Eeny, Meeny, Miny, Mo...

A Multiple Case Study on Selecting a Technique for User-Interaction Data Collecting

Sampo Suonsyrjä(✉)

Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
sampo.suonsyrja@tut.fi

Abstract. Today, software teams can deploy new software versions to users at an increasing speed – even continuously. Although this has enabled faster responding to changing customer needs than ever before, the speed of automated customer feedback gathering has not yet blossomed out at the same level. For these purposes, the automated collecting of quantitative data about how users interact with systems can provide software teams with an interesting alternative. When starting such a process, however, teams are faced immediately with difficult decision making: What kind of technique should be used for collecting user-interaction data? In this paper, we describe the reasons for choosing specific collecting techniques in three cases and refine a previously designed selection framework based on their data. The study is a part of on-going design science research and was conducted using case study methods. A few distinct criteria which practitioners valued the most arose from the results.

Keywords: Agile software development · User-interaction data · Multiple case study · Software data collecting

1 Introduction

In the last few years, the world has witnessed a tremendous progress in the ways software is developed with. On one hand, this has already benefited both customers and vendors by improving productivity, product quality, and customer satisfaction [1]. On the other hand, the acceleration of release velocity has been such a strong focus point, that the evolution of the means of understanding user wants and needs could not have kept up the pace. For example, Mäkinen et al. [2] describe that customer data analytics are still used sparingly. Similarly, research related to the techniques of automatic collecting of post-deployment data and its use to support decisions still seems to be in its infancy [3]. This feels partly unfortunate, because agile software development has always had the intention of faster responding to changing customer requirements – and to achieve this, both rapid releasing and rapid understanding of customers are needed.

Addressing this, one of the promising solutions is to track users in the user-interface level, then analyze that data to understand how they use the software,

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 52–67, 2017.
DOI: 10.1007/978-3-319-57633-6_4

and finally make decisions based on the analysis [4]. To start such a process, the first thing to do is to select a collecting technique that is suitable for the case. There are many restrictions to this, however, and these make the selecting a rather problematic task. Therefore, guidelines for evaluating and selecting a suitable collecting technique are needed. In our previous work [5], we have designed such a selection framework, which should serve as a guideline and help practitioners in these tasks. The objective of this study is to evaluate and refine that selection framework.

In this paper, we describe the reasons for choosing specific collecting techniques in three different case contexts and evaluate and refine the previously presented selection framework based on their data. The study is a part of ongoing design science research in which we have already designed the selection framework. This part uses the case study method to evaluate and refine the previous design and explore its contexts. Specifically, we address the research question:

– **What reasons software teams have for selecting a specific technique for user-interaction data collecting?**

To answer this overarching research question, we have derived two sub-questions. Firstly, the process of choosing a collecting technology will be explained. Secondly, we try to find out if some of the criteria we presented in our previous work are more significant than others or if there are completely other and more relevant reasons for choosing the technologies. The sub-questions for the study are declared as follows:

1. **How were the collecting techniques selected in each case?**
2. **What kind of criteria for choosing a certain technique were the most significant in each case?**

The rest of the paper is structured as follows. In Sect. 2, we present the background of the study, namely the selection framework which consists of selection criteria and a process. In Sect. 3, we explain how and why we used case study methods and describe the cases involved. In Sect. 4, we describe the process and criteria for choosing a specific technique for user-interaction data collecting in each case. In Sect. 5, we discuss those results to evaluate and refine the selection framework and in Sect. 6 we present the final conclusions of the study.

2 Background

To the best of our knowledge, related work for selecting techniques for user-interaction data is very limited. For example, a recently published systematic mapping study by Rodriguez et al. [6] identified the analysis of why certain technologies for monitoring post-deployment user behavior are selected over other similar existing solutions as a concrete opportunity for future work. However as a background for this study, we revisit the basics of the previously designed selection framework for user-interaction data collecting techniques.

