Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use

Dustin Wüest[1], Farnaz Fotrousi[2], and Samuel Fricker[1,2]

[1]FHNW University of Applied Sciences and Arts Northwestern Switzerland,
Institute for Interactive Technologies,
Windisch, Switzerland
[dustin.wueest, samuel.fricker]@fhnw.ch

[2] Blekinge Institute of Technology,
Software Engineering Research Laboratory (SERL-Sweden),
Karlskrona, Sweden
[farnaz.fotrousi, samuel.fricker]@bth.se

Abstract. [Context and motivation] To validate developers' ideas of what users might want and to understand user needs, it has been proposed to collect and combine system monitoring with user feedback. **[Question/problem]** So far, the monitoring data and feedback have been collected passively, hoping for the users to get active when problems emerge. This approach leaves unexplored opportunities for system improvement when users are also passive or do not know that they are invited to offer feedback. **[Principal ideas/results]** In this paper, we show how we have used goal monitors to identify interesting situations of system use and let a system autonomously elicit user feedback in these situations. We have used a monitor to detect interesting situations in the use of a system and issued automated requests for user feedback to interpret the monitoring observations from the users' perspectives. **[Contribution]** The paper describes the implementation of our approach in a Smart City system and reports our results and experiences. It shows that combining system monitoring with proactive, autonomous feedback collection was useful and surfaced knowledge of system use that was relevant for system maintenance and evolution. The results were helpful for the city to adapt and improve the Smart City application and to maintain their internet-of-things deployment of sensors.

Keywords: Requirements Monitoring, User Feedback, Requirements Elicitation, Smart City

1 Introduction

Software maintenance and evolution constitute a large part of the work of software engineers [1]. From a requirements engineering perspective, one of the goals is to gather user feedback about released software to identify user needs that can be translated into requirements for future releases [2]. Various efforts have been spent to monitor system use, elicit user feedback, and analyse the obtained data [3]. Common methods for gathering feedback are hotlines, email, contact forms, and ticket systems, feedback forms embedded in software, and user feedback mechanisms of app stores [2]. This data has been mined and analysed with the aim of extracting requirements [4].

In the context of a European-Asian innovation project, Wise-IoT (www.wise-iot.eu), we have implemented a FAME-like approach of system monitoring and feedback forms [5]. We aimed at understanding the problems of the developed prototype systems and identify opportunities to evolve them to increase value creation and quality-of-experience. The monitoring data turned out to be difficult to interpret, and the user feedback requests were disturbing for users [6] or lacked enough information about the context to which the feedback applied.

These challenges encouraged us to extend the FAME approach by combining system monitoring and user feedback with autonomously generated proactive requests for user feedback. We monitored the fulfilment of end-user goals with data gathered from internet-of-things (IoT) devices to detect whether users are in an interesting situation, such as having achieved a goal or having deviated from the pathway towards the goal. With this approach, we could issue requests for user feedback in a targeted way, making the feedback requests relevant for the concerned users and reducing our dependency on luck for useful feedback to be received.

In this paper, we present our approach of combining system monitoring with autonomously triggered user feedback and report on its evaluation in a Smart City application for parking management. The main research question for the evaluation was: *Do the combination of system monitoring and autonomously triggered user feedback provide added value to system evolution?* We were interested to see whether autonomous triggering of feedback requests allows eliciting requirements that would not have been identified by just using a passive monitoring and feedback collection approach.

The remainder of the paper is structured as follows. Section 2 gives an overview of related work and background. Section 3 presents the approach. Section 4 describes the evaluation. Section 5 presents the obtained monitoring and feedback data and answers the research question. Section 6 discusses the results. Section 7 concludes.

2 Combined Data Gathering for System Evolution

User requirements are changing, which is the primary driver for evolving a software system. Planning the system's evolution needs knowledge of when and what requirements have changed and how to enhance the system. The knowledge can be acquired by frequent observation or monitoring of how the system is used [7] and checking whether it meets the users' requirements [8].

Sometimes, engineers have assumptions of how a system should be evolved and take the proactive approach of implementing and validating a prototype that exposes the change to users at runtime [9]. These innovation experiments generate insight for testing the assumption and deciding whether the change should be sustained or abandoned [10]. Lean Start-up is an example of the innovation experiment method designed for small companies [11] and also adopted in large companies [12]. Critical for the success of innovation experiments is again the monitoring of the system use, e.g. to check the use of the innovation, and the collection of user feedback, e.g. to check whether the innovation generates value for users.

Monitoring the system use allows requirements engineers to determine whether and to what degree the implemented system is meeting its requirements at runtime [13]. The insertion of code or sensors into a running system allows the developers to continuously check the system health, observe users, record their activities and study the system's behaviour [14]. Such monitoring enables requirements engineers to detect requirements violations, e.g. system failures, and react fast to evolve the system [15]. Furthermore, observing the user activities, such as a sequence of feature usage, duration, and other contexts, enables requirements engineers to understand the user needs better [3]. However, such monitoring data alone might not directly show whether users are satisfied, what exactly users require, and what the details of the requirements are.

