FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring

Marc Oriol[1], Melanie Stade[2,3], Farnaz Fotrousi[4], Sergi Nadal[1], Jovan Varga[1], Norbert Seyff[2,5], Alberto Abello[1], Xavier Franch[1], Jordi Marco[1], Oleg Schmidt[6]

[1] Universitat Politècnica de Catalunya, Spain
{moriol, snadal, jvarga, aabello, franch}@essi.upc.edu, jmarco@cs.upc.edu
[2] University of Applied Sciences and Arts Northwestern Switzerland, Switzerland
{melanie.stade, norbert.seyff}@fhnw.ch
[3] Berlin University of Technology, Germany
[4] Blekinge Institute of Technology, Sweden
farnaz.fotrousi@bth.se
[5] University of Zurich, Switzerland
[6] SEnerCon GmbH
oleg.schmidt@senercon.de

***Abstract*—Context: Software evolution ensures that software systems in use stay up to date and provide value for end-users. However, it is challenging for requirements engineers to continuously elicit needs for systems used by heterogeneous end-users who are out of organisational reach. Objective: We aim at supporting continuous requirements elicitation by combining user feedback and usage monitoring. Online feedback mechanisms enable end-users to remotely communicate problems, experiences, and opinions, while monitoring provides valuable information about runtime events. It is argued that bringing both information sources together can help requirements engineers to understand end-user needs better. Method/Tool: We present FAME, a framework for the combined and simultaneous collection of feedback and monitoring data in web and mobile contexts to support continuous requirements elicitation. In addition to a detailed discussion of our technical solution, we present the first evidence that FAME can be successfully introduced in real-world contexts. Therefore, we deployed FAME in a web application of a German small and medium-sized enterprise (SME) to collect user feedback and usage data. Results/Conclusion: Our results suggest that FAME not only can be successfully used in industrial environments but that bringing feedback and monitoring data together helps the SME to improve their understanding of end-user needs, ultimately supporting continuous requirements elicitation.**

***Index Terms*—Feedback gathering, usage monitoring, requirements, user involvement, feedback acquisition, data collection, requirements elicitation, software evolution, user feedback.**

I. Introduction

Software systems face the need to continuously evolve and adapt to meet changing stakeholder needs. This requires continuous elicitation of stakeholder needs regarding the software in use [1], a rapid understanding of requirements violation [2], and an immediate reaction to evolve requirements [3].

Traditional requirements elicitation methods such as interviews, workshops, or prototyping can be successfully applied to software systems where the stakeholders, including end-users, are within organisational reach but are more challenging in the context of software systems used by a heterogeneous crowd of end-users in different locations. The main challenges that requirements engineers face in this context is how to continuously elicit end-user needs for the software in use, and validate if the implemented requirements are aligned with their needs [4]. In this regard, continuous requirements elicitation demands an efficient approach to elicit end-user needs remotely and in a scalable manner.

Academic literature suggests, amongst others, two suitable approaches to support continuous requirements elicitation. One approach, user feedback gathering, allows a vast number of end-users to communicate their needs for a deployed software anytime and anywhere. This approach also involves end-users who are out the organisational reach [5]. The other approach is system monitoring. For example, monitoring the user behaviour, combined with data mining techniques, have been proposed to support requirements elicitation and decision making [6].

Both approaches are the basis for the work done in our research. We have already evidence that they are useful to elicit new requirements [7][8]. However, novel work argues that both approaches need to be combined to improve the requirements elicitation process [9]. Some of this work has also included conceptual solutions and early architectural prototypes. However, to the best of our knowledge, none of the existing solutions is mature enough to be used by companies to support their continuous requirements elicitation process. Furthermore, little is known about the actual benefits of combing user feedback and monitoring in real-world scenarios.

In this paper, we present a framework that provides advanced features for the combined collection and analysis of feedback and monitoring data. To show the validity of our technical solution, we applied FAME in a German software

company, which successfully integrated the framework into its requirements engineering process and are using it to support continuous requirements elicitation for one of their software products. Early evaluation results suggest that the data gathered with the help of our framework supports the software company in better understanding the needs of their end-users.

A. Motivating Example

In this section, we present the main challenges that a German SME (namely SEnerCon[1]) faced in the requirements elicitation process for one of their software products. To understand SEnerCon's current situation, we conducted a case study (for a detailed method description see [15]). The company develops and maintains web applications for end-users in the domain of energy efficiency management. This iESA web application enables end-users to visualise their energy consumption and guides them in saving energy (www.energiesparkonto.de).

To elicit requirements from their end-users, SEnerCon used to collect feedback from traditional feedback communication channels including email, contact forms, support hotline, and an online forum and did not use a dedicated software tool for feedback gathering. SEnerCon also had some monitoring software components installed, which were mainly used to assess the Quality of Service (QoS) and reliability of the web application, rather than helping in the requirements elicitation process. In particular, SEnerCon already obtained the QoS of its services through system logs and hardware statistics, but user feedback was the only source for SEnerCon to elicit requirements. In this situation, SEnerCon faced several challenges in their requirements elicitation process including:

Understanding the Feedback. Before SEnerCon extracted information suitable for refinement or elicitation of a requirement, the company had first to understand the feedback message. In particular, when the user feedback was incomplete, inaccurate, or unstructured, this was a cumbersome step. Sometimes, SEnerCon needed to start further investigations. This included asking the end-user questions for clarification or testing various potential scenarios where the end-user had struggled. Such additional investigations were time-consuming and, in some cases, had even annoyed the end-user when SEnerCon contacted them.

Consolidating and Prioritizing Requirements. Another challenge for SEnerCon was to estimate how many end-users might be affected by a feature request or problem, in particular, when only one end-user communicated this problem. The number of affected end-users could be a good indicator for SEnerCon to decide on the consolidation and prioritisation of a new or revised requirement.

Identifying a Proper Satisfaction Criteria. The final challenge SEnerCon were faced with when eliciting a new requirement, was to identify a satisfaction criteria correctly. Usually, the satisfaction criteria were not explicitly stated by the end-user, or it was not clear-cut in the feedback. This caused the SEnerCon team to define the satisfaction criteria based on their personal opinions.

