Computer and Information Science; Vol. 13, No. 3; 2020
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Enhancing Software Evolution Requirements Engineering Based on User Feedback

M.Redwan aljannan[1], Manal A. Ismail[2], & Akram Salah[3]

[1] National Egyptian E-learning University (EELU), Giza, Egypt

[2] Computer Engineering and System Department, Engineering, Helwan University, Helwan, Egypt

[3] Computer and Information Sciences, Cairo University, Giza, Egypt

Correspondence: Redwan Aljannan, Software Engineering Department, Egyptian E-learning University, Cairo, Egypt. E-mail: m.redwan.aljannan@gmail.com

Received: April 26, 2020 Accepted: May 27, 2020 Online Published: June 8, 2020

doi:10.5539/cis.v13n3p16 URL: https://doi.org/10.5539/cis.v13n3p16

Abstract

End-user feedback has an essential role in the requirement's identification, prioritization, and management of the software evolution process. Several approaches are proposed for utilizing user-pushed feedback collected from social media, forums, and review systems. The collected feedback via the online channels contains a variety of information. Thus, the researchers proposed analytical approaches to classify feedback according to the data it holds. Still, recent results indicate that no single classifier works best for all feedback types and information sources. Also, online feedback does not have a direct mapping to the requirements, and it does not contain user context data. This causes wasting in developers' effort in understanding and analyzing feedback. On the other hand, online feedback cannot be used to explore user satisfaction and acceptance of the implemented and planned requirements. Likewise, the developer cannot collect feedback from a specific segment of the users. To overcome the deficiency of online feedback, this paper proposes a novel approach that utilizes pulling feedback from users while using the software. The proposed approach consists of a model and process for structuring feedback requests, linking feedback to the requirements, embedding feedback with the user context information, specifying the target audience for the feedback request, analyzing collected feedback depending on predefined interpretation rules, which provide insights that support developers in release planning. The feedback request model and process are implemented by a tool called FeatureEcho which was evaluated in a software company by conducting a case study for upgrading a governmental internet portal. The results indicate that FeatureEcho is a valuable step towards increasing the understanding of the end-users needs which supports the decision-making procedure of software evolution.

Keywords: software evolution, end-user feedback, release planning

1. Introduction

The developers must upgrade the software to maintain their user satisfaction and keep the software alive. End-users must be involved in software evolution by taking in their feedback. If the software is not upgraded based on the feedback, the end-user satisfaction of the software will be minimized because their opinion does not influence the software (Maalej, Walid, 2011). In the long run, developers miss a valuable opportunity to validate their software frequently and to discover new requirements (Leite, Software, 1991). The feedback collected from the user increases the understanding of the user's needs (Sauvola, Lwakatare, 2015)(Carreño et al., 2013). The developers use the feedback to improve further releases of the product either by developing new features or fixing existing errors. There are two primary types of end-user feedback (Stade, Fotrousi 2017): 1) User-pushed feedback through social media, online forums, and review systems. 2) Developer-pulled feedback through crowdsourcing platforms, or in-situ from the software in use.

Analyzing feedback collected by the user-push model is slow and complex (Sauvola et al., 2015). This feedback contains a mixture of data. It might contain bugs, feature requests, or suggestions. Consequently, researchers have studied how to use text classification and natural-language processing to categorize feedback (Maalej, Walid, 2015)(Iacob et al., 2013). Investigating analysis results show that no single classifier works best for all feedback types and feedback channels (Maalej, Walid, 2015).

Despite the classification problem, the user-push feedback must be analyzed to relate them to the functional and non-functional requirements (Groen, Eduard C., 2017) (Pagano, Dennis, 2013). Likewise, user-pushed feedback is missing user context information which can carry valuable information that can help make the feedback from the developers (Johanssen, Jan Ole, 2019). There are four main categories of user context information: Task, Spatio-Temporal, Run-time environment, and user information (Dzvonyar, Dora, 2016).

Additionally; there is a significant difference in software usage behavior among users (Lim, Soo Ling, 2014)(Dzvonyar, Dora, 2016)(Maalej W, Happel HJ, 2009). Thus, the developers should start communication with the user or a specific segment of users (Snijders, Remco, 2015) to measure their satisfaction with the implemented requirements. The developers need to assess how many users will benefit from adding or improving a specific requirement (Pagano, Dennis, 2013). Also, the developers want to provide a solution, or clarification of the feedback provided by the user, which is called the feedback loop (Bailey, Kendall, Meiyappan Nagappan, 2019).

To support online feedback processing and overcome their shortness, this paper proposes the utilizing of developer-pulled feedback from the software in use, to merge feedback with the user context information and link them to the functional and non-functional requirements. This paper provides 1) a feedback request model for the feedback enriched with the user context information and linked to the requirements. 2) a process for preparing feedback requests, specifying target audiences for the feedback, analyzing pulled feedback based on predefined rules which support release planning. The main idea is enabling the developers to have direct communication with the users to get their opinion regarding the evolution tasks; that will increase user satisfaction and turn into proactive software evolution.

To evaluate the suggested model, FeatureEcho tool has been developed and used by a development team to upgrade an internet web portal. FeatureEcho provides developers with indicators that help them make the right decisions during the release plan.

This paper is structured as follows: Section 2 investigates related work on consuming user feedback in software evolution. Section 3 provides an overview of the feedback request model and process. Section 4 reports FeatureEcho tool and the conducted case study for preliminary evaluation, and Section 5 concludes with a future direction.

2. Related Work

In conducting this paper, we investigated the research related to utilizing feedback in software evolution, which supports developers in the release plan. The most related papers are classified into four categories. First, the proposed feedback models for structuring feedback. Second, the proposed processes for integrating feedback into software development activities. Third, the method of directing feedback to a specific segment of end-users. Fourth, the feedback analysis techniques.

2.1 Feedback Models

According to the feedback ontology (Morales-Ramirez et al., 2015a), structured feedback is a type of feedback provided in a form that makes it easier to aggregate, process, analyze, and evaluate. This paper advocates that structured feedback in the form of a short questionnaire (Kim, Jun H., 2008) can support requirements engineering during software evolution.