The selection framework forms the basis for this study, as our goal is to evaluate the framework and refine it where necessary. It consists of a set of selection criteria and a process for the selecting. In addition, we introduce different techniques for user-interaction data collecting. These techniques and their evaluations are presented in a more detailed manner in [5]. They are mentioned nonetheless here for an overlook to the different alternatives that software teams have when they start collecting user-interaction data and for demonstrating the criteria part of the selection framework.

2.1 Selection Framework for a Collecting Technique

Criteria. The selection framework guides software teams to evaluate user-interaction data collecting techniques in terms of the technique's *timeliness, targets, effort level, overhead, sources, configurability, security, and reuse.* In the following list, each criterion is described by demonstrative questions which could be asked as a team evaluates collecting techniques.

- *Timeliness.* When can the data be available? Does it have a support for real-time?
- *Targets.* Who should benefit from the data? What is the intended use? Does it support many targets? Does it produce different types of data?
- *Effort level.* What kind of a work effort is needed from the developers to implement the technique?
- *Overhead.* How does it affect performance, e.g. system response time to user-interactions?
- *Sources.* Does it support many source platforms?
- *Configurability.* Can the collecting be switched on and off easily? Can it change between different types of data to collect?
- *Security.* Can the organization who developed the collecting technology be trusted with the collected data? Is the data automatically stored by the same organization?
- *Reuse.* Is the collecting always a one-time solution or can it be reused easily?

Process. The first thing to do when selecting a technique for user-interaction data collecting, is to rapidly **explore the case** to get a grasp of the most critical technical limitations. These include things such as the size of the code base, availability of automated tools and AOP libraries for the target application's language and platform, and access to the UI libraries and execution environments.

If any critical limitations are faced, the next step is to **reject the unsuitable techniques** accordingly. For example, if there are many security issues related to the data being collected or if data needs to be sent in real-time, collecting techniques using 3rd party tools might have critical limitations that cannot be avoided resulting in the rejection of the technique.

The following step is to **prioritize the evaluation criteria**. In addition to the explored case information, one should find out the goals different stakeholders

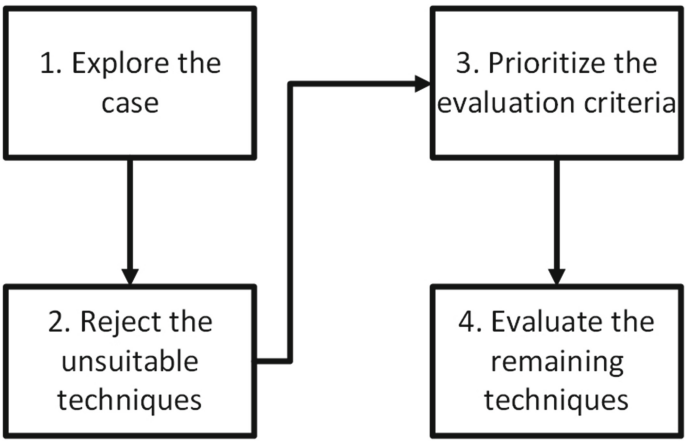

Fig. 1. Selection framework for user-interaction data collecting techniques.

have for the usage data collecting as these can have a major impact on the approach selection. If the goals are clearly stated, and the aim is e.g. to simply find out which of two buttons is used the most, manual instrumentation can work sufficiently. However, if the goal is stated anything like "to get an overall view of how the system is used" or if the goal is not stated at all, the more automated and more configurable approaches most likely become more appealing. Therefore, one of the most crucial things to find out in this step is to understand what different stakeholders want to accomplish with the collected data.

After this, the final step is to **evaluate the remaining approaches**. The plus and minus signs used in Table 1 work as guidelines in this, but their emphasis obviously varies on a case to case basis. To summarize, the selection framework is illustrated in Fig. 1.

2.2 Techniques for User-Interaction Data Collecting

Firstly, in **manual instrumentation** (Manual) developer adds extra statements to the relevant locations of the software. On one hand, this highlights the flexibility of the technique but on the other, adoption to new targets and sources would require significant rework making reuse practically impossible.