[1] https://www.senercon.de/

We expect that the combination of feedback and monitoring data could help SEnerCon to solve or mitigate some of these (and further) challenges. For instance, usage data about what the end-users did before they provided feedback could be useful to clarify the end-users' intention to provide feedback or the problems they faced. Monitoring how many end-users used a specific functionality would provide an indicator of the importance of a requirement related to that functionality. Finally, monitoring data could also provide measurements that could help to define satisfaction criteria that were not explicitly stated in feedback (e.g., the time end-users spent waiting for a task to complete before they leave the web application or provide negative feedback).

B. Research Objective

In this paper, we address the following research objective: *To provide a unified framework capable of gathering and storing both feedback and monitoring data, as well as combining them using an ontology, to support the continuous requirements elicitation process.* To achieve such a research objective we have developed a framework, named FAME, and have conducted the first evaluation together with SEnerCon, demonstrating the successful combination of feedback and monitoring data, and how such combined data can provide valuable information for the elicitation of new requirements. The remainder of the paper is structured as follows. Section II presents the related work. Section III describes our proposed technical solution. Section IV presents our validation in a real-world context, including a description of the execution of the validation, the results, and its discussion. Finally, Section V presents the conclusions and future work.

II. Related Work

A. Feedback Gathering for Requirements Elicitation

Feedback acquisition approaches allow end-users to communicate problems, needs, and opinions while using a software product [8]. From such feedback data, requirements engineers can extract requirements [5].

Several dedicated feedback tools have already been developed, which can be designed as standalone (e.g.,[10]), embedded (e.g., [11]), or cross-platform (e.g., [12]) solutions. These and further tools aim to support and motivate end-users in communicating feedback in linguistic (e.g., free text input, category selection) or non-linguistic (e.g., rating, annotated screenshot) formats [13]. The communication of feedback can be initiated (pushed) by the end-users (e.g., by pressing a feedback button) or by the requirement engineer who asks (pulls) for feedback (e.g., by triggering a feedback dialogue that appears to end-users in a pop-up window) [14], but most of the research tools support only the first scenario.

Although user feedback can be a valuable data source for the requirements elicitation process, several approaches have emphasised that feedback can be hard to understand and interpret when context information related to particular feedback is missing, or when the description is unstructured [15][16][17][18]. Moreover, when every end-user can provide feedback anytime and anywhere, several challenges for re-

quirements elicitation arise, for example, processing of fake and very subjective feedback [19][20].

B. Monitoring for Requirements Elicitation

Monitoring is commonly used to assess if the requirements are satisfied or violated during the execution of the system, i.e. requirements monitoring [21]. Requirements monitoring comprises the analysis and evaluation of the stated requirements, tracing them to the metrics, observing and measuring the metrics of the system, and deriving the requirements status [21]. Until recently, monitoring approaches have been focusing mainly on requirements assessment.

However, with the emergence of data-driven decision-making methodologies, new monitoring approaches have been proposed to support the requirements elicitation process. For instance, some approaches propose to use behavioural data to better understand end-user needs in web-based applications. For this, various types of data can be measured, including the end-users' action flow, their eye movements, and the length of time they spend on different features [22]. In this direction, some approaches propose to specify the end-user goals accompanied with the expected end-user behaviour and then observe deviations on that behaviour to identify the need for a new requirement [7].

Other approaches propose to use not only behavioural data but also runtime data of the system (e.g., QoS) to identify when a high-level indicator (e.g., reliability) is being violated. Such violation would, ultimately, lead to a new requirement [6].

C. Combining Feedback and Monitoring Data for Requirements Elicitation

A number of researchers have proposed to use both feedback and monitoring data to elicit new requirements. For instance, to support requirements elicitation, some approaches [23][24] combine feedback data with monitoring data coming from the same end-user who provided the feedback (e.g., log data). However, using this approach, the authors were only able to capture data of the end-user who provided feedback and were not able to capture data from other end-users (e.g., to identify how many end-users experienced an issue reported in feedback), or other types of monitoring data (e.g., QoS). In contrast, another approach [25] used monitoring data from all end-users and applied process mining techniques to observe end-users behaviour and elicit new requirements. In this work, the authors suggested that such information could be combined with feedback to refine the requirements and help improve the requirements prioritisation process. However, the research direction was not explored in-depth, and most of it was left as future work.

In fact, few technical solutions that combine feedback and monitoring data exist for other purposes. For instance, QoE probe [26] is a lightweight mobile application that combines user feedback and monitoring data for requirements verification and validation. The probe periodically requests for user feedback, while collecting and aligning continuously with usage logs. The tool records usage data including user-id, timestamps of requirement events in the feature level (e.g., starting or completing a feature) and user interaction level (i.e., user input or application output). In completion of user interaction, a feature or a group of features, a feedback dialogue is triggered based on a defined likelihood, to collect the level of users' acceptance and user comments. The probe is only for gathering user feedback and monitoring data and does not include any analysis component.

MyExperience [27] is another tool that combines monitoring data and user feedback, but it is used to support studies on human behaviour or health (e.g., monitoring health-related metrics through sensors and asking end-users how they feel). However, to the best of our knowledge, no solution has advanced from a conceptual solution to a technical implemented framework that comprehensively combines feedback gathering and monitoring to support continuous requirements elicitation.

III. FAME FRAMEWORK

We have developed a framework named FAME (Feedback Acquisition and Monitoring Enabler). FAME[2] enables a requirements engineer to collect and analyse the combined feedback and monitoring data regarding a software system. FAME was developed as part of the SUPERSEDE EU project.

Figure 1 presents a general overview of FAME. This overview shows schematically how FAME can support the requirements elicitation process. As shown, the *End-users* provide their *User feedback* through a *Feedback component*, whereas *Runtime events* from the *Host Application* are being monitored through a *Monitoring component* and stored in a *Data Lake*. The *Combiner* aggregates the data from both sources using an *Ontology* and presents the combined data to the *Requirements Engineer* who will elicit a *New Requirement*.

A more detailed view of FAME is presented in Fig. 2. The *Data Acquisition* components of FAME collect the user feedback and runtime events simultaneously (left side of Fig. 2). The collected data is sent to the *Data Storage and Combination* components (right side of Fig. 2), which store the data following a predefined structure. The data is then interpreted through an *Ontology* that integrates the semantics of both sources. In this sense, the requirements engineer has the combined information to assess the need for a new requirement and elaborate such new requirement.