Feedback given by users is another source of information to understand user needs how satisfied the users are with the system [16]. Several feedback tools have been designed to collect such information with user feedback. These tools are either offered standalone or are embedded into the system [17, 18]. The feedback tools trigger feedback forms either by a user's request, e.g. pressing the feedback button, or by a system request, e.g. by an automatic pop-up window [19]. Such feedback forms enable users to communicate bug reports, feature requests, and praise [20]. The feedback may be collected as a simple combination of free text, selected categories, ratings, and screenshots with annotations [21, 22]. Regardless of the dialogue design, several studies describe challenges of analysing and interpreting user feedback, especially when information about the context is missing that the feedback applies to [2, 23].

Monitoring and user feedback collection at runtime together supports the communication of user needs while capturing information about the context. Seyff et al. proposed to connect user feedback with features of the user interface [17]. Fotrousi et al. proposed to correlate the users' Quality of Experience and with the system's Quality of Service [24]. Oriol et al. proposed a generic framework for combining the collection of feedback and monitoring data [5].

So far, the combination of monitoring and feedback has been validated with passively collected user feedback. While relevant insights could be generated, the passive approach limited developers in targeting feedback collection on interesting situations of system usage and generated the risk of collecting irrelevant or even fake feedback [25]. To avoid this problem, we rely on proactive, autonomous requests for user feedback when an interesting situation in the use of a system is detected.

3 Proactive, Autonomous Gathering of User Feedback

3.1 Control Loop

Our approach of eliciting and using monitoring and feedback data is based on the control loop for self-adaptive systems proposed by Cheng et al. [26]. The control loop allows collecting data, analysing that data with the help of rules or models of expected system usage, deciding how to act by interpreting these insights, and acting according to these decisions. A self-adaptive system fully automates this loop by examining, introspecting, and modifying itself at runtime. An evolving system keeps the engineers in the loop, allowing them to understand the system's achievements, problems, and needs for evolution.

Our approach can be used to build a control loop for a system to be evolved or maintained. The system may be a functional prototype or an operational system. The control loop spans the system runtime, the technical environment in which the system runs, the users of the system, and the engineers doing the development and maintenance.

The control loop is parametrised with the endpoints and model of the data that is to be collected for analysis and may include data from the system, e.g. generated by IoT sensors, data about the users, e.g. user preferences, and data generated as a result of user-system interaction, e.g. click-trails. The data parametrisation includes the definition of the questionnaires for collecting user feedback.

Another type of parametrisation concerns the analysis that is used to detect the interesting situations in which a proactive, autonomous request should be issued to a user for collecting feedback from that user. A there are many potential ways of defining the interestingness of such a situation, we have opted for a flexible plug-in approach. The currently developed plug-in assumes that a user tries to achieve a goal with a journey that can be expressed by a sequence of subgoals that the user will achieve while pursuing the goal.

We have defined interestingness of a situation with respect to the user's fulfilment of the goal and adherence to the journey: a) the goal has been achieved and b) the user has deviated from the following subgoal, or the goal if no subgoals remain, he was expected to achieve. We have implemented the goal monitoring based on the concepts suggested by Qian et al. [27]. This interpretation of interestingness allows asking the user to judge his satisfaction with the proposed goal and journey and offering rationales for the judgment and, if relevant, the deviations.

The start of journeys, the achievement of goals, and deviations from the journeys are offered to the engineer as a stream of insights. The insights are presented as structured and semantically annotated data that include collected supporting data and user feedback and are used by the engineer for further analysis and decision-making about system evolution and maintenance. The engineer may analyse the recently generated insights, for example as part of his continuous or daily system monitoring practice. The engineer may also decide to aggregate or correlate insights based on attributes of the data included in the included data.

The last step of the control loop, the evolution and maintenance of a system, is under the control of the engineer. Maintenance may be initiated if the insights indicate that something is wrong with the deployed system and needs fixing. Evolution may be initiated if important user needs or other opportunities for value creation are discovered and prioritised according to standard roadmapping and release planning activities [1].

3.2 Implementation

For implementing the feedback loop, we have developed a component, called SAR, that may be integrated into a software system and deployed for supporting the evolution of that system. SAR can be connected to streams of data from sensors and system monitors, thus supporting *data collection*. SAR can be instrumented with plugins with models of expected system use and rules for detecting fulfilment of usage goals or deviations from pathways towards achieving these goals, thus supporting *data analysis*. To support the interpretation of the analysis results, SAR can be instrumented with questionnaires to be triggered to obtain user feedback, thus completing data collection. SAR, finally, offers an insights stream that an engineer may subscribe, thus enabling the engineer to *decide how to act*. The engineer may then act with maintenance of the system and its components or by evolving the system as part of system development.

Fig. 1 shows the integration of SAR into a system. The component may be integrated into a system that is to be maintained and evolved. SAR assumes the presence of a front-end application that offers a user interface for interacting with the users, a system back-end that offers system-specific data, and engineering tools for maintaining and evolving the system. In the current implementation, SAR offers libraries helping application developers to connect the front-end with SAR and expects a standard back-end interface, the Orion Context Broker (fiware-orion.readthedocs.io). It offers also an Orion interface that allows connecting engineering tools to the insights stream.