Secondly, there are multiple **tools for automated instrumentation** (Tools) of the code, e.g. GEMS [7], for various data logging, quality assurance and performance monitoring purposes. This technique frees the programmers from the manual work and reduces the probability for errors lowering the effort significantly.

Thirdly in between of the above techniques, **aspect-oriented programming approach** (AOP) is something of a mixture from the two. The research presented e.g. in [8,9] use aspect-oriented programming as a tool for code instrumentation. Aspect-based instrumentation allows the instrumentation to be

Table 1. Summary of the technique evaluations.

Criteria	Techniques				
	Manual	Tools	AOP	UI Lib.	E.E.
Timeliness	+	−	+	+	−
Targets	+	−	+	−	−
Effort	−	+	+	+	+
Overhead	+	−	+	−	−
Sources	+	−	−	−	−
Configurability	+	−	+	+	−
Security	+	−	+	+	−
Reuse	−	+	−	−	+

+ = Supports selecting
− = Technique has limitations

system and application specific, which focuses the collecting better on the relevant targets.

Fourthly, **an alternative implementation of a user-interface library** (UI Lib.) can be set to automatically collect user-interaction data. Because user-interaction is usually implemented with standard UI libraries, their components can be altered so that they include the collection of user-interaction data within them. Finally, the data collection can also be integrated into the environment without modifications to the original application. For languages like Java and JavaScript the virtual machine is an **execution environment** (E.E.) where method and function calls can be monitored by instrumenting critical places.

We have summarized the evaluations of different collecting techniques for the basis of the selection framework, i.e. Table 1, giving each technique either a plus if it has a positive impact or if it does not have restrictions in terms of the criterion. A technique is marked with a minus sign if it limits the selection or the use of a data collecting implementation according to a criterion.

3 Research Approach

The study was conducted using case study methodology. It allowed us to explore and describe the case specific situations and their circumstances related to the selection framework from deeper and more insightful viewpoints than if a research method with set variables had been used. Case study investigates contemporary phenomena in their real-life context [10], and this suited the purposes of the study well. The study is a part of on-going research effort, where we design, evaluate, and diffuse the selection framework by design science guidelines presented in [11] and with the process presented in [12]. The design science method of the underlying research effort affected this study as well especially in how actively the researchers had to take part in the cases. This participation was

obviously required because the automated collecting of user-interaction data and its use for software development was still an unknown area for each of the case organizations. Moreover, the researchers had a substantial expertise considering the designed selection framework.

3.1 Explanatory Case Study

The selection framework, as presented in [5], includes predetermined criteria for evaluating the collecting technologies. These criteria could have been used straightforwardly as variables of a study with more experimental setting. However, the criteria have been derived from a literature survey and from only one case study. Therefore, we acknowledge that there can be other criteria that affect the selection as well, and perhaps with a greater impact. To allow the inclusion of these other possible factors into the selection framework, we have chosen to use specifically *multiple* case study method and gather data from three different cases.

This study uses *explanatory* case study methodology because its aim is at finding the reasons why software teams choose a specific collecting technology. The results of this explanatory case study are used for evaluating and refining the designed selection framework where necessary. Runeson & Höst [13] have categorized case studies by their purposes into exploratory, descriptive, explanatory, and improving. Since explanatory case studies are "...seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship", their aims are well-suited for the study.

Case Selection. Given the purpose of the studied selection framework, its potential users are mainly software teams that are only beginning to collect user-interaction data. This limited the potential cases for this study to software teams that had not yet selected a technique for user-interaction data collecting but were still willing to try such collecting out. Clearly, the selected cases had to be open enough that publishing the results reliably was possible and also accessible in the first place for the first author to do the research with them.