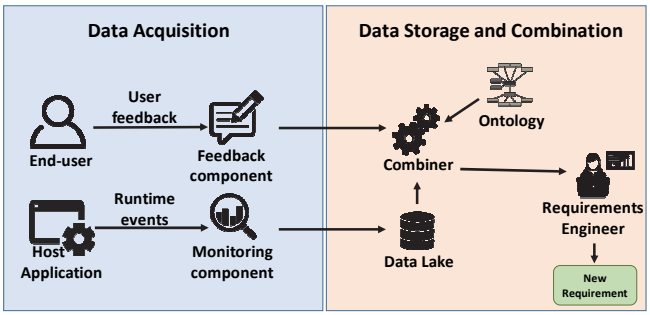

Fig. 1. General overview of FAME supporting the requirements elicitation process.

[2] https://github.com/supersede-project/monitor_feedback

A. Data Acquisition

The architecture of the *Data Acquisition* part of FAME is depicted on the left side of Fig. 2. It is composed of three packages: (1) *Feedback Management Node,* (2) *Monitoring Management Node,* and (3) *Orchestrator Node*, where each package includes various components. User feedback is acquired using the *Feedback Management Node* package, which has been implemented for the Web and Android. The implementation is in the form of a library, which a developer integrates into a *Host Application*. For this, the developer inserts a few lines of code into the application code to invoke the functionalities of the user feedback acquisition. As an example for a web application, the piece of code includes a few links to stylesheets and scripts, as well as a hyperlink to the feedback button.

Feedback Management is the main component that manages and configures the feedback dialogue and finally sends the collected feedback to the *Data Storage and Combination* components. The feedback dialogue consists of one or several so-called *Feedback Mechanisms* including (i) a text feedback mechanism for free text comments; (ii) a rating feedback mechanism for classifying experience usage (e.g., star rating, emoticons); (iii) a screenshot feedback mechanism for visualising the feedback issue with a screenshot that can even be annotated with marks; (iv) an audio feedback mechanism for spoken feedback documentation; (v) a category feedback mechanism for feedback classification (e.g., problem, praise) by using multiple selection boxes, and (vi) an attachment feedback mechanism for additional file upload.

Each time that a feedback dialogue is shown to an end-user, *Feedback Management* requests the latest configuration from the *Orchestrator Node* package, specifically the *Orchestrator CORE* component. The *Orchestrator CORE* component provides APIs that allow a system administrator to define or update configurations of feedback dialogues. The configuration received defines what feedback mechanisms, including what features and in which order they should be presented to the end-user. The configuration also defines how the feedback dialogue should be triggered either by the end-user, for example by pressing a feedback button (push), or automatically under certain conditions (pull). As soon as the *Feedback Management* receives feedback data from end-users, it transfers the feedback to the *Data Storage and Combination* components. The details are explained in Section III-B.

Monitoring Data is collected through the *Monitor Management Node* package, which includes several *Monitoring Tools* following a Service-Oriented Architecture (SOA). The *Monitoring Tool Manager* is the main component of this package and manages the different *Monitoring Tools*. This component is a RESTful service that receives the instructions to run a particular monitoring task from *Orchestrator CORE* and dispatches the request to the specific *Monitoring Tool*, which can fulfil the required task. Examples of the *Monitoring Tools* include (i) a user events monitoring tool to obtain the clickstream and navigation path of end-users in web applications; (ii) an infrastructure monitoring tool to collect infrastructure related metrics (e.g., disk, memory, and CPU consumption of a server), and (iii) a QoS monitoring tool to compute the response time and availability of web services.

The system administrator can deploy *Monitoring Tools* of interest. After deployment, a *Monitoring Tool* may require additional integration activities. For the *Monitoring Tools*, the user events monitoring tool requires the integration of a JavaScript code in the *Host Application*. The infrastructure monitoring tool requires credentials to access the server to monitor through SSH. Only the QoS monitoring tool does not require

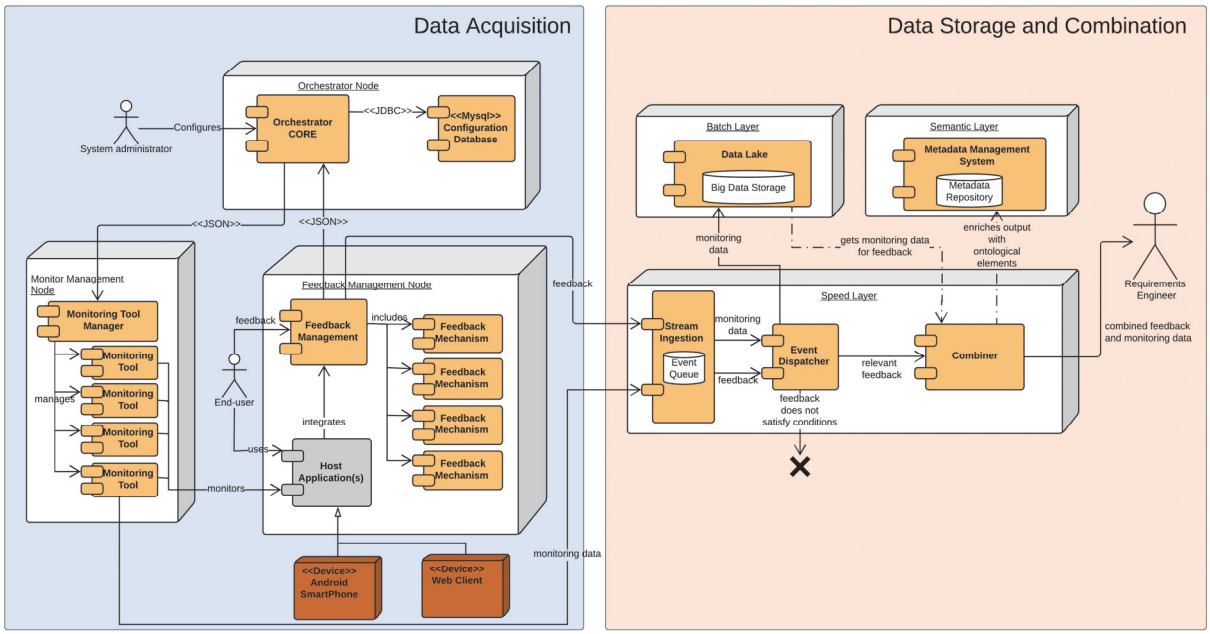

Fig. 2 FAME Architecture.

additional steps, as it follows an active monitoring approach (i.e., it periodically invokes the web service and computes the response time and availability). The system administrator can then activate or deactivate integrated monitors, as well as (re)configure them on demand (e.g., change the monitoring frequency, the metrics to collect).