(Sherief, Nada, 2015) proposes a classification of the user-pushed feedback types. This paper proposes the adoption of these types to be utilized in developer-user pull feedback.

Several researchers (Stade, Melanie, 2017) (Sherief, Nada, 2015) (Maalej, Walid, 2015) found that it is better if the feedback has a direct link with the requirement. This paper proposes an extension of the requirements goal model (Yu, 2009) for linking functional and non-functional requirements with the feedback.

Regarding non-functional requirements, this paper considers reliability, usability, portability, and performance suggested by (Lu & Liang, 2017).

2.2 Integrating Feedback into the Software Development Process

The developers are needed to integrate feedback with their work to benefit from them in the software evolution. (Dzvonyar, Dora, 2016) proposes CAFÉ, which provides an approach for collecting feedback from the software in use and saves them in the issue tracking system. CAFÉ provides a good solution for collecting user context information by depending on software Add-in. However, the feedback does not have a direct link to the requirements automatically. Also, CAFÉ utilizes the push model of feedback. So, the developers can't ask users questions related to their satisfaction with the implemented or planned features. This paper adopts and extends

CAFÉ for pulling feedback and linking them to the requirements. That will enhance the traceability of the feedback, save developers time and effort, and support them in software evolution

There are other approaches like OpenProposal (Rashid, Asarnusch, 2009), iRequire (Norbert et al. Seyff, 2010), ConTexter (Gärtner, Stefan, 2012) and AppEcho (N Seyff, Ollmann 2014) which provide a mechanism for the user to push their feedback. Also, several platforms proposed like iThink (Fernandesa, Joao, 2012), GREM (Lombriser, Philipp, 2016) and Refine (Snijders, Remco, 2015) conduct requirement elicitation from the crowd through online games, where crowd members participate, collaborate and discuss the requirements using an interactive system that contain all game elements (e.g., Avatars, points, and badges). Requirements engineering by these approaches depends on the linguistic processing of the feedback (och Dag, J. Natt, 2005). The elicited feedback is not linked to the software implemented or planned requirements.

2.3 Targeting a Specific Segment of the Users

In some cases, the developers are needed to direct feedback questions for a specific segment of the user, who have some specifications related to the user context information. The feedback question may also be triggered regularly (Froehlich, Jon, 2007) or at a particular moment when users do a specific task during the interaction with the system (e.g., opening the latest news in an internal portal). MyExperience (Froehlich, Jon, 2007) is a system that enables a semi-automated elicitation of user context information on mobile devices. And it provides a mechanism to pull user feedback by triggering a questionnaire when the user does a specific action while using the mobile phone. MyExperince proposes an XML interface for defining questionnaire triggering rules. MyExperience only comprising certain events of the smartphone, not events of the software under evolution. This paper proposes the adoption of MyExperince method to be appropriate for software evolution.

2.4 Feedback Analysis

The feedback is analyzed by a quantitative or a qualitative measure (Olsson & Bosch, 2015)(Morales-Ramirez, Perini, & Ceccato, 2015b)(Fiedler, Markus, Sebastian Möller, 2012). The following are the utilized measures: Mean Opinion Score (MOS), Likert scale, Semantic Differential Scale, GAP Analysis, Net Promoter Score, and sentiment analysis.

2.5 Summarization of the Related Works

The previous researches do not provide practice on how to turn into proactive software, the developers do not have an efficient approach to pull the feedback related to the software evolution activities. The main problems are 1) the collected feedback is not linked to the requirements, 2) the developers are not able to collect feedback from a specific segment of the user or have a feedback loop with them, 3) the collected feedback does not contain the user contextual information, 4) there is a need for an approach to analyzing feedback after linking them to the requirements to prioritize tasks in the software evolution road map, and 4) the proposed approaches for collecting pushed feedback assume that the users have the motivation to provide feedback.

This paper proposes the use of developer-pulled feedback from the software in use to collect feedback enriched with the user context information. The feedback structure of (Sherief, Nada, 2015), and the targeting method of MyExperience (Froehlich, Jon, 2007) are adopted to provide a structure of the feedback request. The feedback request allows developers to elicit requirement-based feedback from the end-users or a specific segment of the users. The process of (Sherief, Nada, 2015) (Dzvonyar, Dora, 2016) has been modified to be appropriate for preparing feedback elicited by the developers, collecting feedback from the users, and analyzing collected feedback.

3. Proposed Feedback Model

This paper proposes an approach that aims to increase the utilization of user feedback by linking them to the requirements. That will enhance requirements engineering during software evolution. The approach achieves the following criteria:

Criteria 1. Involving users to specify their satisfaction with the implemented requirements.

Criteria 2. Involving users to specify their acceptance of a feature evolution.

Criteria 3. Involving users by asking them about their problems and suggestions regarding the planned and implemented requirements.

Criteria 4. Allowing developers to involve a specific segment of users in software evolution.

Criteria 5. Enriching feedback with user context data, to provide a better understanding of the user needs.

Criteria 6. Providing a method for analyzing feedback with links to the requirements.

The pull model is considered for achieving the criteria 1,2,3. The feedback channel will be a software add-in, which achieves the criteria 4. The user contextual information is used as targeting rules to specify the user segment that will be involved in the feedback. This achieves criteria 5. The elicited feedback has a direct link to the requirements. After pulling feedback, an analysis will be processed based on predefined analysis rules. This achieves criteria 6. Figure 1 shows the proposed feedback approach.