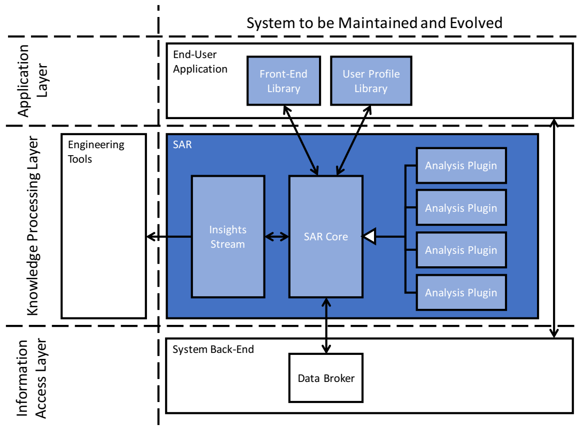


Fig. 1. Integration architecture (coloured: SAR, white: context into which SAR is integrated).

The top layer is the *Application Layer*. It contains the end-user application and two front-end libraries that we developed to simplify the integration of the SAR component, taking care of all communication to the recommender via a RESTful API. The front-end library is an adaptation of the user feedback framework from the Supersede project (www.supersede.eu).

The bottom layer is the *Information Access Layer*. It contains third-party systems and services that provide IoT data to the recommender system and the end-user applications. The layer can also include third-party services, depending on the specific use case, e.g. a street map provider, or a routing framework. In the current implementation, SAR assumes that data is provided by a broker implementing the NGSI Open RESTful API (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification).

The middle layer is the *Knowledge Processing Layer*. It contains the parts of SAR that collect and process data to generate knowledge. SAR consists of multiple modules. Each module is a web service, some of them used as plug-ins with rules or models of expected system usage and functionality for analysing the collected data.

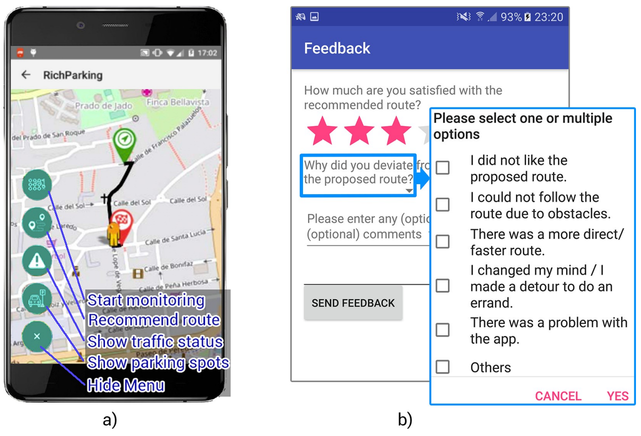
An essential plugin is the *System and Adherence Monitor*. It handles anonymous user sessions and monitors adherence to the recommendations provided to the user. It collects and combines user feedback with monitoring information and forwards the results to the insights stream.

The *Insights Stream* implements a publish-subscribe pattern to publish insights. An engineer may subscribe to the stream for obtaining monitoring data that is of interest, including measurements of the system behaviour, context, and usage and feedback of the users regarding their preferences, intentions, observations, and opinions. The stream is structured so that the engineer understands the relationships between the monitoring data and the user feedback. The tools used by the engineer to listen to the stream may be as simple as a logger that collects the insights in a format that may be inspected by an engineer. They may be as complex as a big data analytics tool that feeds packages of issues to development backlog management tools such as Atlassian Jira. Analysing the insights stream helps engineers to capture the users' needs and problems, which in turn can lead to short cycle times for improving and evolving the system.

4 Initial Evaluation

4.1 Smart City Application for Parking Management

We have evaluated our proactive, autonomous gathering of user feedback in a smart city prototype application for Android smartphones, called *Rich Parking*. The University of Cantabria (UC) had developed the application for the City of Santander in Spain. The application made use of thousands of IoT traffic and parking sensors that were deployed in the city of Santander and helped users find free parking spots within the city when they are travelling by car. Fig. 3a shows a screenshot.



How much are you satisfied with the recommended route?

Why did you deviate fro the proposed route?

Please enter any (optio (optional) comments

Please select one or multiple options

- ☐ I did not like the proposed route.
- ☐ I could not follow the route due to obstacles.
- ☐ There was a more direct/ faster route.
- ☐ I changed my mind / I made a detour to do an errand.
- ☐ There was a problem with the app.
- ☐ Others

Fig. 2. Screenshots of a) the Rich Parking application and b) a feedback form.

The application used a recommender system to generate recommendations of an unoccupied parking and a pathway to the recommended parking for end-users. The parking spot sensors provided data about the spots' current states (free/occupied) and allowed the app to display the free spots. Some of the streets contained sensors to measure traffic load and allowed the app to recommend routes that avoided traffic jams. Each recommendation consisted of a free parking spot and a fast route to the spot.

