feedback is collected and used to derive requirements. Approaches that have collected implicit feedback have used it to estimate service performance or to compute user profiles [13] but not to combine it with explicit feedback. There are more recent approaches that are not included in the study, but combine explicit and implicit feedback. CAFE [2] enriches explicit push feedback with implicit feedback to facilitate the interpretation of the explicit push feedback. CAFE shows the requirements engineer what screens the user visited, what UI elements the user interacted with prior to giving feedback, as well as hardware information (e.g. operation system and device model). CAFE, however, does not give examples on how exactly the interpretation of explicit push feedback is facilitated by the enrichment with implicit feedback. FAME [9] combines implicit and explicit push and pull feedback through an ontology. The ontology links explicit push and pull and implicit feedback by user, timestamp, application and domain concept. The ontology is presented to the requirements engineer, which allows the requirements engineer to get a better understanding of the explicit push and pull feedback and to prioritize requirements derived from the feedback FAME does not use implicit feedback to adapt pull feedback questions. QoE probe [3] and Wuest et al. [16] introduce explicit pull feedback based on implicit feedback. 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. Wuest et al. [16] triggers feedback collection based on user goals in the context of a navigation system. An example for a goal is that the user reached her target destination. This is automatically recognized in the implicit feedback, when the users GPS coordinates match those of the destination, and then explicit pull feedback through a feedback form is triggered. We adapt this in CREII, where we trigger explicit pull feedback when the user does not behave according to the defined ideal usage behaviour. While all of the mentioned approaches combine implicit and explicit feedback, none of them integrates discussion elements from crowd-based requirements engineering (see Table 1) to support user participation and requirements derivation.

4 Crowd-Based Requirements Elicitation Via the CREII Method

In the following, we describe our approach CREII (Crowd-based Elicitation Integrating Implicit Feedback) that employs the tool SF to collect feedback and supports the derivation of requirements. The process is shown in Fig. 1. We first give a brief explanation of Fig. 1 and then we describe the collection of pull feedback based on usage behaviour by using adaptive questions (Sect. 4.1). After that, we describe our plan for bundling similar feedback (Sect. 4.2) and the derivation of requirements (Sect. 4.2) in more detail. An explanation of the steps of CREII with examples is given in Table 1.

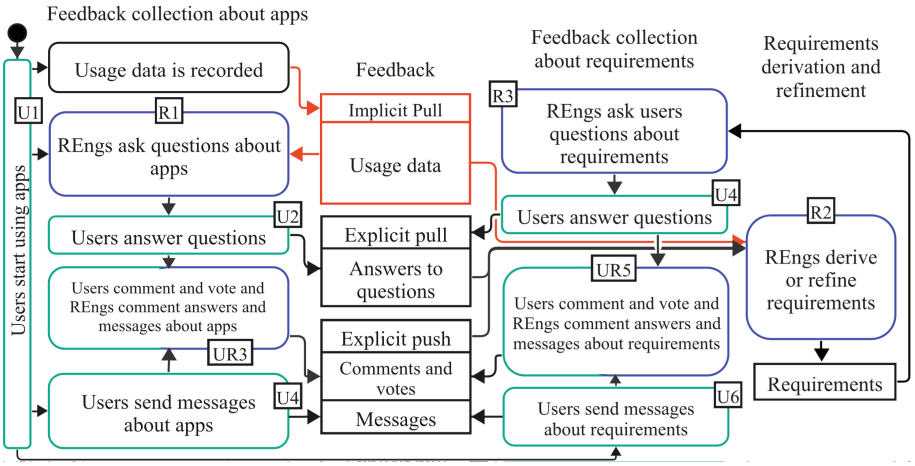

Fig. 1. Diagram representing the process of collecting feedback about apps and requirements, as well as requirements derivation and refinement. Activities in green are executed by users, activities in blue are executed by requirements engineers. Implicit pull feedback and associated arrows are red. Everything else is black.