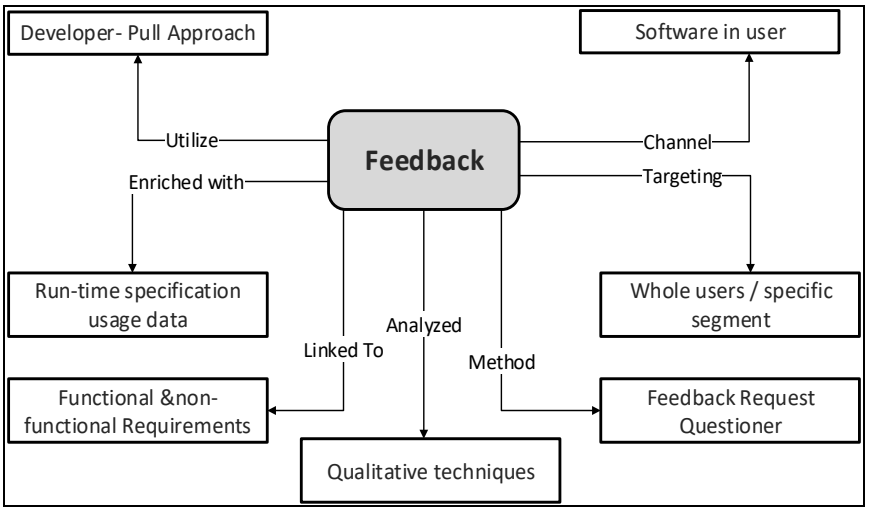

Figure 1. Feedback proposed approach

This paper proposes a structure of the feedback request that allows developers to pull requirements-based feedback.

3.1 Feedback Model Overview

This paper proposes a model for the feedback request derived from a combination of the goal requirement model, findings of (Sherief, Nada, 2015), and the targeting method of MyExperience (Froehlich, Jon, 2007). The model comprises three main components: feedback request, feedback response, and feedback analysis. The feedback request component is a questionnaire prepared by the developers and is directed to users to elicit their feedback. The feedback response is the collected feedback enriched with user context information. The analysis feedback component specifies the interpretation rules of the elicited feedback. The following sections describe the method of linking requirements to the feedback request and the specification of model components.

3.2 Linking Feedback to the Requirements

The feature is defined as a "triplet: F = (R, W, S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification" (Classen, Heymans, & Schobbens, 2008). To link requirements to the feedback, the requirement goal model has been adopted by adding the task to the default model. Each requirement contains tasks. The task is user-system interaction to do a specific activity. The feedback request will be linked to the task. Figure 3 shows the software backlog structure after adopting the goal model. For example; For e-shop web applications. There are features like a shopping cart, product catalog. The product catalog feature requires sorting products. The sorting requirement has tasks like sorting ascending or sorting descending.

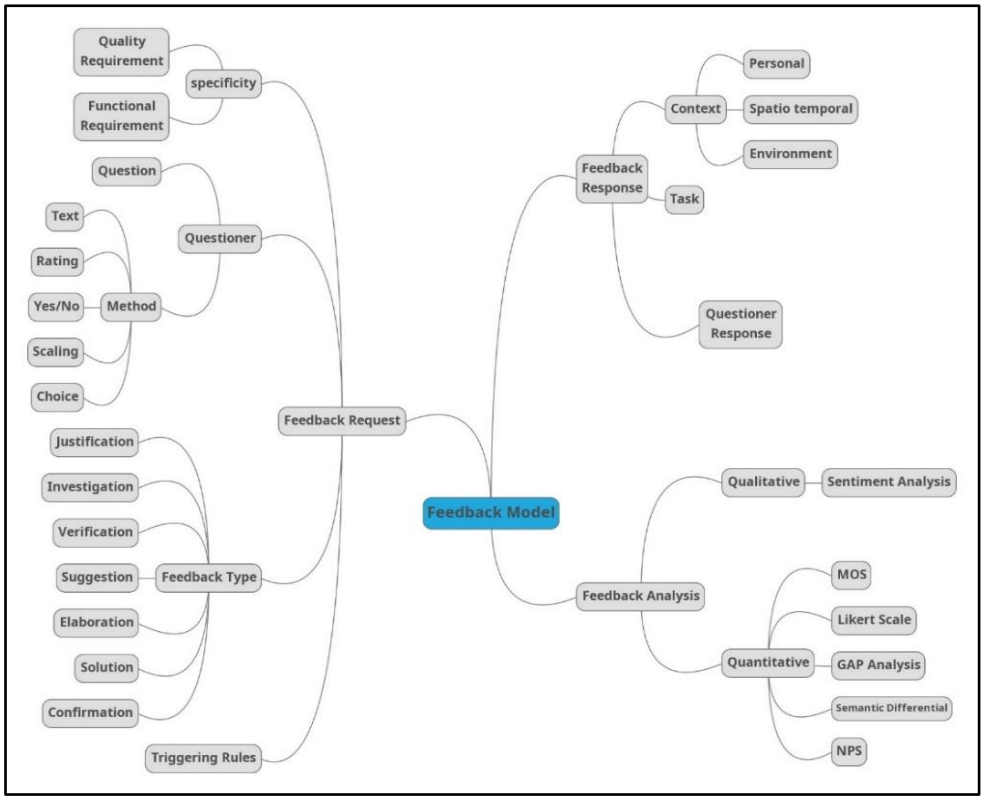

Figure 2. Feedback Model

Figure 3 shows the adopted goal model. It links the feedback request to the task to specify when and where to trigger the feedback request during the user-system interaction. Also, this structure is appropriate for asking questions about the non-functional features after the user interacts with a specific feature. For example; for an e-shop web application. Ask users about their satisfaction with the application usability when opening the home page several times. This paper considers the following non-functional requirements: reliability, usability, portability, and performance.

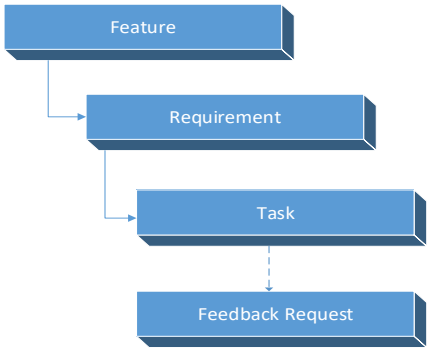

Figure 3. Adopted goal model

3.3 Feedback Request Component

Feedback request refers to the feedback questionnaire which has a link to a specific task. The feedback request comprises the following sub-components: Specificity, Questionnaire, Feedback Type, and Triggering rules.

3.3.1 Specificity

Specificity refers to the functional or non-functional requirement that the feedback request is linked to. The feedback request is related to the requirement via Task. Thus, the task specifies when to trigger the feedback

request. The developer will make a remote call for the feedback request in the task code. For instance; the feedback request will be called when the user clicks along the sort ascending button.