The UC team integrated our user feedback mechanism into their Android application using the front-end and user profiles libraries. We offered the SAR core as a service that we connected with the Smart City data broker from Santander using the NGSI API. For the UC team maintaining the Android application and for the Santander team maintaining the Smart City infrastructure, we offered access to the Insights Stream using an own instance of the same NGSI API.

4.2 Parametrisation of the Control Loop

We parametrized the control loop to reveal issues with the Rich Parking application (e.g., usability problems, missing functionality), the parking and pathway recommender system (e.g., bad recommendations, slow performance), the behavior of the physical IoT devices (e.g., sensors delivering wrong values), and third-party software systems (e.g., outdated street map data).

To detect interesting situations, our system mapped the recommendation to a goal tree and monitored the user's adherence to the recommendation. The parking spot was the main goal, and each street segment of the route was a subgoal. When the users' GPS location matched with a street segment, the corresponding subgoal was set to *fulfilled*. Previous subgoals that were not already marked as *fulfilled* got marked as *skipped*. We specified the interesting situations as being those when subgoals were skipped, the user achieved the main goal, or abandoned it. In these situations, our goal monitor issued a feedback request tailored to the situation.

Based on the situation, the recommender system selected the feedback to be gathered from the user. One of the following three feedback forms was then displayed:

- Type 1, the user deviated from at least 50% of the pathway: the feedback form asked about the user's satisfaction with the recommended route (star rating) and the reasons for the deviation (multiple choice and free text answer). See Fig. 2b.
- Type 2, the user adhered to the pathway but selected a parking spot other than the recommended one: the feedback form asked about the user's satisfaction with the chosen spot (star rating) and the reasons for not taking the recommended parking spot (multiple choice and free text answer).
- Type 3, the users took the recommended spot: the form asked about the user's satisfaction with the recommended route and parking spot (star ratings) and offered to comment with a free text answer.

For the users' safety, we displayed the optional short feedback forms to the user only at the end of a session when the user stopped driving. The user could then rate her experience and provide reasons for the rating. If the user mainly followed the recommended route, the form asked about the user's experience with the parking spot she took. Otherwise, the form asked about the user's experience with the route and the reason for the deviation.

For providing anonymity and control of the data collection to the end-user, only sessions were tracked and only with a random ID. The user could decide to start the monitoring and whether she agreed to send her GPS data during the monitoring regularly. The session ended when the user parked the car or cancelled the monitoring.

All events, such as the creation of a session, the start and stop of the monitoring, the user feedback, and the monitoring result of each location comparison, got augmented with a timestamp and written into the insights stream. The structure of the insights stream used the session objects as the top-level entities. This structuring allowed combining the user feedback with the events that happen in the respective session. For example, when a user provides feedback about a route recommendation, we could compare the feedback with the actual route taken by that user.

4.3 Evaluation Setup and Method

The UC team initiated a pilot study to evaluate the Rich Parking application in Santander. In this context, we performed the first evaluation of our concept of combining system monitoring with autonomously triggered user feedback. Public meetings for citizens of Santander interested in IoT were held. The UC team informed the citizens about the evolution of Santander as a smart city and gave an overview of the most relevant projects, including Wise-IoT. The Rich Parking application was presented, and citizens could volunteer for the pilot study, given the prerequisite that they have an Android phone and a car. There was no reward for participants who agreed to test the application. The citizens who volunteered did so because of intrinsic motivation to help the city's evolution as a smart city. The pilot study lasted three months, from the end of February to the end of May 2018.

During the test phase of the pilot study, we were running a logger that listened to the insights stream and wrote the data from the stream into log files, one file per day. Once per month, manually analysed the log files and shared the main findings with the developers of the Rich Parking application.

Since we wanted to know whether the insights stream, i.e. the combination of system monitoring and proactively, autonomously gathered user feedback, could provide added value to system evolution, we were not merely observing during the three-month test phase but applied action research principles. The Spanish developers evolved the Rich Parking software and maintained the IoT-based smart city system. The application and its recommender system received two minor updates during the test phase.

When the test phase ended, we analysed the insights stream data. One goal of the insights stream was to support system evolution by helping engineers discover potential problems, user needs, and new requirements. Therefore, the stream should help to provide answers to engineering questions, such as: how satisfied are the users with the system? What are the issues that generate user churn? Table 1 shows the full list of questions.

Table 1. Summarised answers to the engineering questions for system evolution.

Engineer	Questions Answered by Analysing the Insights Stream
Application Developer	**How satisfied are the users with the app and the information it offers?** Mediocre ratings for recommended routes and proposed parking spots.
	What are the issues that generate user churn? Parking spots got occupied before the user arrives. Sensors showed free spots, but there was not enough space to park the car. Construction work blocked streets and parking spots.
	What are the preferences (likes, dislikes) of users? Feedback in the insights stream shows what parking spots and routes received very good or bad ratings.
	What segments of the street map are incorrect? Street segments with construction work not correctly marked as blocked.
	What Point-of-Interest information is incorrect? Spaces between cars too small to park. Sensors between cars. Blocked spaces due to construction work.
Smart City System Engineer	**What is the users' trust rating in the IoT entities?** Mediocre ratings for the parking spots.
	What are the users' reasons for the trust ratings in the IoT entities? Parking spots occupied, or not enough space to park the car.
	How credible are context information offered by the IoT entities? The sensors worked fine but needed to see the size of the available spaces and see objects that are not placed directly on top of them.