After the users begin to use the apps (U1), they can provide feedback about the apps by sending messages to the requirements engineers (U4) or they can answer questions about the apps (U2), that the requirements engineers have asked (R1). The requirements engineers ask different types of questions. They ask questions to collect opinions, improvement ideas and problems with the apps and they pose *adaptive questions* that ask for reasons for the observed implicit feedback of the users and corresponding improvement ideas (see Sect. 4.1). The questions can address the app itself, as well as functional and non-functional requirements of the app. After the users have answered questions about the apps, they can comment and vote other answers to the same questions, as far as other users shared their answers (UR3). We stipulate that this discussion possibility enhances the motivation to the user to give feedback. The requirements engineers can also comment the answers to questions and messages about apps to thank the users for their feedback and to ask for clarification (UR3). The users can comment back on messages that they sent (UR3). The requirements engineers do not vote on answers, because they do not want to influence the opinion of the users. Furthermore, we explicitly do not allow users to comment on answers of questions that they did not answer

by themselves, because we believe a minimum level of commitment is necessary to have a meaningful discussion. Based on the collected explicit and implicit feedback, the requirements engineers derive and refine requirements (R2). These requirements are presented to the users and users are asked about their opinions (R3). Implicit feedback is used during the derivation, to make sure that users receive only questions about requirements for apps, that they have accessed. To support requirements derivation, users can answer the questions about the requirements (U4), send messages about the requirements to the requirements engineers (U6), and comment and vote on the requirements. The requirements engineers can comment answers and messages about requirements in the same way as answers and messages about apps (UR5). Based on the feedback about the requirements, the requirements engineers derive new requirements and refine existing requirements (R2). Whenever users give feedback, they have the opportunity to assign a priority (low, medium, high) to the feedback and a sentiment (very dissatisfied, dissatisfied, neutral, satisfied, very satisfied). These two attributes can be used by the requirements engineers to bundle feedback (see Sect. 4.2). The users can also indicate whether to share the feedback with other users or whether to give the feedback privately to the researchers. This is important because certain users place high value on privacy [12].

4.1 Collecting Pull Feedback by Using Adaptive Questions

Adaptive questions ask for reasons for the observed implicit feedback of the users and corresponding improvement ideas. The process of asking adaptive questions for the app SMARTIMPULS is illustrated in Fig. 2.

The user with user id "User1" starts interacting with the app (A). The resulting implicit feedback is sent to SF (B). 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 the answer) and the data to which the event was created (*Created*). SF receives the implicit feedback and saves it to the database (C). SF now periodically loads the history of the implicit feedback (D) and checks whether it does not represent the ideal usage behaviour of SMARTIMPULS (E). The ideal usage behaviour is configured initially by the requirements engineers. It consists of different metrics, e.g. the ideal usage frequency of the app, the ideal usage duration of the app and the ideal answer rate to questions, as in SMARTIMPULS the user has to answer certain questions about his or her health. Checking for ideal usage behaviour means calculating the metrics based on the history of implicit feedback (e.g. accumulating the time difference between START and STOP events per day, to get the usage duration per day) and then comparing it to the expected ideal behaviour (10 min per day). The check happens once per day. If the implicit feedback does not represent ideal usage behaviour, the user receives an adaptive question (F), which asks for the reason and for improvement ideas. Adaptive questions are only asked again after some time has passed, so that the user does not feel disturbed. When collecting pull feedback with adaptive questions (for an example, see Q2 in Table 1), we combine asking for the reason of a users' usage behaviour with asking for an improvement idea, because knowing the