3.3.2 Feedback Type

The feedback types are assumed to be appropriate for pulling feedback. There are two parties: developers who ask questions, and users who provide feedback. The feedback types are divided into two categories: 1) Normal Feedback: when developers want to ask a normal question related to specific requirements to users, and 2) Feedback to feedback (loop): when developers want to get in contact with users based on their posted feedback to have more clarification or provide a solution to a problem. These two types of feedback allow developers to start communication with the users to ask them about the feedback or to provide clarification, solutions, or information about the new features. The user will be notified to ensure that their feedback affects the software. Table 1 shows the description of feedback types.

Table 1. Feedback Types

Type	Category	Description
Justification	Normal	Developers ask users about their opinion about an evolution task related to a specific requirement
Problems Investigation	Normal	Developers are asking about problems facing users when interacting with a specific requirement.
Confirmation	Normal	Developers ask users to specify their satisfaction with the implemented requirements or their acceptance of adding new requirements.
Suggestion	Normal	Developers are asking users for their improvement suggestion of a specific requirement
Elaboration	Feedback Loop	Developers ask a user to explain the posted feedback.
Solution	Feedback Loop	Developers provide a solution for a posted problem
Verification	Feedback Loop	Developers confirm that the problem sent by the user is solved.

3.3.3 Questionnaire

The feedback request is a question directed to users to pull their feedback; this question will be prepared by the developer as a questionnaire. This paper concerns the short questionnaire. An instance of the question can be "Can you scale your satisfaction of sorting function?"

The response to the question will be in one of the following styles: Text, Rating, Yes/No, Scaling or Choice

3.3.4 Triggering Rules

There are significant differences in software using behaviors across users. So, there is a need for the developers to pull feedback from a specific segment of the users. In some cases, developers want to pull feedback from the users who use specific features many times, because they know their problems and have a clear vision about how to upgrade.

This paper proposes the user segmentation based on the user context information. There are three related items to define triggering rules:

- **Task**: the system-user interaction with a requirement. Specifies when to trigger the questionnaire.
- **Rules**: a set of conditions related to the user context information. Specify for Whom to trigger the questionnaire.

This paper proposes an interface for defining triggering rules extended from MyExperience (Froehlich, Jon, 2007). Figure 4 shows an example of the triggering rule. First, the questionnaire and the related task are specified, then the triggering rules are defined and the relation between them is specified to be "And", "Or".

```
<Questioner name="MeasureUserStatisfaction_ProductCatlog" Task="Click_Sort_Button">
<Rules condition= "And">
    <Rule type="CountryCheck" value="Egypt">
    <Rule type="OSCheck" value="Andriod 7.1">
    <Rule type="PerfomeBeforeCheck" value="3">
</Rules>
</Questioner>
```

Figure 4. Triggering Rule Example

The user contextual information is Country, OS, Browser, Language, and Task. The feedback request is prepared by the developers and contains the pulling request question, triggering rules, and the request is linked to a specific task.

3.4 Feedback Response Component

The feedback is collected from the software in use through an add-in. The add-in calls the questionnaire and posts the feedback. When a user interacts with the software and does one of the predefined tasks which are associated with a questionnaire, the triggering rules of the questionnaire are checked, and the questionnaire is triggered to the user. The feedback response component refers to the user feedback on the triggered questionnaire. The response holds the user's contextual information and the feedback of the user.

3.5 Feedback Analysis Component

Developers are specifying the analysis technique of elicited feedback. This paper considers the following quantitative analysis techniques: Mean Opinion Score (MOS), Likert Scale, Semantic Differential Scale, GAP Analysis, and Net Promoter Score. Also, this paper considers sentiment analysis as a qualitative analysis technique.

The Feedback request model uses these techniques in the form of interpretation rules that can be defined by the development team when preparing a feedback request to facilitate decision making (e.g., a development team can use GAP analysis technique to measure the gap between the expected and real satisfaction of a specific requirement). The analysis of the feedback will help developers in taking the right decisions related to software evolution activities. The analysis results may cancel a planned evolution or may change or confirm them.

4. Proposed Feedback Process

A process for collecting and analyzing feedback is proposed by adopting the process of (Dzvonyar, Dora, 2016) to be appropriate for pulling feedback. Figure 5 shows the process which consists of three phases. The proposed process allows developers to collect two types of data: 1) contextual information only: when developers want to collect user contextual information about the user without triggering feedback requests. 2) User feedback embedded with the context information: when developers want to collect user context information and trigger a feedback request to the user. The following sections describe the process tasks.

4.1 Preparation Phase

In this phase, the developers are preparing for the feedback, they start by deciding what data they want to collect for every software requirement, user context information only, or user contextual information and user feedback. The following are the steps of the feedback preparation by the developers:

A. Defining software requirements: As a result, the software backlog is defined
B. Specify the requirement that the developers want to collect their user contextual information or feedback and define tasks for.
C. Change the code of the software under development to make remote calls for the tasks, in the places where the developers want to trigger the feedback request
D. If developers want to collect feedback, they will define the feedback request and associate them to the defined task by
    a. Specifying the feedback type
    b. Specifying feedback question and style
    c. Specifying feedback triggering rules