Similarly, the number of the cases selected for the study was affected by the fact that the first author had to spend considerable effort in each case. As suitable software teams for this study had not tried out user-interaction data collecting or explored its techniques, the first author had to have access to a potential software team to tell about the possibilities of such data collecting and initiate these tasks. All of these limited the number of selectable cases to few, and finally three software teams were selected for the study.

Data Collection. The data was gathered from February to December 2016. Main parts of the data consist of meeting notes written down by the first author of this paper. Workshop type of meetings were held in each of the cases. Since collecting user-interaction data was a novelty for each participating software team, simple interviewing would not have worked. Rather, the meetings were organized as workshops where the first author motivated the software team to

try out user-interaction data collecting and described the different possible techniques for doing so. In addition to the data gathered in meetings, the first author had designated work desks in the same rooms where the software teams were working in cases A and B. Therefore, data was also gathered by observation and by participating in informal meetings. However, these data were only used for verifying some of the previously collected meeting note data, such as how many standup meetings a team have in a week. Although these observational data were not collected in a formal fashion, for the first author it improves the reliability of the results in terms of data triangulation.

Validity and Reliability Considerations. Although this study tries to investigate what kind of things have an effect on the decisions of software teams, the aim is not to find definitive proofs or certain amounts of statistical significance in these relations – rather to broaden the scope of possible causes. Therefore, the internal validity needs especially careful considering. Firstly, selections in earlier cases can have had effects on later ones. This was obviously not intentional but still surely possible because the same researcher explained the different options for the teams in each case. However, the author of this paper separated himself from the decision making in each of the cases and the decisions were made only by the software teams.

Secondly, the criteria presented with the selection framework can have guided the author of this paper to identify only those as the reasons for selection. Consequently, there can have been reasons that have not been mentioned aloud in the meetings but which still have had an effect on the decision. For example, a technology might have been seen as an unsuitable option in such an indisputable manner that the software team has not even mentioned it. This risk was mitigated in cases A and B by not only gathering data from meetings, but also by observing the working of the teams in the their offices and participating in their informal meetings.

The results of this study will not be generalizable for any software team. However, they provide a detailed look on the reasons these three software teams had for choosing a user-interaction data collecting technique. The three case organizations are different from each other in many ways, and therefore the results can give interesting insights to a wide audience. Although only one researcher gathered the data in each case, the meeting notes were shown to and accepted by team members in each case.

3.2 Case Organizations

Case A. Organization A is a large international telecommunications company. The software team that was involved in this case consisted of around eight members. The border of one team in this organization is quite flexible as employees work for many products. The team members had titles of software architect, UX designer, software developer, and line manager. Their products consist primarily of software in the field of network management, and these range from Java software

to web based systems. The software development method used in their team has some properties from agile development methods such as Scrum. They, for example, have bi-daily standup meetings and they use Kanban boards to organize their work. New versions of their product are released usually a few times a year.

Case B. Organization B is a privately held software company in Finland. At the time of the study, they had around 300 employees and offices in three major cities in Finland and they primarily develop software in projects for their customers as ordered. The software team involved in this case, however, develops their own software-as-a-service solution. As in case A, the software team in case B also uses things such as daily standup meetings, Kanban boards and retrospective sessions familiar from some of the agile development practices. On the contrary however, they are releasing new versions of their product to the end-users far more often – usually biweekly. Their software team consists of seven members with titles such as product owner, UX specialist, software architect, and software developer.

Case C. Organization C is a research and education center of around 10000 students and 2000 employees. The case C software team is part of a research group who have specialized in embedded systems design. They have developed Kactus2, which is an "open source IP-XACT-based tool for ASIC, FPGA and embedded systems design"[1]. The software has created traction from users world wide. It has been downloaded around 5500 times during the last year requests coming mainly from the USA and from middle Europe. The development team consists of four employees with the titles of software developer, software architect and business architect. The developed tool itself is an installable software system and installer packages for Windows and Linux tar-packages of its new versions are released three to four times a year.