5 Results

5.1 Collected Data

41 citizens had registered and took part in the pilot study. A total of 303 sessions were created with recommendations for a route and parking spot. In 68 out of the 303 cases, users have started the session monitoring after receiving a recommendation. In 26 out of the 68 monitoring sessions, users allowed the mobile app to send their GPS positions to the system, enabling the system's adherence monitoring functionality. In ten out of these sessions (38.5%), the users have adhered to the recommended route (i.e., less than 50% of the user coordinates sent during the session were farther away from the recommended route than eight meters).

In 16 out of the 26 adherence-monitoring-enabled sessions, users submitted a feedback form. Seven of these forms were of type 1 concerning route deviations, five of type 2 concerning a parking spot deviation, and four of type 3 concerning a fulfilled recommendation.

We could not track users over multiple sessions due to the session-based privacy mechanism. We counted individual sessions instead. The users allowed the mobile app to send their GPS data in 38.2% of all monitoring sessions (26 out of 68). We considered this to be a good amount because some of the 68 monitoring sessions may have been started for testing the app functionality and not to go somewhere.

From the 26 sessions with user GPS data, we received 16 submitted feedback forms. This figure means that 61.5% of users who were presented with a feedback form decided to fill it out and submit it. If we consider all the sessions in which the monitoring has started, the ratio is 68/16 = 23.5%. Again, some of the sessions may have been started for testing the app functionality. We consider the 23.5% feedback ratio to be high in comparison to other feedback approaches or uses of surveys to collect feedback. The reason for this result could be that we kept the forms small and simple and that they were presented to the users in situations and with content that was relevant to them.

5.2 Data Analysis

The average user satisfaction rating of the parking spots was 2.125 and of the recommended routes was 2.071 out of 5 stars. We analysed the automatically collected user feedbacks for reasons why the scores were not higher and coded the free-text answers. Table 2 shows the resulting categories and number of answers in each category.

Table 2. Categorised user feedback from the free-text answers.

User Feedback	Number of Answers
The parking spot was occupied	6
There was a more direct or faster route	5
The parking spot was too small or the sensor in a bad location	5
The route or parking spot was blocked by construction work	2
The app was too slow or stalled	2
Found a free parking spot before arriving at the proposed one	1

To put the above feedback into context, we visualised the monitoring data on a map and added the feedback according to the GPS data of the users. Fig. 3 shows the parking spots rated by the users, together with their feedback. It also shows one of the recommended routes and the corresponding route taken by the user.

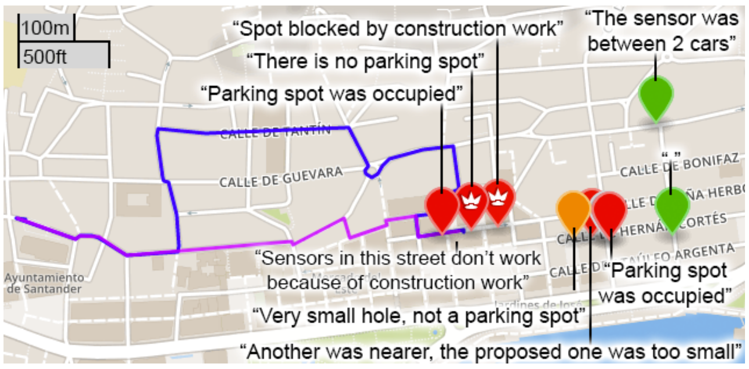

100m
500ft
"Spot blocked by construction work"
"There is no parking spot"
"Parking spot was occupied"
"The sensor was between 2 cars"
"" ""
"Sensors in this street don't work because of construction work"
"Parking spot was occupied"
"Very small hole, not a parking spot"
"Another was nearer, the proposed one was too small"
CALLE DE TANTÍN
CALLE DE GUEVARA
CALLE DE BONIFAZ
Ayuntamiento de Santander

Fig. 3. Parking spot feedback (red: negative, red with crown: discussed in the text, green: positive). Blue line: a route recommended to a user. Violet line: route taken by that user.

Despite the few user feedbacks received by the system, important issues could be identified with the analysis of the correlated monitoring and feedback data.

One feedback mentioned a blocked parking spot due to construction work and another a non-existent spot. When looking at the map, these two spots were close to each other (white crown markers). Also, a recommendation was leading to another spot nearby on the same street, and the user gave the feedback that the sensors on that street did not work because of construction work. The construction work in that street was an issue that was unveiled by combining user feedback and monitoring data.

The second group of red markers shows an accumulation of parking spots that were either occupied or too small. However, because of the few data points, we could not say whether this was a feature specific to that location, or whether this was a more general problem with the parking spots in the city. For example, one of the positively rated spots also received negative feedback (stating that the sensor was between two cars, which means that it was not possible to park there). But the green markers were both located in less crowded areas of the city, where the chance of finding a free spot was higher (whether it was the recommended one or another one close by).

