Understanding challenging situations in agile quality requirements engineering and their solution strategies: insights from a case study

Wasim Alsaqaf
University of Twente
Enschede, The Netherlands
w.h.a.alsaqaf@utwente.nl

Maya Daneva
University of Twente
Enschede, The Netherlands
m.daneva@utwente.nl

Roel Wieringa
University of Twente
Enschede, The Netherlands
r.j.wieringa@utwente.nl

***Abstract*— In the last few years, agile development methods are getting increasingly popular in large-scale distributed contexts. Despite this popularity, empirical studies have reported several challenges that large-scale distributed agile projects face regarding the implementation of quality requirements. However, there is little known about the mechanisms behind those challenges and the practices currently used by agile practitioners to adequately assure the implementation of quality requirements in distributed context. To look deeper into this, we performed a qualitative multi-case study in six different organizations in the Netherlands. Our multi-case study included seventeen semi-structured open-ended in-depth interviews with agile practitioners of different background and expertise. The analysis of the collected data resulted in identifying eleven mechanisms that could be associated with the previously published list of challenges. Moreover, the analysis uncovered nine practices used by agile practitioners as solutions to the challenges, in order to ensure the implementation of quality requirements. Last, we have mapped the identified mechanisms and practices to the previously identified challenges to get insight into the possible cause and mitigation of those challenges.**

***Index Terms*— Empirical research method, Quality requirements, Agile, Requirements engineering, Interviews, Case study.**

I. Introduction

The need for organizations to keep up with the rapidly changing environment and preserve their competitive position forces them to move from a heavyweight traditional approach of software delivery to a more flexible lightweight agile approach. Despite the original context that Agile Development Methods (ADMs) were sketched for – small and single co-located teams – more and more organizations scale up their ADM to fit their globalization and distribution context [1]. Scaling up ADMs does not only mean stretching agile practices to generate the intended benefits in large-scale distributed settings but also deals with aggravated challenges reported in their original context. One of these challenges concerns the engineering of non-functional requirements – or, as we call them here, quality requirements, in agile projects. Several studies have reported the neglect of quality requirements (QRs) in ADMs [2][3][4][5]. In our most recent paper [6] we have reported results of an exploratory multi-case study which indicated 13 QRs challenges that distributed agile teams face in agile large-scale development (ALSD). However, to be able to correctly understand the challenges and critically evaluate the possible solutions to them, we need to look at the underlying mechanisms behind those challenges and how the agile distributed teams currently cope with them. Therefore, we extended our qualitative analysis activities of the data collected in our previously presented exploratory study [6] in order to answer two research questions (RQ): *(RQ1) What are the mechanisms behind the reported challenges? (RQ2) What are the practices that agile teams currently use to mitigate the impact of the reported challenges?* The answers to these RQs form the contribution of this paper. By answering RQ1, we will get insights into the processes by which the reported challenges took place. These insights are useful not only to understand the environments where the challenges could take place but also to predict future challenges that agile teams could face when particular process (e.g. mechanism) is in place. The answer to RQ2 will help us to grasp the current practices used in distributed agile projects to engineer the QRs. By mapping those practices to the identified challenges we will get insight in practices that distributed agile teams could use to mitigate the reported challenges.

The remainder of this paper is structured as follows: Sect. II presents related work and Sect. III – our research process [6]. Sect. IV is about our results. Sect. V discusses them. Sect. VI treats limitations. Sect. VII concludes and Sect. VIII points to future work.

II. Related work

Several evidence-based studies [3][7][8][9][10][11] have reported requirements-related challenges and practices in large-scaled agile projects. However, not all of the aforementioned studies described the extent of "distributedness" of their studied project. The case study of Käpyaho et al. [3] involved approximately 30 agile practitioners and investigated the effectiveness of prototyping in solving reported agile requirements engineering (RE) challenges. The study reported that while prototyping can help with solving some agile RE challenges such as lack of

2332-6441/18/$31.00 ©2018 IEEE
DOI 10.1109/RE.2018.00035

documentation, motivation for RE work and poor quality communication, it does not help with quality requirements challenges without complementary practices. Dikert et al. [7] performed a systematic literature review (SLR) on industrial large-scale agile transformation. The median size of the organizations discussed in the review was 300 people. The study reported QR and global distribution as agile large-scale challenges among others. QR and team distributions were also reported as challenges in agile projects in another recent SLR [8]. Furthermore, Kasauli et al. [9] studied RE challenges in large-scale agile projects, in each of which approximately 30 sub-teams were involved. These authors concluded that companies are facing challenges when trying to address QR, such as safety and security. Petersen and Wohlin [10] investigated the effect of moving from plan-driven approach to incremental software development with agile practices. The authors described their case-study as large. Their study identified requirements quality assurance practices and requirements prioritization as challenges remaining for agile after migration to incremental software development. Finally, Rolland [11], an agile practitioner, reported his own experiences in large-scale agile projects regarding the challenges of verifying the importance of the identified requirements and getting the big picture of those requirements and their interdependencies. The investigated projects involved 13-14 teams and more than 200 participants.