4 Results

The results of the study are twofold. Firstly, we describe the processes with which the techniques were selected in each case. Secondly, we dive into the reasons the software teams had for their selection.

4.1 The Processes of Choosing a Collecting Technology

Case A. In February 2016, members of the software team of Case A explained to the researcher that they had an overall interest in trying out the use of user-interaction data for the further development of their software products. The researcher had presented the different technological approaches for collecting such data in a previous informal meeting. These were the same approaches as described in [5]. Two of the software team's products had been then analyzed by

[1] http://funbase.cs.tut.fi/.

the Organization A in terms of the suitability of the products in experimenting with user-interaction data collecting. The first of the two was Tool X written in Java, and the second one a JavaScript based Web-system Y. The team decided to carry on the collecting efforts with the System Y.

After this decision, the team had a meeting with the researcher to give a short presentation about the code base of the System Y and its software architecture. The meeting was arranged as a workshop to find out what kind of user-interaction data the team wanted to have collected. In addition, the team described what is important for the collecting technology and its implementation.

From this point on, the job of the researcher in the eyes of the software team was to develop a demonstrative collecting tool for their product. The researcher then used the criteria from the selection framework and was left with only one suitable technology approach – developing a new tool for **monitoring the execution environment**. After developing a prototype of such a collecting tool, the researcher presented it in a demo show for the team in March and got a thumbs up from the team to go on with experimenting with the actual System Y. A testing day with eight users from within the Organization A was held in December 2016 to try out an improved version of the collecting tool implemented in a lab version of the System Y. The developed collecting tool is available in GitHub[2].

Case B. In case B, a similar workshop meeting as in Case A was held by the researcher with the software team in March 2016. The team explained the method they use for developing their software and what kind of a software the product is architecturally. It turned out, however, that this team had more experiences with collecting use related data even at that point. For example, they had tried out Google Analytics with some default settings for their product already. After explaining that the data was mainly collected for debugging, two of the team members and the researcher worked out also new targets in their software development process which could be improved with user-interaction data. These ideas ranged from prioritizing their product backlog to improvements in the user interface of the product.

The team was well motivated to try out user-interaction data collecting. However, as its return on investment was still unclear the first few tasks for data collecting were agreed upon to be completed with as little work effort and changes to the software architecture as possible. Therefore, three very specifically described places in the UI of the product were selected to be improved with the help of user-interaction data collecting. As the team had already tried out Google Analytics on the same product, it was a straightforward choice for the storing and analyzing the data of the tasks at hand as well.

At that point, the researcher described the same technological approaches to the team as in Case A. Also similar to Case A, the selecting of the collecting technology was an obvious pick since the three tasks were specified so explicitly. The team members and the researcher made an unanimous decision to use **manual implementation** for instrumenting the required places of the source code.

[2] https://github.com/ssuonsyrja/Usage-Data-Collector.

The researcher was then given rights to change the source code. After applying the collecting code to six places in it, the version was sent to end-users for a two week collecting period in April 2016.

Case C. In case C, an initial meeting was held with two members of the software team and the researcher in September 2016. Similar to the previous cases, the team members described the environment for which they develop software and the architecture of their product. The meeting then continued as a workshop, where each participant tried to figure out ways for how user-interaction data collecting could be used for their software development. Such targets were plenty, and no specific tasks were selected at that point. The researcher then explained the same technological approaches for user-interaction data collecting to the team members. The option of monitoring execution environment was rejected at this point, but the rest still remained possible for selecting.

The evaluation criteria from the selection framework were then used for the analysis of the product and its environment. Since the aspect-oriented approach raised the most interest among the software team, it was decided that the availability of AOP libraries and their suitability to the product were to be examined. An alternative implementation of a UI library was considered as a second choice, but the rest of the alternatives were rejected at this point. During the fall of 2016, the **aspect-oriented approach** was implemented technically successfully to the product. The first data collecting period is planned to be held during the spring of 2017 with a student group as experimental end-users.