Further, one user stated that no parking spot was available at the recommended location. However, the user's GPS data showed that the user never was in that location but went somewhere else instead. This observation is an example of how monitoring data can be used to verify the validity of user feedback. It seems that there was a different issue instead, e.g. the user may not have been able to read the map correctly.

5.3 Generated Insights

The analysis surfaced findings with a significant effect on the maintenance and evolution of the smart city system and the Rich Parking application.

Construction work. Increasing construction work blocked streets and parking spots during the test phase. The recommender used an external street routing framework that was not updated with the construction work information in a timely fashion. Therefore, the recommender sometimes proposed routes with blocked street segments. The effect of increasing construction work during the trial phase appeared to be larger than the effects of other factors that could have led to improved route recommendations over time. 38.5% route adherence seems to be quite good, given the construction work problem as well as possible cases where users might have decided not to follow the proposed route for other reasons. However, this insight must be taken with care due to the low number of sessions and the fact that the users were aware of participating in a test, which could have biased them to follow the proposed routes eagerly. Furthermore, construction work could also have had a negative effect on the parking spot ratings. If a user gave a bad rating to a parking spot because it was lying inside a construction zone and thus not reachable, there was still a chance that next time, the system would propose one of the parking spots that are close to the badly rated spot and that are still located within the (same) construction zone.

Parking sensors and fluctuation in parking spot availabilities. The system proposed parking spots to users that were unoccupied at the time when the user requested a recommendation. During high traffic, there was a good chance that another vehicle would park on the proposed spot before the user arrives. As a result, users may have experienced occupied spots and gave bad ratings. Vehicles were sometimes inaccurately placed on the parking sensors, in the worst case in the middle of two parking spots, and did not trigger the parking sensors. The insight implied that upgrading the hardware or updating the software is needed. An improvement to the software could be to let the recommender prioritise regions with large numbers of free parking spots or to introduce a reservation system. These issues pointed out by the insights stream could not be solved during the pilot phase. However, they provided developers with facts to think of how to improve the system.

Ratio between user ratings and available parking spots. While we received nine user ratings about parking spots, the city of Santander contained hundreds of spots. The coverage of the city's parking spaces is relatively low. Broader use of the Smart City-generated IoT data is needed to generate insights for the totality of the city.

6 Discussion

6.1 Revisiting the Research Question

Our research question was: *do the combination of system monitoring and autonomously triggered user feedback provide added value to system evolution?* The initial evaluation of the proposed approach shows that we can answer our research question positively. The insights generated by the system have exposed important issues to the Rich Parking application and Smart City system engineers. The successful exposition and detection of these issues have set the control loop for system evolution in motion, which allows developers to come up with improved solutions [26].

The insights generated by combing monitoring data and autonomously gathered user feedback helped to answer the initially posed engineering questions, as Table 1 briefly summarizes. The insights could inform system evolution with advice for new features and how to enhance existing features: better parking sensors that can sense cars or obstacles that are not placed exactly on top of the sensors, a solution that takes into account the amount of fluctuation in the available parking spots, and a better synchronization of the routing framework with the real-world situation, maybe by connecting the system to the city's database with information about construction work or otherwise blocked streets. Also, knowing the context, which the user feedback applies to, allows adjusting the recommender to prefer less centrally located spots, or regions in which the density of free parking spots is higher. This modification could increase the probability for the user to find a free spot. These findings show the valuable outcome of combining monitoring data and autonomously gathered user feedback, which is also aligned with the research by Oriol et al. [5].

The combined analysis of monitoring and feedback data also allowed to identify two invalid user feedbacks: users who gave feedback about parking spots they never visited. Without the monitoring data, it would not have been possible to distinguish between valid and invalid feedback. This result shows that our proposed approach may be used to reduce the risk of collecting irrelevant or even fake feedback [25].

These findings were possible even though we had to make some compromises because of data privacy and because few users participated in the pilot and generated just a small number of feedbacks. User privacy implied that we were not allowed to identify or track users over multiple sessions. We had to introduce a random ID for each session and could only perform evaluations on a per-session rather than a per-user basis. The compromises, however, underline the effectiveness of proactive, autonomous gathering of user feedback for generating significant insights that can be translated into system maintenance and evolution actions.

Our approach targets the collection of user feedback on interesting situations of system use by basing the feedback requests on monitoring the fulfilment of user goal. This technique reduces the disturbance of users that is due to the feedback requests, but still requires feedback. In the presented use case, such disturbance could lead to accidents and, in the worst case, liability of the vendor for such an accident. We had chosen to avoid dangerous disturbance and ask users for feedback when they were in a safe situation after the experience. This delayed request for feedback may have been reducing the ability of the users to remember important details of the experience and produced ambiguity for understanding the context the feedback applies to. The insights generated about the monitored applications and systems may still be considered relevant. Earlier research showed that the disturbances have negligible impacts on the satisfaction level of the users with systems [6].