The system administrator can then activate or deactivate integrated monitors, as well as (re)configure them on demand (e.g., change the monitoring frequency, the metrics to collect).

If needed, a new *Monitoring Tool* can be implemented and integrated with the *Monitoring Tool Manager* (e.g., to carry out a monitoring task that is not supported by any of the *Monitoring Tools*). To do so, the system administrator should implement a RESTful service embedding the new *Monitoring Tool*. Such RESTful service should be compliant with an API[3] that is common for all *Monitoring Tools* and is used by the *Monitoring Tool Manager* to configure and run *Monitoring Tools*.

B. Data Storage and Combination

The *Data Acquisition* components of FAME can produce large quantities of data. In particular, the *Monitoring Tools* provide a continuous data stream consisting of runtime events from the *Host Application*. Moreover, end-users can provide feedback within the *Host Application*, which can also result in high-data volumes. As a result, FAME needs to deal with Big Data, which is supported by the *Data Storage and Combination* components of FAME (see right side of Fig. 2). These components instantiate a Software Reference Architecture for semantic-aware Big Data systems [28], which decouples Big Data processing into the *Speed Layer* (for real-time data processing) and the *Batch Layer* (for offline data processing). Its heart is the *Semantic Layer*, which enables data governance using Semantic Web technologies.

To process feedback and monitoring data from different sources, FAME exploits an ontology stored in the *Metadata Management System*. The ontology provides a formal, machine-readable conceptualisation of the domain of interest as well as the key concepts of feedback (e.g., *Rating*, *Message*) and monitoring data (e.g., *URL*, *ElementText*). Figure 3 depicts a fragment of the ontology at the high abstraction level as used by SEnerCon (see Section IV), where feedback and monitoring data are linked via their shared schema elements (*User*, *Timestamp*, *Application*; bold framed, grey boxes in Fig. 3). The feedback and monitoring data can also be related via one or more domain-specific concepts, which we generally depict as the *DomainConcept* element (bold framed box in the bottom centre of Fig. 3). For instance, a feedback entry discusses an issue related to the domain concept invoice; thus we look for monitoring data also related to invoicing. The integration and mapping of different schema elements in the ontology are based on our approach for ontology-mediated queries [30]. We then detect that both feedback and monitoring data sources have shared schemas or domain-specific concept elements. This allows us to define a query that automatically joins both sources and presents an integrated result.

In addition to data processing, FAME provides custom functions for data pre-processing, which can filter feedback relevant for requirements elicitation, such as feedback categorised as feature requests, or negative comments. To this end, FAME adopts machine learning techniques to automatically identify the feedback category (e.g., bug, feature request) and feedback sentiment (positive, negative, or neutral). We specifically employ Multinomial Naive Bayes classifiers for such predictive tasks [29].

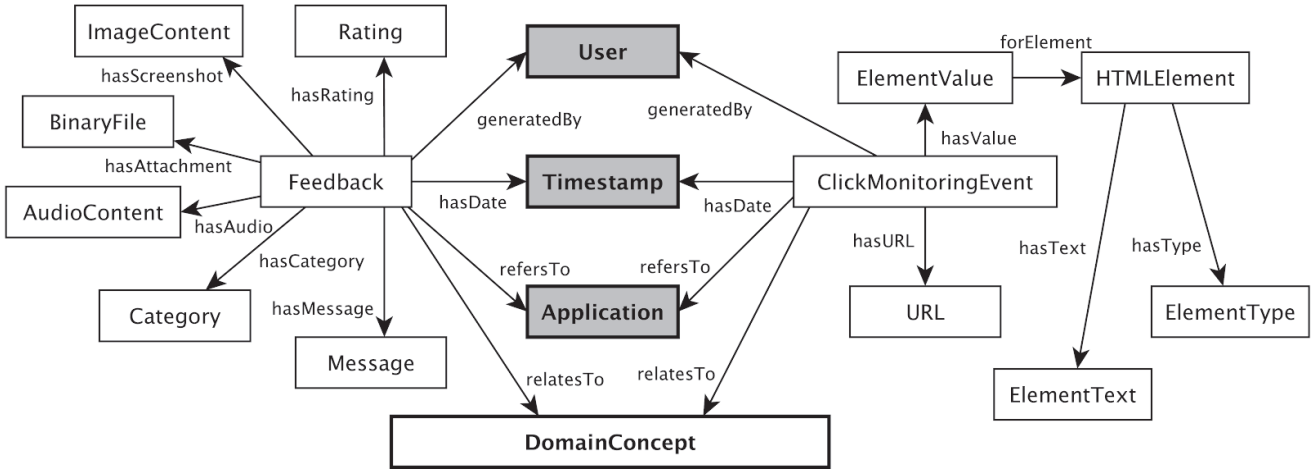

Fig. 3 Excerpt of the ontology.

[3] https://github.com/supersede-project/monitor_feedback/blob/master/monitoring/src/main/java/monitoring/controller/ToolInterface.java