Table 1. Explanation of steps of CREII with examples

Step	Step description (discussion elements are underlined)	Action (Q = Question, A = Answer, C = Comment, M = Message)
R1	Requirements engineers ask questions	*Q1*: How could SMARTIMPULS be improved in your opinion? *Q2* (adaptive): Why do you not use SMARTIMPULS every day"
U2	Users answer questions	*Q1A1*: It would help to be reminded to answer questions *Q2A1*: I find the questions not suitable for me
UR3	Users comment and vote	Some users vote for *Q1A1*, some users vote for *Q2A1* *Q1A1C1*: "I'd like to be reminded every day."
UR3	Requirements engineers comment	*Q1A1C2*: "Thank you very much for your input." *Q2A1C1*: "Why are the questions not suitable for you?
U4	Users send messages	*M1*: The letters of the app are too small for me to read
R2	Requirements engineers derive and refine requirements	From *Q1A1* a new system function "Remind user to answer questions" is extracted. By *Q1A1C1* the SF is detailed by adding a rule about the frequency of reminding. *M1* details the NFR Accessibility
R3	Requirements engineers ask questions about requirements	To validate the SF, the question *Q3* "How would you like a new functionality in SMARTIMPULS that reminds you every day to answer questions? Please explain your judgement?" is asked To validate *M1*, the question *Q4* "How would you like an increased font size in SMARTIMPULS? Please explain your judgement?" is asked
U4	Users answer questions	*Q3A1*: "I would love that, because I am a bit forgetful." *Q3A2*: "I am a bit sceptic." *Q4A1*: "That's a good idea, then I use the app without my glasses."
UR5	Users comment and vote	Some users vote for *Q3A1*, some user vote for *Q3A2* *Q3A2C1*: "Me too, I don't know if that helps"
UR5	Requirements engineers comment	*Q3A2C2*: "Thanks for giving feedback. Why are you sceptic?"
U6	Users send messages	*M2*: I don't want to tell it publicly, but I think reminders about answering questions would stress me."
…		

reason alone might not be enough to derive a requirement. Adaptive questions can avoid asking users for feedback before the users have gained minimal experience with the app

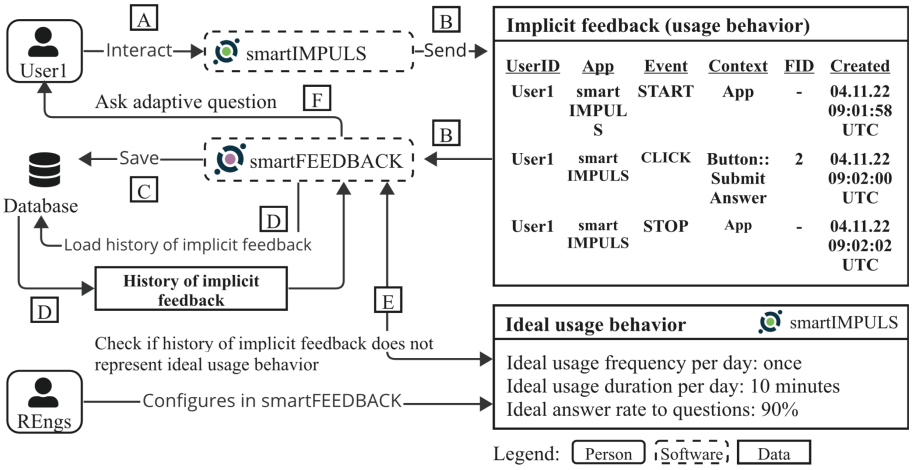

Fig. 2. Diagram representing the process of asking adaptive questions.

or functionality, which would be disturbing [15]. In SMART-AGE, each user joins the study at a different time. We give the users a few days to get familiar with the apps and then start asking the questions relative to the users' start date.

4.2 Bundling of Explicit Feedback

The more explicit feedback is collected, the greater the likelihood that feedback will be similar. To save effort, we bundle similar feedback on the basis of its attributes before deriving requirements. Table 2 shows attributes that we deem useful for bundling.