d. Specifying the interpretation rules

E. Publishing a new release of the software

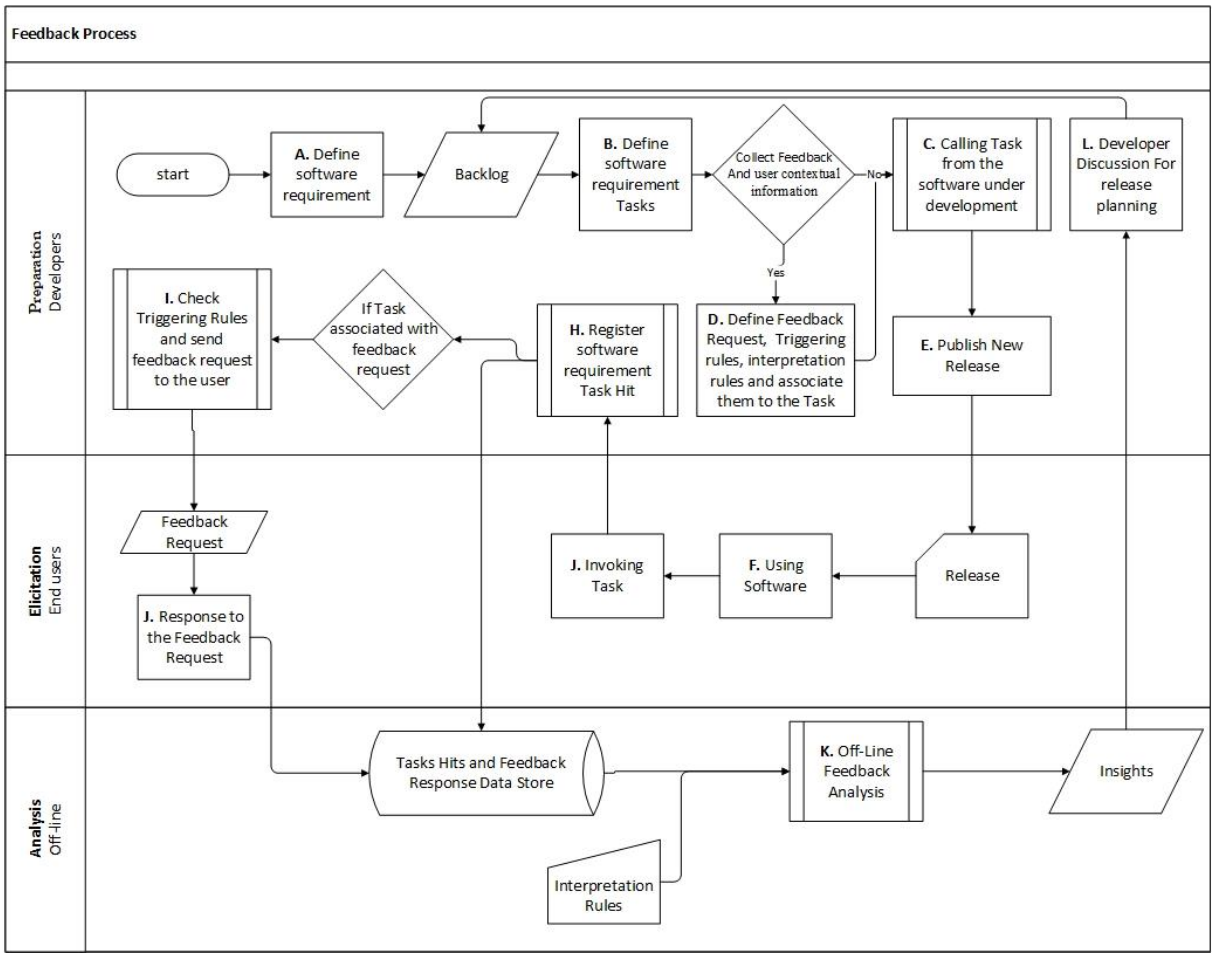

Figure 5. Feedback Preparation, Elicitation and Analysis Process

To save developers time a feedback request templates are defined which cover the most common questions the developers want to ask during evolution tasks. Figure 6 shows a sample of the pre-defined templates.

Requirement Evaluation Template	**Type**: Verification
	Description: Let your crowds widely evaluate their experience with your software functional and non-functional requirements.
	Hints: Appropriate for new added requirement
	Question: Overall, how satisfied are you with the "Software Requirement"?
	Analysis: MOS
	Targeting Rules: Ask long time software users

Figure 6. Feedback Request Template Example

4.2 Feedback Elicitation Phase

A new version of the software is released for the users. The following are the steps of user interaction with the software and how to collect the feedback:

F. The user interacts with the software.

G. If the user interacts with a pre-defined task, a remote call for the task is invoked

H. A Task hit is registered. The task hit contains the following data:

   a. Software requirement

   b. User context information

I. If the task is associated with a feedback request, the triggering rules of the request are checked. If the rules are achieved, the feedback request will be sent to the user.

J. The user responds to the feedback request and the response will be registered with links to the task and the software requirement.

4.3 Feedback Analysis Phase

When passing the duration of feedback collection, the analysis of the feedback starts as:

K. An off-line analysis is performed for the user contextual information and the collected feedback which is linked to the software requirements. The analysis is run based on the pre-defined interpretation rules which were defined when defining the feedback request by the developers. The analysis provides a set of indicators (e.g., users are not satisfied with the specific feature) to support the developers in release planning.

L. The developers will discuss the analysis result, then they make the required changes in the software backlog, they may remove some features, add some new ones, or plan to adopt another feature.

5. FeatureEcho

FeatureEcho tool was developed to evaluate the proposed Feedback request model and process. FeatureEcho provides a plugin to elicit feedback in-situ. Also, FeatureEcho provides developers with a back-end control panel to define the feedback request and preview analysis results. FeatureEcho needs a few modifications in the software under development code. FeatureEcho implements service-oriented architecture SOA; the back-end component and the feedback gathering plugin are communicating with each other through RESTful web service using JSON data format. Figure 7 shows FeatureEcho Architecture.

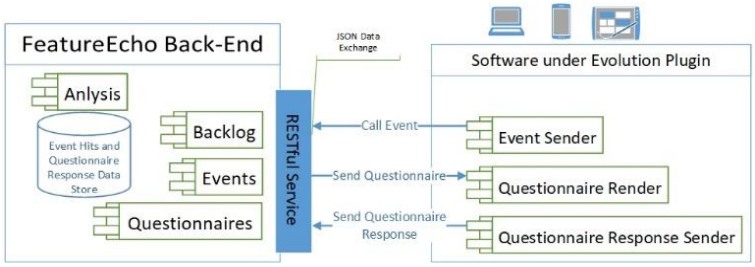

Figure 7. FeatureEcho Architecture

The feedback should be pulled from the user one time after achieving the triggering rules. So, an algorithm for user identification is implemented. The main idea is combining user context information with the user mac address, then hashing the result to provide the user with a unique identifier. Consequently, the feedback will be pulled once from them.

5.1 Evaluation Case Study

To investigate the applicability of the proposed model in practical circumstances, FeatureEcho was used in a software company to upgrade a governmental internet web portal. The portal was specially designed to bring information from different sources, restructuring them and presenting them uniformly. Besides enabling visitors from different domains (individuals, businesses, and government sectors) to browse the portal's content, the portal provides features for many users.