III. Our Empirical research process

ADMs as well as RE depend in their application on human interactions and interpretations. Therefore, in our view the only way to understand how ADMs treats QRs is to explore the subject in real life settings [6]. To this end, we carried out a qualitative exploratory multi-case study that followed by the methodological guidelines of R. Yin [12]. We make the note that our research design was first presented in our 2018 publication [6]. Here, we further elaborate on those research-methodology-related aspects of our research process that help understand the results reported in this paper.

Our multi-case study [6] used semi-structured open-ended in-depth interviews designed according to the guidelines of Boyce and Neale [13]: First, we made a plan describing (1) the kind of information we intended to collect, (2) the kind of practitioners who could provide us with the sought-after information and (3) the kind of project settings that would be an appropriate candidate for inclusion in the multi-case study.

The unit of analysis in our empirical research process is the project in ALSD context. As we wanted to gain a solid understanding of the challenges of engineering QRs in ALSD projects from different perspectives, we decided to include practitioners with various backgrounds (e.g. different expertise and roles, e.g. architects, testers, different years of experience, different application domains). This choice for a multi perspective research (MPR) [14] design is motivated by our desire to get as broad understanding of the QRs challenges as possible. As per research methodologists (e.g. [12][13]), MPR allows qualitative researchers to include accounts of related individuals, e.g. developers, scrum masters, architects, product owners, in order to gain a more nuanced and more comprehensive understanding of the functioning of interactional systems or groups (like an ALSD project organization) or professional circumstances of individuals from different perspectives (like an agile software architect and a product owner in ALSD). As Eisikovits and Koren [14] suggest, covering possibly ccontrasting views on a phenomenon of research interest allows for insights regarding the underlying mechanisms characterizing the phenomenon, and can, in turn, stimulate theories that have more explanatory power than the sum of theories based on individual perspectives.

Second, we developed an interview protocol with instructions to be followed by each interview. The interview questions were developed by the first author based on the information we planned to collect and validated by the senior researchers (the other two authors). The interview questions were improved and finalized based on the feedback received from the senior researchers. Moreover, a pilot interview with an agile practitioner was done in order to check the applicability of the questions in real-life context. No changes were made to the interview questions after this stage. We make the note that we did not include the pilot interview in the case study because the respective project setting did not meet the requirement of project distributedness. Interested readers can find our interview questions by clicking the following link: https://wasimalsaqaf.files.wordpress.com/2017/07/interview-questions.docx. Our interview questions are organized in two parts. The first explores the organizational context of the interviewee, while the second focuses on the practices experienced by the participants while engineering the QRs in one particular project of their choice.

Third, for the purpose of our data collection, the first author interviewed seventeen agile practitioners (participants) from different organizations. The interviews were conducted in Dutch since all the organizations and participants were located in the Netherlands. The term ‘organization’ used in this paper refers to the organization that employs the participant and not the organization where the participant performed the project under investigation. The organizations included in the case study all claimed to follow agile development methodologies. Three of the organizations have a long history in IT consulting. They employ highly skilled consultants and IT coaches specialized in ADMs, among other subjects. Two organizations provide customized IT services. One of them is specialized in providing transport services and the other provides administrative software packages. Both organizations have been developing their software using an ADM for several years. The sixth organization in our study is a big government agency that adopted an agile large-scale framework several years ago. The anonymized information about the organizations is summarized in Table 1. Due to confidentiality agreements with the participants all data that refers to the participants and/or to the organizations employing them, is anonymized. The second column of Table 1 indicates the approximate size of each organization based on the number of its employees. The third column shows how many projects from each organization we have included in our study. The rightmost column shows how many participants from each organization joined our study.

TABLE 1. MULTI-CASE STUDY ORGANIZATIONS

Organization	Size in employee's number	# of projects	# of participants
O1	Medium (51 – 200)	2	4
O2	Medium (51 – 200)	1	2
O3	Large (200 – 500)	1	1
O4	Large (300 – 700)	3	3
O5	Enterprise (10000 – 30000)	3	3
O6	Enterprise (50.000 – 100.000)	4	4

Table 2 presents the studied projects' settings. All the studied projects used Scrum [15] as their ADM. The second column of Table 2 shows the total number of team members and the number of agile teams in the project, e.g. project P1 had 21 team members that formed 3 distributed teams. The third column shows which scaled-framework is used by each project. A cell with 'none' means that no framework was used. The rightmost column indicates the application domain.