Table 2. Attributes of feedback used for bundling (R = Specified by requirements engineer, U = Specified by user, A = Automatically recorded by SF)

R	U	A	Attributes
X			Task Oriented Requirements Engineering (TORE) Category [10] (Goal & Task, Domain, Interaction, System Level)
X			Degree of readability (measured by different common readability formulas)
	X		Category (improvement, problem, opinion, neutral)
	X		Sentiment (very happy, happy, neutral, sad, very sad)
	X		Priority (low, normal, high)
		X	Implicit feedback (usage behaviour)

We think that TORE [10] could help with bundling, because feedback can be grouped by different levels. We think that the degree of readability of feedback is helpful for bundling, because unreadable feedback can contribute less to requirements derivation.

We also think that the category of feedback plays a role for requirements derivation. Feedback of the category improvement could have more potential for deriving requirements than feedback which is classified as an opinion. Feedback can also be bundled by sentiment and priority. Feedback with low happiness and an indication of the reason could be especially useful for improving the apps. Feedback that was assigned a high priority by the user is more relevant for requirements derivation than feedback with low priority. We also think that implicit feedback can be helpful to bundle feedback. For example, feedback can be bundled by the users' usage time.

5 Application of CREII in SMART-AGE

Preliminary Research. We performed a convincing proof-of-concept version of CREII in a pilot test with 20 participants over the period of one week. Overall, 208 responses to 24 questions and 33 messages were sent. Feedback for SF was very positive.

Research Plan. We follow the design science methodology proposed by Wieringa [14]. We plan to deploy CREII in SMART-AGE to collect feedback and derive requirements for the apps from around 500 users in the first five months of the study and then follow up with one month of requirement validation and refinement. For the evaluation of CREII, we also ask questions about the acceptance of SF by using the System Usability Scale [1]. In order not to overwhelm the users, we plan to ask them no more than five questions on the same day. As a reminder and for motivation, we also plan to remind users after a week of inactivity to participate in the feedback collection again.

Evaluation. Table 3 shows our research questions.

Table 3. Research questions

Usage behaviour	
RQ1	Does the usage behaviour of a user influence the quality or quantity of the provided feedback?
Feasibility	
RQ2	Is it feasible to collect high quality feedback with CREII and SF?
RQ2.1	What is the quantity and quality of the collected feedback?
RQ2.2	Do adaptive questions and discussion elements influence the quality or quantity of the provided feedback?
RQ2.3	Does reminding to give feedback influence the quality or quantity of feedback?
RQ3	Is it feasible to collect high quality requirements with CREII and SF?
RQ3.1	What is the quantity and quality of the derived requirements?
Acceptance	
RQ4	What is the acceptance of SF?

RQ1 investigates whether the usage behaviour of a user influences the quality or quantity of feedback. For example, high usage duration could lead to feedback of higher quality. RQ2 and RQ3 investigate the feasibility of CREII and SF to collect high quality feedback and high quality requirements. RQ2.1 investigates the quantity and quality of the collected feedback and RQ3.1 investigates the quantity and quality of the derived requirements. Investigating the quantity and quality of derived requirements allows us to make a comparison to similar approaches [15]. RQ2.2 investigates the influence of adaptive questions or discussion elements on the quality and quantity of feedback. If adaptive questions or discussion elements have a positive influence on the quality and quantity of feedback, then it would make sense for others to implement this practice as well. RQ2.3 investigates whether reminding to give feedback affects its quality of quantity. Reminding is easy to implement and would be an implementable practice for others. RQ4 evaluates the acceptance of SF through the questions of the System Usability Scale [1].

Quality of Requirements. We assess the quality of requirements manually. We use established quality criteria from the International Requirements Engineering Board (IREB) manual [7], such as Adequacy, Necessity, Unambiguity, Completeness, Understandability, Verifiability, Consistency, Redundancy.

Quality of Feedback. We assess the quality of feedback manually. To our best knowledge there does not exist an established set of quality aspects for user feedback about software. We therefore derive quality aspects from established standards. We plan to adapt the characteristics for data quality of ISO 25012 [6] and the quality criteria for requirements (IREB) on feedback and to establish metrics that let us quantify each quality aspect manually.