6.2 Discussion of the Results

One could argue that the issues found are to some extent obvious and that the same conclusions could be reached by “thinking hard” about the application. In retrospective, such sense-making is relatively easy. Identifying and expressing such issues in advance is difficult [28] and one of the reasons why requirements engineering is non-trivial. Our evaluation has shown an example of field testing of a prototype application and has shown that proactive, autonomous gathering of user feedback may be effective.

More lightweight approaches, such as questionnaires, may be used as an alternative or complement to elicit requirements [29]. With our approach, real system may be observed in use in the real environment and by real users. This gives the advantage that elicited input such as a feedback may be connected to a specific context, such as a physical location, time, or situation, where a system needs modification. Alternative elicitation approaches, such as surveys, would be too short in time, disturb many users, and would not allow understanding the contexts to which the users’ feedbacks pertain.

In addition to being used during prototype development, our approach also has the advantage of being useful for a situation where a deployed system is evolved or maintained. With very little effort and cost, our approach allows continuously monitoring a system and warn support engineers for failures that result from the slow decay of the system that is being used often in situations where the context is changing. The availability of a development kit, outlined in section 3.2, limits the effort of setting up the monitoring and feedback mechanism.

In comparison to bespoke methods, our approach also has the benefit that physical presence of an analyst or engineer is not required. It thus offers support and scales well in situations where the system is being deployed and used over geographical space. That has been the case for the Smart City application, where it would have been impractical to put a person on the side of the users.

The presented full automation of data collection, analysis, and streaming of insights may offer benefits beyond system maintenance and evolution. For example, the approach may be used by a system to self-adapt. Integration of the insights stream into a recommender system would allow the system to use the user feedback, ratings, and context data from the stream to improve future recommendations by automatically enhancing or discounting recommendation options. If some entities contained in the recommendations (such as points of interest) receive many bad user ratings, the recommender system may avoid recommendations with these entities in the future.

6.3 Threats to Validity

Conclusion validity. We conducted a qualitative evaluation and did not focus on statistical significance, which can be seen as a threat. The number of logged sessions with GPS data was relatively small. However, the insights stream proved its usefulness by pointing out possibilities for system improvement which are valid for the involved group of users. The findings are in accordance with feedback given in the final survey conducted by the pilot partners.

Internal validity. Due to the participant selection method, the participants had an interest in IoT that is above average. The participants could have been motivated to send GPS data or feedback just because they knew that they were part of a study. There is the possibility that the participants were friendlier than the average user. However, looking at the feedback ratings, we saw that they were not hesitant to give one-star ratings when they encountered a problem with a recommendation.

Construct validity. The pilot has focused on the data collection and analysis steps of Chen's control loop. The short duration of the pilot period limited the ability to observe the decision-making and acting steps in the evaluation. A longer testing period would have allowed to include major software updates and measure their impact. Further, it needs to be noted that the feedback forms in the application focused on parking spot and route recommendations, and not on other aspects of the Smart City application and system. This focus could have held back participants from providing broader feedback.

Another concern is the construct of the interestingness of a situation. In the presented study, the interestingness was defined by asking developers for assumptions about what could go wrong, such as users not reaching the parking or being dissatisfied with it. This method depends much on human input, and it would be interesting to find methods to decide about interestingness in an automated manner. Using such triggers for the most "interesting situations" could allow reducing the need of disturbing the users, e.g. by reducing the number or frequency of feedback requests.

External validity. So far, we have evaluated our concept in only one scenario with participants were interested in the scenario. Our results show the applicability of our method in systems that have a physical dimension, i.e. a city. They also show results that can be obtained with users that have a positive attitude, leaving the impact of neutral or negative users open. Generalization should be the subject of further evaluations.

7 Conclusion

We have presented an approach that combines system monitoring with proactive, autonomous user feedback collection. The approach offers automated collection of data from a runtime system and its users and analysis of that data to offer insights that support engineers in decision-making for system maintenance and evolution. We let our implementation of the approach be integrated in a pilot of a Smart City prototype application. The initial evaluation has shown that the approach was valuable for system evolution: the results were helpful for the Smart City Santander partners to adapt and improve their application as well as the IoT sensors deployed in the city. The evidence from the use case shows that our concept provides a systematic approach for gathering user needs, potential issues, and new requirements. Such an approach can be especially helpful for distributed systems and the IoT where it is difficult to localize the reasons for potential issues and weaknesses of the system.

Acknowledgement: Part of this work has been supported by the European Union's Horizon 2020 Research and Innovation Programme within the project WISE-IoT under the EU grant agreement No. 723156 and the Swiss SERI grant agreement No. 16.0062.