TABLE 2. MULTI-CASE STUDY PROJECTS

Project	# members / teams	Scaled-Framework	Domain
P1	21 / 3	none	Public sector
P2	24 / 2	none	Public sector
P3	117 / 13	SAFe [16]	Government
P4	30 / 3	none	Commercial
P5	50 / 5	Scrum of Scrums [17]	Banking
P6	175 / 25	SAFe [16]	Commercial navigation
P7	56 / 7	none	Public sector
P8	12 / 2	none	Public sector
P9	28 / 4	none	Government
P10	40 / 6	none	Health care
P11	27 / 3	SAFe [16]	Government
P12	24 / 3	SAFe [16]	Government
P13	14 / 2	none	Insurance
P14	200 / 22	Spotify [18]	Telecom

As we could see from Table 2, the participating organizations employ a variety of agile approaches for their projects. Some of the organizations use large-scale frameworks such as Scaled Agile Framework (SAFe) [16] and Scrum-of-Scrums [17].

We note that some participants performed more than one role in the respective project, so the number of roles (20) is larger than the number of interviewees (17). Next, Table 3 indicates the years of work experience each participant has in general in the field of Software Engineering and the role(s) (s)he performed in her/his respective projects which were described in Table 2. As indicated earlier, we included a broad variety of backgrounds, in order to explore the phenomenon of interest from multiple perspectives.

TABLE 3. YEARS OF EXPERIENCE AND ROLES OF THE PARTICIPANTS

Participant	Years of experience	Project	Role
PA1	4	P1	Software Developer
PA2	20	P1	Software Developer & Software Architect
PA3	15	P2	Scrum Master
PA4	36	P2	Software Tester
PA5	21	P3	Scrum Master & Software Tester
PA6	6	P4	Scrum Master
PA7	20	P5	Agile Coach
PA8	22	P6	Agile Coach & Product Owner
PA9	10	P7	Software Architect
PA10	29	P8	Delivery Manager
PA11	25	P9	Software Architect
PA12	22	P10	DevOps Manager
PA13	17	P11	Scrum Master
PA14	15	P12	Software Designer
PA15	18	P7	Information Analyst
PA16	5	P13	Software Developer
PA17	7	P14	Agile Coach

The interviews were conducted between February and April 2017. The length of the interviews varied from 50 to 95 minutes. Each interview started with introduction of the research objective and the structure of the interview. The researcher informed the participants further about their rights and responsibilities towards the research. All interviews were audio-recorded to avoid loss of data.

Our last step was the data analysis. The audio files were transcribed to a written version by a professional external organization. We chose not to do the transcription ourselves to avoid any interpretation bias that could be passed into the transcripts by the researchers involved in preparing and taking the interviews. The analysis process in this paper was done based on the grounded theory method of Charmaz [19]. We chose it,

because it is suitable for qualitative exploratory research where theory should emerge from the data. Thereafter the first two researchers read the transcripts separately and inductively applied descriptive labels (called codes) to segments of texts of each transcript. Table 4 provides two examples of the process of coding a segment of text. In the next step, the researchers involved in the analysis stage came together and discussed the descriptive codes they derived. Similar descriptive codes were combined in higher-level categories. Different descriptive codes were resolved by conducting an argumentative discussion [20] between the researchers to reach a shared rationally supported position and then combined in higher-level categories. No unresolved different descriptive codes remained after this step. The results of this process were reviewed by the third researcher. Concerning our use of Charmaz' grounded theory method, we make the note that we employed the coding and the code-comparison practices of grounded theory to the qualitative interview data for the purpose of analysis only. We did not aim at a full-blown grounded theory study which includes the processes of theoretical sampling and saturation. Our choice of using grounded theory practices for data analysis only agrees with Matavire and Brown [21] who profiled the use of grounded theory in information system research, and Ramesh et al. [5] who also used grounded theory coding for data analysis exclusively.

TABLE 4. TEXTS AND CODES

Original text	**Codes**
PA8: "*To avoid any conflicts and useless discussions with the business representative we decided to divide the cake. Hence, we maintained three product backlogs*"	Division of requirements documentation
PA4: *"We wanted to make the software modular, so we wanted to have the frontend and the backend really well separated. So we said: you know what, we force that by assigning them to two different teams. One team has to implement the frontend and the other the backend. Hard agreements need then to be made regarding the interfaces"*	Components' teams

IV. RESULTS

A. RQ1 – What are the mechanisms behind the reported challenges?

1) Implementing QRs based on unstated assumptions

In our multi-case study we observed that agile teams unconsciously make assumptions about the feasibility of QRs, especially when those QRs's implementation depends on resources of the customer that the system was built for. For example, in project P1, agile teams were supposed to implement a highly available system for public use (e.g. 24/7 days). The system depended on collecting its data on a customer's data source which was located behind a proxy firewall. The agile teams did not know about the proxy firewall and assumed the customer's data source would be made available for use. Late in the development cycle, the agile teams discovered that due to other QRs (security) the customer's data source was only available during working hours. The teams were forced then to ad-hoc rearchitect the system by introducing a copy of the customer's data source with higher availability. However, this ad-hoc work-around caused the emergence of data consistency requirements.