Limitations and Risks. Eliciting pull feedback requires the users to answer the questions of us researchers. This can be time-consuming and strenuous, especially for elderly people. In [11] we discuss how CREII is tailored to the individual needs of older adults.

Acknowledgement. We thank the Carl Zeiss Foundation for the generous 5-year funding of SMART-AGE (P2019-01-003; 2021–2026).

References

1. Brooke, J.: SUS - a quick and dirty usability scale. In: Jordan, P.W., Thomas, B., Ian Lyall, M., Bernard, W. (eds.) Usability Evaluation in Industry, pp. 207–212 (1996)
2. Dzvonyar, D., Krusche, S., Alkadhi, R., Bruegge, B.: Context-aware user feedback in continuous software evolution. In: Proceedings of the International Workshop on Continuous Software Evolution and Delivery, CSED 2016, pp. 12–18 (2016).https://doi.org/10.1145/2896941.2896952
3. Fotrousi, F., Fricker, S.A.: QoE probe: a requirement-monitoring tool. CEUR Workshop Proc. **1564**, 7–8 (2016)
4. Groen, E.C., et al.: The crowd in requirements engineering: the landscape and challenges. IEEE Softw. **34**, 44–52 (2017). https://doi.org/10.1109/MS.2017.33

5. Groen, E.C.: How Requirements Engineering can benefit from crowds. Requirements Eng. Mag., 1–13 (2016)
6. International Organization for Standardization/International Electrotechnical Commission: Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Data quality model ISO/IEC 25012:2008(E) (2008)
7. IREB: Certified Professional for Requirements Engineering – Foundation Level. Karlsruhe, Germany (2015)
8. Maalej, W., Happel, H.-J., Rashid, A.: When users become collaborators: towards continuous and context-aware user input. In: International Conference OOPSLA, pp. 981–990. ACM (2009). https://doi.org/10.1145/1639950.1640068
9. Oriol, M., et al.: FAME: supporting continuous requirements elicitation by combining user feedback and monitoring. In: IEEE Requirements Engineering Conference (RE), pp. 217–227. IEEE (2018). https://doi.org/10.1109/RE.2018.00030
10. Paech, B., Kohler, K.: Task-driven requirements in object-oriented development. In: do Prado Leite, J.C.S., Doorn, J.H. (eds.) Perspectives on Software Requirements. The Springer International Series in Engineering and Computer Science, vol. 753. Springer, Boston, MA (2004). https://doi.org/10.1007/978-1-4615-0465-8_3
11. Radeck, L., et al.: Understanding IT-related well-being, aging and health needs of older adults with crowd- requirements engineering. In: Workshop on Requirements Engineering for Well-Being, Aging, and Health of the International Requirements Engineering Conference, pp. 57–64. IEEE (2022). https://doi.org/10.1109/REW56159.2022.00018
12. Tizard, J., Rietz, T., Blincoe, K.: Voice of the users: a demographic study of software feedback behaviour. In: IEEE International Conference on Requirements Engineering, pp. 55–65 (2020). https://doi.org/10.1109/RE48521.2020.00018
13. Wang, C., Daneva, M., van Sinderen, M., Liang, P.: A systematic mapping study on crowdsourced requirements engineering using user feedback. J. Softw. Evol. Process. (2019). https://doi.org/10.1002/smr.2199
14. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43839-8
15. Wouters, J., Menkveld, A., Brinkkemper, S., Dalpiaz, F.: Crowd-based requirements elicitation via pull feedback: method and case studies. Requirements Eng. **27**, 429–455 (2022). https://doi.org/10.1007/s00766-022-00384-6
16. Wüest, D., Fotrousi, F., Fricker, S.: Combining monitoring and autonomous feedback requests to elicit actionable knowledge of system use. In: Knauss, E., Goedicke, M. (eds.) REFSQ 2019. LNCS, vol. 11412, pp. 209–225. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-15538-4_16