References

1. Kittlaus, H.-B. and Fricker, S., *Software product management*. Springer, 2017.
2. Stade, M., Fotrousi, F., Seyff, N., and Albrecht, O., Feedback gathering from an industrial point of view, IEEE 25th International Requirements Engineering Conference, Lisbon, Portugal, 2017.
3. Maalej, W., Nyebi, M., Johann, T., and Ruhe, G., Toward data-driven requirements engineering, *IEEE Software,* 33(1):48-54, 2016.
4. Guzman, E. and Maalej, W., How do users like this feature? A fine grained sentiment analysis of app reviews, 22nd International Requirements Engineering Conference (RE 2014), Karlskrona, Sweden.
5. Oriol, M., Stade, M., Fotrousi, F., Nadal, S., Varga, J., Seyff, N., *et al.*, FAME: Supporting continuous requirements elicitation by combining user feedback and monitoring, 26nd International Requirements Engineering Conference (RE 2018), Lisbon, Portugal, 2018.
6. Fotrousi, F., Fricker, S., and Fiedler, M., The effect of requests for user feedback on quality of experience, *Software Quality Journal,* 26(2):385-415, 2018.
7. Chapin, N., Hale, J., Khan, K., Ramil, J., and Tan, W.-G., Types of software evolution and software maintenance, *Journal of Software Maintenance and Evolution: Research and Practice,* 13(1):3-30, 2001.
8. Fickas, S. and Feather, M., Requirements monitoring in dynamic environments, 2nd International Symposium on Requirements Engineering (RE'95), York, U.K., 1995.
9. Ali, R., Dalpiaz, F., Giorgini, P., and Silva Souza, V., Requirements evolution: From assumptions to reality, BMMDS/EMMSAD, London, UK, 2011.
10. Bosch, J., Building products as innovation experiment systems, International Conference on Software Business (ICSOB 2012), Cambridge, MA, USA, 2012.
11. Blank, S., Why the lean start-up changes everything, *Harvard Business Review,* 91(5):63-72, 2013.

12. Edison, H., Smørsgård, N., Wang, X., and Abrahamsson, P., Lean internal startups for software product innovation in large companies: Enablers and inhibitors, *Journal of Systems and Software,* 135:69-87, 2018.
13. Carreño, L. and Winbladh, K., Analysis of user comments: An approach for software requirements evolution, 35th International Conference on Software Engineering (ICSE 2013), San Francisco, CA, USA, 2013.
14. Wellsandt, S., Hribernik, K., and Thoben, K., Qualitative comparison of requirements elicitation techniques that are used to collect feedback information about product use, 24th CIRP Design Conference, Milano, Italy, 2014.
15. Leucker, M. and Schallhart, C., A brief account of runtime verification, *Journal of Logic and Algebraic Programming,* 78(5):293-303, 2009.
16. Knauss, E., Lübke, D., and Meyer, S., Feedback-driven requirements engineering: The heuristic requirements assistant, 31st International Conference on Software Engineering (ICSE 2009), Vancouver, British Columbia, Canada, 2009.
17. Seyff, N., Ollmann, G., and Bortenschlager, M., Appecho: A user-driven, in situ feedback approach for mobile platforms and applications, 1st International Conference on Mobile Software Engineering and Systems (MOBILESoft 2014), Hyderabad, India, 2014.
18. Fotrousi, F. and Fricker, S., Qoe probe: A requirement-monitoring tool, Requirements Engineering: Foundation for Software Quality (REFSQ 2016), Göteborg, Sweden, 2016.
19. Maalej, W., Happel, H., and Rashid, A., When users become collaborators: Towards continuous and context-aware user input, OOPSLA 2009, Orlando, FL, USA, 2009.
20. Maalej, W. and Nabil, H., Bug report, feature request, or simply praise? On automatically classifying app reviews, 23rd International Requirements Engineering Conference (RE 2015), Ottawa, Ontario, Canada, 2015.
21. Morales-Ramirez, I., Perini, A., and Guizzardi, R., An ontology of online user feedback in software engineering, *Applied Ontology,* 10(3-4):297-330, 2015.
22. Elling, S., Lentz, L., and de Jong, M., Users' abilities to review web site pages, *Journal of Business and Technical Communication,* 26(2):171-201, 2012.
23. Pagano, D. and Brügge, B., User involvement in software evolution practice: A case study, in *35th international conference on Software engineering (ICSE 2013)*, San Francisco, CA, USA, 2013, 953-962.
24. Fotrousi, F., Fricker, S. A., and Fiedler, M., Quality requirements elicitation based on inquiry of quality-impact relationships, 22nd IEEE International Conference on Requirements Engineering, Karlskrona, Sweden, 2014.
25. Dalpiaz, F., Social threats and the new challenges for requirements engineering, 1st International WOrkshop on Requirements Engineering for Social Computing (RESC 2011), Trento, Italy, 2011.
26. Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J.*, et al.*, Software engineering for self-adaptive systems: A research roadmap, in *Software engineering for self-adaptive systems*, B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., ed: Springer, 2009.
27. Qian, W., Peng, X., Wang, H., Mylopoulos, J., Zheng, J., and Zhao, W., MobiGoal: Flexible achievement of personal goals for mobile users, *IEEE Transactions on Services Computing,* 11(2):384-398, 2018.
28. Ericsson, K. A. and Simon, H. A., Verbal reports as data, *Psychological Review,* 87(3):215-251, 1980.
29. Zowghi, D. and Coulin, C., Requirements elicitation: A survey of techniques, approaches, and tools, in *Engineering and managing software requirements*, A. Aurum and C. Wohlin, Eds., ed Berlin, Germany: Springer, 2005.