The web portal comprises 30 features. The development team planned to implement the portal in three releases. Table 3 shows the initial release plan. After finishing release 1, the development team used FeatureEcho to pull end-user feedback regarding three implemented features and one planned feature. Also, they collected user context information of three features without pulling end-user feedback. After finishing feedback elicitation, the feedback was analyzed and the results provided the developers with potential information that led to changes in the release plan.

Table 2. Portal Initial Release Plan

Release	Number of Features
Release 1	17
Release 2	**7**
Release 3	6

5.2 Case Study Preparation

After finishing Release 1, the development team used FeatureEcho to prepare feedback requests for 3 implemented features as described in Table 4. Also, they prepared a feedback request for a planned feature, as described in Table 5, to measure the user's acceptance of implementing this feature.

Table 3. Feedback Requests for the Implemented Requirements

Feature	Requirements	Feedback
1. News Center	• Req1.1: Provide news in different types like national news, international, and legal news. • Req1.2: Provide end-user subscription for receiving the latest news. • Req1.3: Provide news printing and sharing on social media. • Req1.4: Provide a panel for viewing the hottest news.	• Scope: Measuring end-users' satisfaction with the implemented requirements • Targeting Rules: After visiting the news center, two times. • Analysis Approach: Mean Opinion Score (MOS)
2. Media Center	• Req2.1: Provide media library for different media types; photo, video, animation, and document. • Req2.2: End-user can rate media content. • Req2.3: End-user can download media to his device.	• Scope: Measuring end-users' satisfaction with the implemented requirements • Targeting Rules: After using the media player. • Analysis Approach: Mean Opinion Score (MOS)
3. Portal Usability	• Req3.1: Non-Functional Requirements	• Scope: Investigate the gap between expected and real end-users' satisfaction with portal usability • Targeting Rules: After visiting the portal three times, and browsing them on Mobile • Analysis Approach: GAP Analysis

Table 4. Feedback Request for the planned requirement

Feature	Requirements	Feedback
4. Live Video Conference	Req4.1: The government agency has over 15 branches of customer service, they want to apply a feature on the portal for enabling live video conferences between the citizen and customer service representative	Scope: Measuring end-users' acceptance of implementing video conference feature. Targeting Rules: After visiting the portal four times. Analysis Approach: Likert scale

Besides, developers wanted to collect user context information only for three features: 1) Frequently Asked Questions, 2) Content Search, and 3) Organization Events catalog.

5.3 Case Study Result

After one week of portal release, the user context information was collected for 743 times. Figure 8 illustrates the features usage frequency. The feedback requests triggered 157 times as shown in Figure 9.

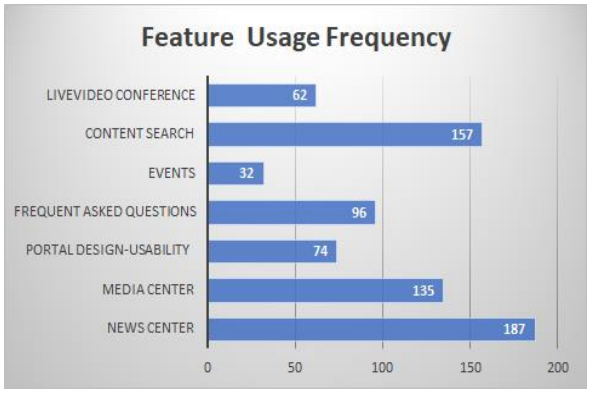

Figure 8. Feature Usage Frequency

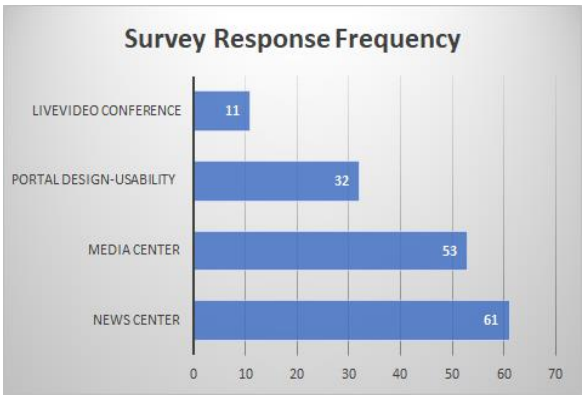

Figure 9. Survey Response Frequency

The collected user context information on all requirements shows important information for the development team as stated in Table 6. They discovered that a large number of portal visitors are using mobile.

Table 5. Information Extracted from implicit feedback data

Implicit Feedback Data	Insights
Browsers	Mobile: 40%, Chrome: 28%, Firefox: 18%, IE: 11%, Other: 3%
Language	Arabic: 65%, English: 28%, Other: 7%

FeatureEcho provides several insights into the collected feedback. The team was expected that the GUI will be satisfied by the end-users, but the feedback request of feature#3 has shown that the end-users are not satisfied as illustrated in Figure 10. The feedback request of feature#4 results that the end-users are not interested in adding a live video conference feature in release 2 as shown in Figure 11.