4.2 Reasons for Choosing a Collecting Technique

Case A. The first decision made by the Organization A was that they selected to try out user-interaction data collecting with System Y. This decision was based on **the sources and the reuse possibilities** of the collecting effort, because the motivation was to specifically try out this kind of data collecting as a technical concept rather than immediately produce actionable insights from exact places of a product. Had the collecting effort been carried out with the Tool X, the reuse would have been practically impossible since its environment was not as common as with the System Y.

Although the overall motivation was to test user-interaction data collecting conceptually, the team wanted to focus the requirements of the data collecting after the selection of the specific **source**, i.e. product. Finding a technology that could be easily reused with as little implementation **effort** as possible became a goal. This made the option of manual instrumentation heavily unfavorable. The team also emphasized how **the security and configurability** were important for the collecting technology. For example, the environment of their product was such that the collecting should be easy to be left out of the whole product when necessary. Consequently, the unobtrusiveness of the technology was highly valued.

Although the need for low configuring effort increased the attractiveness of using an automated tool for instrumentation, the security concerns were so heavy

that the use of a tool developed outside the organization was not recommended. Therefore, the option of finding and using 3rd party tools was quickly rejected. In addition, the availability and effects of AOP libraries to things such as the **overhead** were unknown in the environment of System Y. Possibly the most significant of all, there was no motivation to make as big **a change to the software architecture** as needed by the aspect-oriented approach. The same reason applied for rejecting the option of an alternative UI library, because having different versions of the libraries was not acceptable for the delivery pipeline.

Case B. Similar to case A, the motivation for the team of case B in user-interaction data collecting was to try it out as a concept. On the contrary however, this resulted in this case in a faster and a narrower scoped experiment. In other words, the **targets** and the **source** of their data collecting were very clearly defined in the first place. At the same time, this resulted in the lack of significance of the implementation **effort** because it would be so low even with the manual approach. Similarly, **reuse** was not considered as a significant reason, since there were no guarantees that the data collecting mechanism would be ever reused. All this resulted in a very straightforward choice of the manual approach. It was by far the easiest approach to implement on a small scale and it allowed the team to try out if user-interaction data collecting in a fast and low-effort way.

Case C. Being a new thing for the case C software team, the user-interaction data collecting was again designed as a demonstrative experiment similar to the case A. Likewise, the interests of the team in this case were technical in the sense that they firstly wanted to find out a suitable technique for user-interaction data collecting. In the best case scenario, this technique could be then used with their actual product and actual end-users after the initial experiment. Because there was no simple access to experiment the collecting with real users, in the manner of case B, and the **security** requirements were weighted a lot heavier, the technical design of the collecting was the primary focus. Although the possible user-interaction data types and collection places, i.e. **sources**, were plenty, they were to be considered only secondly after validating the technical setup for the collecting.

This affected the evaluation of the collecting techniques in terms of prioritizing the criteria from the selection framework. Not limiting the **sources and targets** became important, because the collecting technique would not be selected and designed for just a one time try out. Although not mentioned out loud by the team, this could hint towards them valuing the **reuse** possibilities. All of these resulted in the attractiveness of the techniques enabling lower work **effort** spend on each distinct collecting place. Further on, the whole collecting was required to be able to be switched off as easily as possible. In other words, the **configurability** of the collecting technique was valued high.

5 Discussion

In each case, the process of choosing a collecting technology for user-interaction data was more or less the same. Members of the software team and the researcher had a meeting, where the researcher described the different technologies overall. After finding out what was the underlying goal for the team in the user-interaction data collecting, the most important criteria for the selecting became quite clear for both the researcher and the team members.

Comparing those criteria with the ones in the selecting framework, it is safe to say that most of the evaluation criteria from the selection framework were used without the researcher pushing the team towards those specific points. However, **timeliness** was never mentioned by the teams, which could signal either its insignificance or that its need is self-evident. On the contrary, **overhead** rose up in each case as a conversation topic but similar to the timeliness it did not seem to have any effect on the selecting in any case.