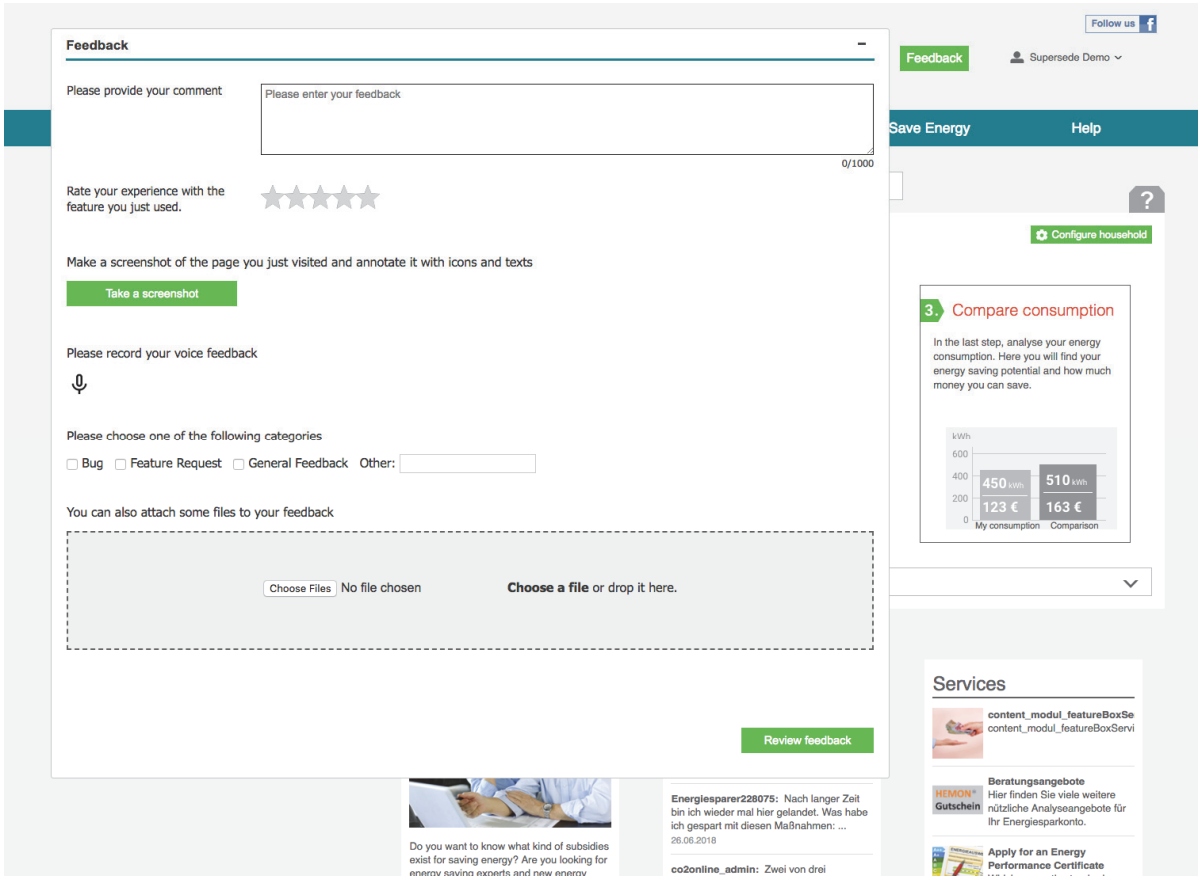

Fig. 4 Feedback dialogue.

In the context of *Data Storage and Combination* components, data flows as follows. The *Stream Ingestion* receives data coming from *Feedback Management* and *Monitoring Tools*, which are then consumed by *Event Dispatcher*. The *Event Dispatcher* then checks and routes the data in the following way: Monitoring data is forwarded to and stored in the *Data Lake* - a massively distributed storage system - so that it can be retrieved for processing at a later stage. In parallel, the *Event Dispatcher* applies pre-processing techniques to the feedback data. This allows the automatic filtering out feedback that may not be relevant for the requirements engineer. For instance, they might be interested only in feature requests with negative sentiment. Next, the output of the filtering function integrated into the *Event Dispatcher* is forwarded to the *Combiner*, which retrieves from the *Data Lake* the monitoring data that is related to the feedback data. The *Combiner* joins both types of data via their shared schema elements (i.e., User, Timestamp, and Application identifiers). Once combined, *Combiner* forwards the data to the requirements engineer for further interpretation.

It is worth noting that due to the flexible configuration features of FAME, several different options exist on how FAME can be implemented. Furthermore, the loosely coupled design of FAME enables the addition of more data sources and tools that the ones here presented.

IV. Validation

A. Deployment and Configuration of FAME

We tailored FAME to the characteristics of the iESA (*Host Application*) and the needs of SEnerCon and SUPERSEDE. As a result, decisions about the feedback dialogue configuration received from the *Orchestrator Node* (e.g., mechanism order, instructions) were not only taken together with SEnerCon but were also driven by research aims. To obtain as much information as possible from the user feedback, we decided that the *Feedback Management Node* be deployed and configured to include all the *Feedback Mechanisms* available in FAME. As a consequence, all these feedback mechanisms are presented to the iESA end-user once they press a visible feedback button (push) that is available from every page in the iESA[4] (see Fig. 4). In this single-page feedback pop-up, the iESA end-user must first provide their comment in a mandatory text field with a limitation of 1000 characters (text feedback mechanism). Second, they can use a star rating to express their experience with the feature they used (rating feedback mechanism). Third, by using the screenshot feedback mechanism, they can take and edit (e.g., draw arrows) a screenshot of the page in the background. As fourth input, they can record their

[4] For the German feedback dialogue used in our validation, please see http://co2onl.in/5f21b50d

voice (audio feedback mechanism). Next, they can indicate the category (or categories) of their feedback, e.g. “Bug”, “Feature Request”, “General Feedback”, “Other”; category feedback mechanism). Note that for the feedback type classification we have decided to allow the feedback sender to choose multiple categories because we have seen in previous feedback that SEnerCon has received that their end-users sometimes communicate more than one issue in the same feedback message (e.g., starting with general praise, followed by a problem description).

As the last input, the iESA end-user can upload further files (attachment feedback mechanism). After providing all their input, they click on the “next” button, check the summary of their feedback in a second dialogue page, and finally press the “send” button.

The *Monitoring Management Node* was deployed and configured with just one *Monitoring Tool*. In particular, SEnerCon was interested in monitoring the behaviour of its end-users, and hence, we deployed the user events monitoring tool to obtain the clickstream and navigation path of end-users in iESA. The infrastructure and QoS monitors were not deployed since SEnerCon was not interested in the metrics gathered by these *Monitoring Tools*. In the Terms of Service, which was included in the registration form end-users accept before using the iESA web application, end-users were informed they were being monitored.

Data Storage and Combination components were deployed without configuring any filter for the feedback in the *Event Dispatcher*. This is because the amount of feedback expected by SEnerCon about iESA, based on their experience with their other feedback communication channels, was not large enough to justify a filter and SEnerCon wanted to analyse all the feedback obtained without any pre-processing. Nevertheless, filters might be required in other cases, such as when there is a significant amount of feedback received that is not relevant for the generation of a new requirement.