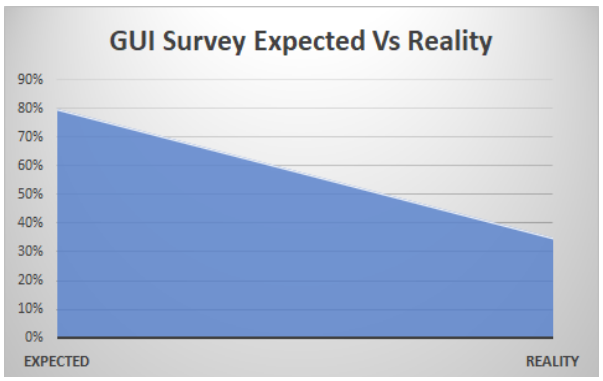

Figure 10. GUI Survey Expected Vs Reality

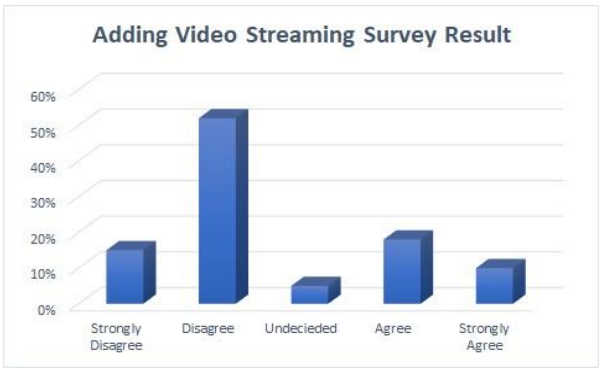

Figure 11. Adding video streaming feature Survey Result

5.4 Case Study Discussion

By discussing the insights provided by FeatureEcho, the development team modified the release plan. Depending on the questionnaire's response, the GUI should be enhanced in Release 2, and the live video conference feature will be removed from the evolution road map. On the other side, and depending on the user contextual information, the user experience for the mobile and chrome browser should be considered, and the English content needs to be reviewed in Release 2. As a result, three tasks added to the release 2 and the video conference feature have been removed.

6. Discussion

In this section, we discuss the implications of our suggested model, process, and tool as well as its current limitations.

6.1 Implications

The case study results confirm that the proposed model and process can support developers in software evolution and provide them with in-depth data for a better understanding of users' needs. The suggested model and process allows developers to prepare feedback requests with links to the requirements. Consequently, the developers avoid wasting time and effort in mapping feedback responses to the requirements. That will increase the accuracy of the collected feedback and the feedback can be quantified, which allows for measuring user satisfaction and acceptance of the requirements. While the model proposed by (Sherief, Nada, 2015) is appropriate to extract the feedback from the natural text and link them to the feature. However, (Sherief, Nada, 2015) approach provides more details, but it decreases the accuracy of the elicited feedback. Likewise, (Sherief, Nada, 2015) model is missing the user context information which helps in understanding software usage behavior. Also, the suggested model allows the targeting of a specific segment of users to collect their feedback.

Several proposed tools use user feedback as the main source of software product evolution. Figure 12 illustrates a comparison between FeatureEcho and the other previous tools. By investigating comparison, FeatureEcho enhances the targeting technique provided by MyExperince to use the user context information as the segmentation attributes. Also, FeatureEcho enhances the Contexter technique to collect the context data by increasing the types of the collected data and embedding them in the user feedback. FeatureEcho provides a direct link between the feedback and the requirements, while Refine, GREM, and iThink are collecting feedback on the software level and need manual work to map the pushed feedback to the software features. Like AppEcho and OpenProposal the proposed tool FeatureEcho uses add-in to collect the feedback from the software in use. Besides, FeatureEcho provides predefined templates to facilitate feedback request preparation. Additionally, FeatureEcho provides a novel technique based on predefined interpretation rules to provide a primary analysis of the collected feedback, that helps developers in release planning.

Feature / Tool	Feedback Elicitation		Feedback Template		Software evolution		Feedback Linked to Requirements		Collecting Context information		Feedback Loop		Analysis	
	Push	**Pull**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Manual**	**Automated**
AppEcho	X			X	X			X		X		X	X	
OpenProposal	X			X	X			X		X		X	X	
Contexter	X			X	X			X	X			X	X	
MyExperince		X		X		X		X		X		X	X	
iThink		X	X			X	X			X		X		X
GREM		X	X			X	X			X		X		X
Refine		X	X			X	X			X		X		X
FeatureEcho		X	X		X		X		X		X			X

Figure 12. Comparison Matrix

6.2 Limitations

There are some limitations of the proposed model and process, the users are not able to submit the feedback at any time, they are restricted to respond to the feedback requests after doing a specific activity during using software, this will lead to a lack of a comprehensive view of users' opinions of the system. Likewise, asking for feedback during the use of software features may confuse users and affect user experience.

On the other hand, the case study conducted to evaluate FeatureEcho is limited to a web portal and one release evolution. To get more accurate results, FeatureEcho has to be evaluated in a long-term setting, preferably by following the amount of feedback over several consecutive releases. Also, FeatureEcho should be evaluated with different types of software like mobile and desktop.

7. Conclusion

The process of software evolution can be improved by consuming end-user feedback. The proposed feedback request model and process help in capturing feedback, conducting feedback requests, and linking feedback to the

software requirements. A case study showed the potential of FeatureEcho in the support of software evolution.

It is necessary to find a good trade-off between features corresponding to perceived user needs and new inventive ones that may provide a competitive advantage. Several factors interact with each other during the decision-making process of software evolution; some of these factors are related to marketing and competition, and other factors are related to the development cost. We tried to isolate the factor of user satisfaction by implementing features and acceptance for adding new requirements as an essential factor in the evolution, but that does not mean that other factors must be ignored. The proposed feedback request model can be developed to provide a mechanism for improving communication between marketers and developers to improve the quality of release planning, which increases the quality of the product.

Today the end-users can use different channels to provide feedback like social media, forums, and store review, that will lead to a steady stream of feedback. The proposed feedback request model can be developed in the future to integrate these data with the pulled feedback to have a wide view of users' needs and satisfaction on the software's requirement level.

References

Bailey, K., Nagappan, M., & Dig, D. (2019). Examining User-Developer Feedback Loops in the iOS App Store. *Scholarspace.Manoa.Hawaii.Edu*. Retrieved from https://scholarspace.manoa.hawaii.edu/handle/10125/60178

Carreño. et al. (2013). Analysis of user comments: an approach for software requirements evolution. *International Conference on Software Engineering (ICSE),* 582-591. *IEEE*. Retrieved from https://dl.acm.org/citation.cfm?id=2486865

Classen, A., Heymans, P., & Schobbens, P. Y. (2008). What's in a feature: A requirements engineering perspective. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4961 LNCS*, 16-30. https://doi.org/10.1007/978-3-540-78743-3_2

Dzvonyar, D. et al. (2016). Context-Aware User Feedback in Continuous Software Evolution. *IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED). IEEE, 2016.*

Fernandesa, J. et al. (2012). iThink: A game-based approach towards improving collaboration and participation in requirement elicitation. *ELSEVIER Procedia Computer Science, 15*, 66-77. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050912008216