For both of these, it is worth mentioning that none of the techniques had a known disadvantage nor a limitation in terms of these criteria (timeliness and overhead) that would have been significant enough to get the whole technique rejected. However, in the original selection framework they were marked with minus signs for the monitoring execution environment technique. Therefore, the summary table with the evaluation criteria from the original selection framework, i.e. Table 1, requires some refining.

Firstly, the evaluations should consist of a wider scale than a plain plus or a minus sign. In these cases, some of the criteria affected the selection clearly a lot more than others. For example, the timeliness and overhead criteria did not seem to have an effect on the selection but on the other hand, the effort level of the manual technique had it rejected. Therefore, we propose an additional exclamation mark to the evaluations in case the criterion is a possible ground for a rejection. We have gone through the rest of the summary evaluations and added an exclamation mark where necessary based on the cases.

Secondly, some of the evaluations are not clearly pluses nor minuses. Therefore, we have added an option of $+/-$ marking for the evaluation, if the technique does not definitely support nor limit the selection in terms of the specific criterion. Adding this option has had effects especially on the evaluations of the techniques that are heavily intertwined with specific tools. For example, the minus signs in the execution environment column of timeliness and overhead rows can be then replaced with this option. We have reviewed the evaluations and changed the original signs into $+/-$ markings where necessary.

Thirdly, the *effort* criterion should be divided into two and renamed to *scalability*. The intention of the criterion is to depict the work effort that is required from the software developers to implement collecting snippets to the different places of the source code. Finally, however, there was a clear need for an evaluation criterion of how great an effort is needed from the software developers to change the software architecture and/or environment of the moment to support the collecting technique. This criterion could be named as the *change* that is

Table 2. Refined summary of the technique evaluations.

Criteria	Techniques				
	Manual	Tools	AOP	UI Lib.	E.E.
Timeliness	+	+/−	+	+	+/−
Targets	+	−	+!	−	+/−
Scalability	−!	+	+!	+	+!
Overhead	+	−	+/−	−	−
Sources	+	−	−	−	−
Configurability	+	−	+	+	+/−!
Security	+	+/−!	+/−	+	+/−
Reuse	−	+	−	−	+!
Change	+!	+	−!	−!	+

+ = Supports selecting
− = Technique has limitations
+/− = No clear support nor limitations
! = A possible ground the rejection

required. With these refinements to the criteria and evaluations, the summary table of the evaluations is as listed in Table 2.

In addition to the changes in the evaluations, the original selection framework requires some refinements based on the cases as well. First of all, in these cases the underlying goal of the whole collecting effort was the most important driver in the selection process. In cases A and C the delivery pipelines did not allow fast and flexible releases of new software versions with user-interaction data collecting capabilities, and so the software teams decided to develop their environment so that the collecting would be possible in the future. This became their real target, where as the team in case B did not have to develop their environment. On the contrary, they had the luxury of aiming straightforwardly at just testing out the collecting and the resulting user-interaction data with a minimum effort.

Therefore, the first step of the selection framework, *exploring the case*, should be clarified and replaced by a step of *defining a main goal* for the collecting effort. Based on these cases, it would be easy to then *remove the irrelevant evaluation criteria* after defining such a goal. For example, in case B the *scalability* of the collecting technique was seen unnecessary after the collecting was designed to be implemented as a one time solution.

Exploring the case still included important things that should be part of the selecting framework. Thus, the next thing of the process should be to *find out the critical limitations*. The rest of the original selection framework worked out as it was in these cases, and so no other changes were required to the final refined version of the selection framework. This framework is illustrated in Fig. 2.