B. Validation Protocol and Execution

The validation protocol distinguished two key stages: feedback and monitoring data gathering; and requirements elicitation using such data.

The feedback and monitoring data were collected between October 1st, 2017 until January 31st, 2018 (4 months). The requirements elicitation process was conducted through a workshop. The workshop involved a researcher and an employee from SEnerCon (both are authors of this paper). The employee who acted as SEnerCon representative was one of the leading developers in the production of the iESA.

The workshop was divided into two phases. In the first phase, the SEnerCon representative had to elicit requirements considering only feedback data (as SEnerCon has done so far). In the second phase, they had to elicit further requirements or refine the previously elicited ones considering the combination of feedback and monitoring data. With this validation procedure, we were able to assess the benefits of combining feedback and monitoring data concerning using just feedback data as an information source in the requirements elicitation process.

To execute the first phase of the workshop, the researcher presented the feedback obtained by FAME to the company representative who identified which feedback entries were relevant to elicit a requirement (as not all feedback might lead to a requirement). Then, for the feedback entries that were identified as relevant, they elicited the requirement and documented it by filling in a predefined template (see Table I). This template included the following fields: the id of the requirement; the ID of the feedback entry that triggered the requirement; a description of the requirement (with an optional satisfaction criteria); the priority of the requirement (with five possible values from very high to very low); and a field to document other observations that the representative may consider relevant. An example of a requirement obtained in the first workshop phase is also depicted in Table I. The execution of the first workshop phase lasted two hours and fifteen minutes with thirty minutes to identify which feedback entry was relevant, and one hour and forty-five minutes to elicit and document the requirements.

To execute the second workshop phase, the researcher provided the relevant feedback entries, identified in the previous workshop phase, combined with the monitoring data to the company representative. The presented monitoring data included the clickstream and navigation path of the end-user who provided the feedback and how many end-users used the same functionality. While going through the combined feedback and monitoring data, the company representative identified further requirements and modified (some of) the previously documented requirements. The researcher annotated these changes. The execution of the second workshop phase lasted two hours and thirty minutes.

TABLE I. EXAMPLE OF AN ELICITED REQUIREMENT AFTER THE FIRST WORKSHOP PHASE

Requirement ID	R1
Feedback ID related to the Requirement	F3
Description of the Requirement	The user should be able to see meter readings of all meters, not depending on the state of the meter (exchanged, active, smartmeter) in the Android application.
Priority of the Requirement	Low
Other Observations (if any)	None

C. Validation Results

FAME was successfully used in practice. In the defined period, FAME collected thirty-one feedback entries from twenty-four end-users. Figure 5 summarises how the end-users used the feedback mechanisms to communicate their feedback.

End-users categorised the feedback as “Bug”, “Feature Request”, “General Feedback” (multiple categories were allowed per each feedback); the category “Other” was not chosen.

Screenshots and attachments were used in seven cases, and star-ratings were provided in twenty-four cases.

In the same period, FAME collected user events from all 5.185 end-users who logged-in during the period, regardless if they provided feedback or not. FAME collected 957.260 clicks in the clickstream (including 936.740 left-clicks, 6.547 right-clicks, and 13.973 double-clicks), and 160.888 navigation actions (i.e., steps of the navigation path). As an example, an excerpt of the clickstream of one end-user is shown in Fig. 6 (translated from German).

In the first workshop phase, sixteen out of thirty-one feedback entries collected with FAME were identified as relevant by the company representative and led to nineteen requirements (in three cases, the feedback entry led to two requirements). The fifteen feedback entries that were considered not relevant to elicit a requirement were bug reports and customer service related issues.

In the second workshop phase, the company representative analysed the combined feedback and monitoring data. The combined monitoring data covers the time span between user login and when the feedback is sent and includes the actions of the end-user who provided the feedback, as well as the list of end-users, who did not provide feedback but used the same actions as the feedback provider. It is worth noting that by applying the ontology-mediated queries, only the monitoring data related to the sixteen relevant feedback points were considered. From 957.260 clicks, only 2.164 were presented and analysed by the SEnerCon representative.

During the analysis of the combined data, the company representative found one additional requirement and modified four requirements they had documented in the first workshop phase. Below we describe some examples of how those requirements were modified as well as how the additional requirement was identified.

Monitoring data proved to be useful to either confirm or refute the priority of a requirement. For example, an end-user requested that an existing feature in the iESA web application should also be present in the Android application. Once the feedback was analysed, the requirement was considered "low priority" as the feature was not very relevant. Monitoring data confirmed such perceptions, as this feature was only used by 11 out of the 5.185 iESA end-users in the web application during this period.

In another example, an end-user requested a new feature to transfer data from their old household to a new household. One feedback from another end-user seemed to be about something very similar. As a first impression, such feature request seemed very relevant, and the documented requirement was assigned with a "high priority". Monitoring data, however, disproved such prioritisation as the number of end-users who looked for the data of their old household and had a new household in this period was just two, including the end-user who provided the first feedback in this example. For the second similar feedback, the monitoring data showed what the end-user was trying to achieve and clarified that they were asking for something else; they wanted to enter data from previous years to the same household instead of transferring old data to a new household. Ultimately, this led to an entirely different and new requirement.

Monitoring data was also used to decide among two possible variants of the same requirement. For example, an end-user requested that the system should describe what "conversion factor" meant and how it should be calculated when entering consumption data taken from the gas bill. Considering only the feedback entry, a first conclusion by the SEnerCon representative was that the iESA web application should compute this "conversion factor" automatically when the end-user enters other parameters of the gas service. However, monitoring data of this end-user showed that they did not struggle to understand and compute the needed "conversion factor" as