Fiedler, M., Möller, S., & Reichl, P. (2012). Quality of experience: From user perception to instrumental metrics (Dagstuhl Seminar 12181). *Dagstuhl Reports, 2*(5), *Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 201*. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2012/3598/

Froehlich, J. et al. (2007). MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services,* pp. 57-70. Retrieved from https://dl.acm.org/citation.cfm?id=1247670

Gärtner, S., & Schneider, K. (2012). A method for prioritizing end-user feedback for requirements engineering. *5th International Workshop on Co-Operative and Human Aspects of Software Engineering (CHASE),* pp. 47-49. *IEEE*. Retrieved from https://ieeexplore.ieee.org/abstract/document/6223020/

Groen, E. C. et al. (2017). The crowd in requirements engineering: The landscape and challenges. *IEEE Software, 34*(2), 44-52. Retrieved from https://ieeexplore.ieee.org/abstract/document/7888433/

Iacob, et al. (2013). Retrieving and analyzing mobile apps feature requests from online reviews. *10th Working Conference on Mining Software Repositories (MSR) (Pp. 41-44). IEEE.* Retrieved from https://dl.acm.org/citation.cfm?id=2487094

Johanssen, J. O. et al. (2019). Feature Crumbs: Adapting Usage Monitoring to Continuous Software Engineering. *International Conference on Product-Focused Software Process Improvement (Pp. 263-271). Springer, Cham.* https://doi.org/10.1007/978-3-030-03673-7_19

Kim, J. H. et al. (2008). Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. *443-452.* Retrieved from https://dl.acm.org/citation.cfm?id=1357126

Leite, J. C. S. D. P., & Freeman, P. A. (1991). Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering, 12,* 1253-1269. Retrieved from https://ieeexplore.ieee.org/abstract/document/106986/

Lim, S. L. et al. (2014). Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering, 41*(1), 40-64. Retrieved from https://ieeexplore.ieee.org/abstract/document/6913003/

Lombriser, P. et al. (2016). *Gamified Requirements Engineering: Model and Experimentation*. https://doi.org/10.1007/978-3-319-30282-9_12

Lu, M., & Liang, P. (2017). *Automatic Classification of Non-Functional Requirements from Augmented App User Reviews*. https://doi.org/10.1145/123_4

Walid, M., & Dennis, P. (2011). *On the Socialness of Software*. https://doi.org/10.1109/DASC.2011.146

Walid, M., & Hadeer, N. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. *IEEE 23rd International Requirements Engineering Conference (RE).* IEEE, 2015. Retrieved from https://ieeexplore.ieee.org/abstract/document/7320414/

Maalej, W. et al. (2015). Toward data-driven requirements engineering. *IEEE Software, 33*(1), 48-54. Retrieved from https://ieeexplore.ieee.org/abstract/document/7325177/

Maalej, W., & Happel, H. J. R. A. (2009). When users become collaborators: towards continuous and context-aware user input. *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications* (pp. 981-990*).* Retrieved from https://dl.acm.org/citation.cfm?id=1640068

Morales-Ramirez, et al. (2015a). An ontology of online user feedback in software engineering. *Applied Ontology, 10*(3-4), 297-330. Retrieved from https://content.iospress.com/articles/applied-ontology/ao150

Morales-Ramirez, I., Perini, A., & Ceccato, M. (2015b). Towards supporting the analysis of online discussions in OSS communities: A speech-act based approach. *Lecture Notes in Business Information Processing*, *204*, 215-232. https://doi.org/10.1007/978-3-319-19270-3_14

och Dag, J. N. et al. (2005). *A linguistic-engineering approach to large-scale requirements management.* Ieeexplore.Ieee.Org. Retrieved from https://ieeexplore.ieee.org/abstract/document/1377120/

Olsson, H. H., & Bosch, J. (2015). *Towards Continuous Customer Validation: A Conceptual Model for Combining Qualitative Customer Feedback with Quantitative Customer Observation*. https://doi.org/10.1007/978-3-319-19593-3_13

Pagano, D., & Bernd, B. (2013). User involvement in software evolution practice: a case study. *Nternational Conference on Software Engineering (ICSE)* (pp. 953-962). IEEE, 2013. Retrieved from https://dl.acm.org/citation.cfm?id=2486920

Rashid, A. et al. (2009). Bringing developers and users closer together: the open proposal story. *PRIMIUM-Process Innovation for Enterprise Software*. Retrieved from https://dl.gi.de/handle/20.500.12116/23212

Sauvola et al. (2015). Towards customer-centric software development: a multiple-case study. *41st Euromicro Conference on Software Engineering and Advanced Applications* (pp. 9-17). IEEE. Retrieved from https://ieeexplore.ieee.org/abstract/document/7302425/

Norbert, S., Gregor, O., & Manfred, B. (n.d.). *AppEcho: a user-driven, in situ feedback approach for mobile platforms and applications.* Dl.Acm.Org. Retrieved from https://dl.acm.org/citation.cfm?id=2593927

Seyff, N. et al. (2010). Using mobile re tools to give end-users their own voice. *IEEE International Requirements Engineering Conference* (pp. 37-46). IEEE. Retrieved from https://ieeexplore.ieee.org/abstract/document/5636923/

Sherief, N. et al. (2015). Modelling users feedback in crowd-based requirements engineering: An empirical study. *FIP Working Conference on The Practice of Enterprise Modeling*, *235*, 174-190. https://doi.org/10.1007/978-3-319-25897-3_12

Snijders, R. et al. (2015). REfine: A gamified platform for participatory requirements engineering. *IEEE 1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE)* (pp. 1-6). *IEEE*. Retrieved from https://ieeexplore.ieee.org/abstract/document/7367581/

Stade, M. et al. (2017). Feedback gathering from an industrial point of view. *IEEE 25th International Requirements Engineering Conference (RE)* (pp. 71-79). IEEE. Retrieved from https://ieeexplore.ieee.org/abstract/document/8048892/

Yu, E. S. (2009). Social modeling and i*. *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos (*pp. 99-121). https://doi.org/10.1007/978-3-642-02463-4_7