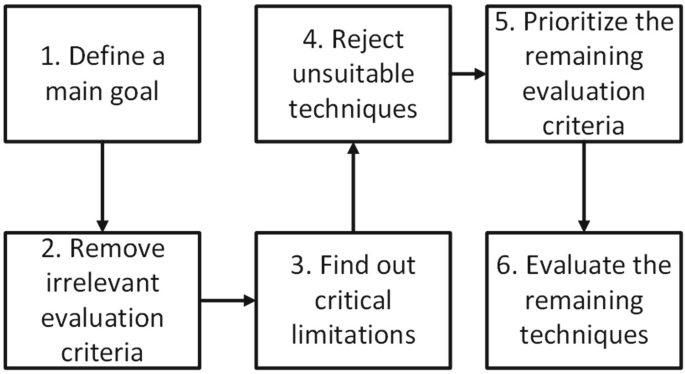

Fig. 2. Refined selection framework for user-interaction collecting techniques.

6 Conclusions

In this paper, we studied three cases where software teams selected techniques for user-interaction data collecting. More specifically, we examined the reasons the software teams had for the selection. To complement this, we evaluated our previously designed selection framework and refined it based on the data gathered from the cases.

In these cases, two of the most valued criteria for the selection were the scalability of the technique and the lack of changes required to the software architecture and deployment pipeline of the moment. Additionally, teams appreciated the reuse, security, and configurability of the techniques as well as the support for a wide range of monitoring targets. On the other hand, the rest of the criteria presented with the original selection framework, i.e. timeliness, overhead, and support for different source applications, did not seem to have a significant effect on the selections.

The original evaluations of the different user-interaction data collecting techniques were refined to include markings for the different levels of significance. In addition, the original selection framework was fixed to better support these more detailed evaluations. With these changes, we think the selection framework and its complementary technique evaluations can help practitioners greatly to the beginning of their journey of user-interaction data collecting.

Acknowledgments. The authors wish to thank DIMECC's Need4Speed program (http://www.n4s.fi/) funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes/) for its support for this research.

References

1. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. IEEE Softw. **32**(2), 64–72 (2015)

2. Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A.L., Laukkanen, E., Pagels, M., Männistö, T.: Improving the delivery cycle: a multiple-case study of the toolchains in finnish software intensive enterprises. Inf. Softw. Technol. **80**, 175–194 (2016)
3. Fabijan, A., Olsson, H.H., Bosch, J.: Customer feedback and data collection techniques in software R&D: a literature review. In: Fernandes, J., Machado, R., Wnuk, K. (eds.) ICSOB 2015. LNBIP, vol. 210, pp. 139–153. Springer, Cham (2015). doi:10.1007/978-3-319-19593-3_12
4. Suonsyrjä, S., Mikkonen, T.: Designing an unobtrusive analytics framework for monitoring Java applications. In: Kobyliński, A., Czarnacka-Chrobot, B., Świerczek, J. (eds.) IWSM/Mensura -2015. LNBIP, vol. 230, pp. 160–175. Springer, Cham (2015). doi:10.1007/978-3-319-24285-9_11
5. Suonsyrjä, S., Systä, K., Mikkonen, T., Terho, H.: Collecting usage data for software development: selection framework for technological approaches. In: Proceedings of The Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE 2016) (2016)
6. Rodriguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M.: Continuous deployment of software intensive products and services: a systematic mapping study. J. Syst. Softw. **123**, 263–291 (2017)
7. Chittimalli, P.K., Shah, V.: GEMS: a generic model based source code instrumentation framework. In: Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation, pp. 909–914. IEEE Computer Society (2012)
8. Chen, W., Wassyng, A., Maibaum, T.: Combining static and dynamic impact analysis for large-scale enterprise systems. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 224–238. Springer, Cham (2014). doi:10.1007/978-3-319-13835-0_16
9. Chawla, A., Orso, A.: A generic instrumentation framework for collecting dynamic information. SIGSOFT Softw. Eng. Notes **29**(5), 1–4 (2004)
10. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2013)
11. Von Alan, R.H., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
12. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manage. Inf. Syst. **24**(3), 45–77 (2007)
13. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. **14**(2), 131 (2008)