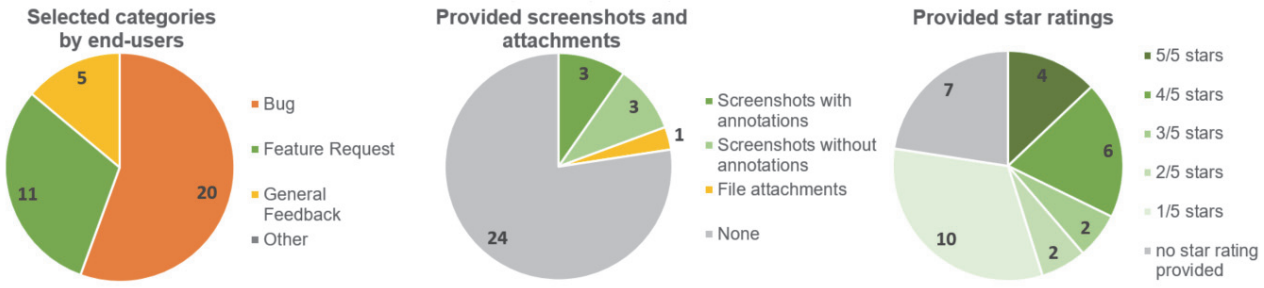

Fig. 5. Overview of feedback collected with FAME.

timestamp	event Type	element type	element ID	text	element value	URL
01/10/2017 4:36:03	click	INPUT	meter reading			https://www.energiesparkonto.de/esk/content/startPage/?portal_id=co2online
01/10/2017 4:36:09	click	INPUT	enterMeterreading_save		Save meter reading	https://www.energiesparkonto.de/esk/content/startPage/?portal_id=co2online
01/10/2017 4:37:30	click	SELECT	view	Years \t Months \t Weeks \t Days\t Hours \t	year	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:37:32	click	OPTION		Months	month	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:38:21	click	SELECT	view	Years \t Months \t Weeks \t Days\t Hours \t	month	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:38:23	click	OPTION		Days	day	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom
01/10/2017 4:39:07	click	A	logout	Logout	undefined	https://www.energiesparkonto.de/esk/content/bereichePage/?bereich=strom

Fig. 6. Excerpt of the clickstream of one end-user collected with FAME
(columns *element ID*, *text* and *element value* have been translated here from German to English).

they spent less than one minute to introduce the "conversion factor" and all the data from the gas bill. Moreover, such functionality was only used by 203 end-users and in total 332 times. This means that end-users who use that functionality use it on average less than twice in a four-month period. With this information, the effort to implement a functionality to compute the "conversion factor" was not justified, and, the requirement was documented as "the term conversion factor should be explained when requested in the gas consumption form".

D. Threats to Validity

Threats to Internal Validity. The first threat is a possible bias in the data analysis conducted by the company representative. SEnerCon is a partner in the SUPERSEDE research project and is aware of the effort the research partners spent on the development of FAME. As a result, we encouraged the representative to share any thoughts with us, including critiques or doubts. Moreover, the validation workshop was moderated by the main author, and they might have unconsciously guided the requirements elicitation process in the research authors' desired direction. To limit the workshop moderator's influence, the research authors had defined a protocol before the validation execution including a semi-structured workshop guideline. The workshop moderator was also not allowed to express their own opinions.

A second threat is about the requirements elicitation expertise of the company representative. Although they are an expert in the iESA web application and knows the existing requirements, they are not an expert in requirements elicitation. In the workshop, this might have influenced the number of feedback points they indicated as relevant for requirements elicitation, the final number of elicited requirements as well as their modifications.

Third, the low number of feedback points obtained is limiting our evaluation method. We have been able to prove the feasibility of FAME and provided an initial qualitative evaluation that demonstrated how FAME improves the requirements elicitation process. However, we could not conduct a quantitative evaluation as it would require a higher amount of feedback data points collected with FAME.

Threats to External Validity. One limitation to the generalizability of our results is that only one member of one software company was involved in our validation study. Secondly, regarding the selected *Host Application*, we have focused on the integration of FAME in a web application in the domain of energy saving. Thirdly, we run our feedback and monitoring data collection with only one particular configuration of FAME. Finally, we could not investigate the support of FAME for requirements engineers with different expertise levels.

V. Conclusions and Future Work

In this paper, we have presented FAME, a framework for the combined and simultaneous collection of feedback and monitoring data in Web and mobile contexts to support continuous requirements elicitation.

FAME proposes managing both acquired feedback and monitoring data through the same infrastructure as well as combining and structuring them employing an ontology. The FAME architecture is split into two main groups of components: *Data Acquisition* components, and *Data Storage and Combination* components. The *Data Acquisition* components integrate a feedback acquisition tool with multiple *Feedback Mechanisms* and a monitor manager with multiple *Monitoring Tools* that collect user feedback and runtime events respectively. The *Data Storage and Combination* components instantiate a Software Reference Architecture for semantic-aware Big Data systems that structure and combine the collected data via an ontology that maps the relationship between user feedback and runtime events.

To validate our solution and assess in which situations the combination of monitoring and feedback data provided valuable information to elicit new requirements, we deployed FAME in a web application of a German SME and conducted a requirements elicitation process through a workshop using the data obtained by FAME. Results showed that FAME was deployed and used in an industrial context to combine feedback and monitoring data, also bridging a research gap in RE. Moreover, our first validation results identified a few examples where the combination of feedback and monitoring data could improve the requirements elicitation process in contrast to just considering only the feedback. Due to the small data set and the data fragments used in our study, conclusions regarding quantitative, situation-specific benefits of combined feedback and monitoring data cannot be drawn. However, we assess our presented flexible and extensible FAME framework as a robust tool solution to advance research on combined feedback gathering and monitoring as a mean to support continuous requirements elicitation.

As future work, we plan to validate FAME with companies of different domains, sizes, and requirements elicitation processes, as well as with different amounts of end-users involved. This future validation will be on various configurations of FAME including both web and mobile *Host Applications*, more *Monitoring Tools*, and various filters for data preprocessing. Such future work will allow us to assess the benefits of combining user feedback and monitoring data and to derive recommendations and best practices, e.g., which amount of which data types in which aggregation level are most useful to combine feedback and monitoring data.

We also plan to extend FAME in several directions. For example, we may use the monitoring data first to trigger a new requirement and then combine the monitoring data with the related feedback. Moreover, we may integrate further user feedback channels, such as social media into the FAME framework; and further *Monitoring Tools*, such as sensors embedded in mobile devices. Because FAME requires a continuous flow of feedback data to support continuous requirements elicitation, we also plan to investigate how we can best encourage end-users to provide feedback. This, in turn, might result in further extensions of FAME, for example, feedback-to-feedback or gamification components. Regarding the *Data Storage and Combination* components, we plan to incorporate automatic clustering of feedback data and advanced aggregation of monitoring data. This would improve how the infor-

mation is presented to the requirements engineer to elicit requirements (e.g., grouping similar feedback, showing monitoring data with computed metrics instead of only runtime events).

ACKNOWLEDGMENT

The authors thank the anonymous reviewers. A special thanks goes to Colin Venters for his support. This work was supported by the European Commission within the SUPER-SEDE project (Agreement No. 644018).

REFERENCES

[1] P. Forbrig. "Does Continuous Requirements Engineering need Continuous Software Engineering?" in Proceedings of Requirements Engineering: Foundation for Software Quality (REFSQ), 2017.

[2] M. Vierhauser, R. Rabiser, P. Grünbacher. "Requirements monitoring frameworks: A systematic review" in Information and Software Technology (IST), Vol. 80, pp. 89–109, 2016

[3] L. V. Galvis Carreño, K. Winbladh. "Analysis of user comments: An approach for software requirements evolution" in Proceedings of International Conference on Software Engineering (ICSE), 2013.

[4] R. Snijders, F. Dalpiaz, M. Hosseini, A. Shahri, R. Ali. "Crowd-centric Requirements Engineering" in Proceedings of Utility and Cloud Computing (UCC), 2014.

[5] N. Seyff, I. Todoran, K. Caluser, L. Singer, M. Glinz. "Using popular social network sites to support requirements elicitation, prioritization and negotiation" in Journal of Internet Services and Applications (JISA), Vol. 6, issue 1, pp. 7:1–7:16, 2015.

[6] L. Guzman, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, M. Oivo. "How can quality-awareness support Rapid software development? A research preview" in Proceedings of Requirements Engineering: Foundation for Software Quality (REFSQ), 2017.

[7] O. Brill, E. Knauss. "Structured and unobtrusive observation of anonymous users and their context for requirements elicitation" in Proceedings of Requirements Engineering (RE), 2011.

[8] S. Wellsandt, K. A. Hribernik, K. D. Thoben. "Qualitative Comparison of Requirements Elicitation Techniques that are Used to Collect Feedback Information about Product Use" in Procedia CIRP, Vol. 21, pp. 212–217, 2014

[9] W. Maalej, M. Nayebi. T. Johann, G. Ruhe. "Toward Data-Driven Requirements Engineering" in IEEE Software, Vol. 33, issue 1, pp. 48–54, 2016.

[10] N. Seyff, G. Ollmann, M. Bortenschlager. "AppEcho: a user-driven, in situ feedback approach for mobile platforms and applications" in Proceedings of International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2014.

[11] A. Rashid, J. Wiesenberger, D. Meder, J. Baumann. "Bringing developers and users closer together: the OpenProposal story" in Proceedings of Process Innovation for Enterprise Software (PRIMIUM), 2009.

[12] J. Hess, L. Wan, B. Ley, V. Wulf. "In-situ Everywhere: A Qualitative Feedback Infrastructure for Cross Platform Home-IT," in European Conference on Interactive TV and Video (EuroiTV), pp. 75–78, 2012.

[13] I. Morales-Ramirez, A. Perini, R. Guizzardi. "An ontology of online user feedback in software engineering" in Applied Ontology, Vol. 10, issue 3-4, pp. 297–330, 2015.

[14] W. Maalej, H. J. Happel, A. Rashid. "When users become collaborators: towards continuous and context-aware user input" in Proceedings of Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), 2009.

[15] M. Stade, F. Fotrousi, N. Seyff, O. Albrecht. "Feedback Gathering from an Industrial Point of View" in Requirements Engineering Conference (RE), 2017.

[16] D. Zowghi, F. Rimini, M. Bano. "Problems and challenges of user involvement in software development: an empirical study." in Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), 2015.

[17] D. Pagano, B. Bruegge. "User involvement in software evolution practice: a case study," in Proceedings of the International Conference on Software Engineering (ICSE), 2013.

[18] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, C. Weiss. "What Makes a Good Bug Report?" in IEEE Transactions on Software Engineering (TSE), Vol. 36, issue 5, pp. 618–643, 2010.

[19] F. Dalpiaz. "Social threats and the new challenges for Requirements Engineering" in Proceedings of Requirements Engineering for Social Computing (RESC), 2011.

[20] T. Johann, W. Maalej. "Democratic mass participation of users in Requirements Engineering?" in Proceedings of Requirements Engineering Conference (RE), 2015.

[21] W. Robinson. "A roadmap for comprehensive requirements modeling" in Computer. Vol 5, issue 5. pp. 64–72, 2009.

[22] L. Liu, Q. Zhou, J. Liu, A. Cao. "Requirements cybernetics: Elicitation based on user behavioral data" in Journal of Systems and Software (JSS), Vol. 124, pp. 187–194, 2017.

[23] F. Fotrousi, S. A. Fricker, M. Fiedler. "Quality requirements elicitation based on inquiry of quality-impact relationships" in Proceedings of Requirements Engineering conference (RE), 2014.

[24] D. Dzvonyar, S. Krusche, R. Alkadhi, B. Bruegge. "Context-aware user feedback in continuous software evolution" in Proceedings of the International Workshop on Continuous Software Evolution and Delivery (CSED), 2016.

[25] J. Dabrowski, F. M. Kifetew, D. Muñante, E. Letier, A.Siena, A. Susi. "Discovering Requirements through Goal-Driven Process Mining" in Second Workshop on Crowd-Based Requirements Engineering (CrowdRE), 2017.

[26] F. Fotrousi, S. A. Fricker. "QoE probe: A requirement-monitoring tool" in Proceedings of Requirements Engineering: Foundation for Software Quality (REFSQ), 2016.

[27] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, J. A. Landay. "MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones" in Proceedings of International Conference on Mobile systems, applications and services (MobiSys), 2007.

[28] S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren, D. Valerio. "A software reference architecture for semantic-aware Big Data systems" in Journal of Information and Software Technology (IST), vol. 90, pp. 75–92, 2017.

[29] E. Guzman, M. Ibrahim, M. Glinz. "A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution" in Proceedings Requirements Engineering Conference (RE), 2017.

[30] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, S. Vansummeren. “An integration-oriented ontology to govern evolution in big data ecosystems” in Information Systems, (2018).