Moreover, in multiple agile projects it turned out that new QRs could and did emerge during the development cycle. However, those requirements remained ambiguous until the Product Owner (PO) acknowledged the need for their implementation and hence specified them clearly and unambiguously. In situations where the PO could not specify those QRs, agile teams had to proceed with implementing them based on their judgment. PA2 explains: *"after a while I saw a pattern in how different versions of maps were generated. We could not verify this versioning pattern with the PO, since nor the PO, nor the customer could answer our questions clearly. We decided to implement this versioning's pattern in our software anyway. Fortunately, there was no need to rollback our implementation".*

2) Priority assigned to conflicting QRs turns out to be suboptimal

QRs are often related to specific functional requirements (FRs) and rarely act upon the entire set of FRs for a project [22] . In agile context, uunrecognized and conflicting QRs can land on the Product Backlog (PB) [15] with different priorities because of their related FRs. The implementation of a QR related to a FR with a higher priority could result in limiting the implementation of a QR related to a FR with a lower priority or eliminating it at all. In project P7 the agile teams had two important QRs among others, namely security and performance. The security requirements were related to authentication and authorization functions which were of high priority. Whereas, the performance requirements (e.g. no longer than three second response time) were related to end-users data retrieval functions with lower priority. The security requirements were implemented by one agile team and resulted in implementing several security filters the data had to go through before being submitted to the end users. Another agile team had to retrieve the data and made it within three seconds available to the end users. However, by the time the performance related FR got a higher priority, it was not possible to retrieve the data within three seconds. At the end, the performance requirement was dropped since enforcing its implementation would result in unaffordable costs.

3) Emerging QRs are hard or impossible to implement in the chosen architecture

Architecting software is a solution-related activity, while discovering and specifying the requirements is a problem-related activity [23]. Agile development encourages Just-In-Time (JIT) requirements analysis and implementation which lead to JIT software architecture activities [24]. Therefore, agile teams should be continuously looking for appropriate solutions (e.g. re-architecting the software) when understanding of the problem domain changes (e.g. the emerge of new requirements). We observed that ALSD teams defined and agreed on the overall system architecture early in the development cycle

based on limited requirements knowledge. This approach could result in QRs to be infeasible at the time the needs for them emerge. In project P7, agile teams agreed at the beginning of the project on using an event-driven architecture. The choice for this architecture was motivated by the number of existing systems that the new system needs to communicate with. Besides, the teams agreed on making the events as small as possible to avoid network overhead and gain extra performance. In an advanced stage of the project the need for more complex events with more data grew but couldn't be fulfilled due to architectural limitations. As a work-around, the teams queried the database multiple times to retrieve the needed data in sequential small chunks which negatively affected the performance as one of the most important QR's of the project.

4) Focusing on one's own component and losing the big picture

QRs are by nature cross-cutting requirements. Therefore, the right implementation of QRs could require a well-structured interaction of different system components (Fig.1). Agile teams responsible for implementing specific components take full ownership of the respective components but may take a careless attitude toward the overall characteristics of the system. PA15 explains: *"We had a software architect responsible for designing the financial component. He overdesigned the system to make it as generic and flexible as possible which resulted in very complex architecture. We could not integrate this component with other parts of the system because we did not understand what goes through it. Therefore, we were forced to re-architecture the component to be able to integrate it"*.

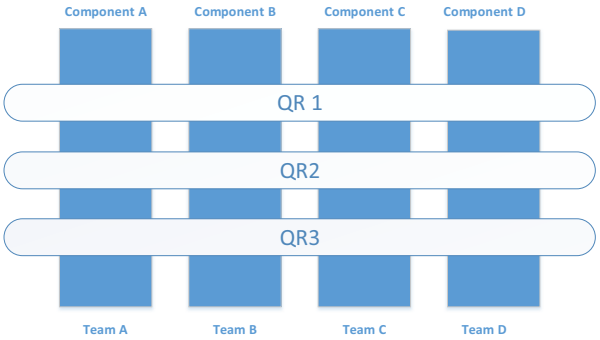

Fig. 1. Cross-cutting QRs

5) Uninformed choice of an inappropriate communication model impedes the implementation of QRs

The agile philosophy − in contrast to waterfall − encourages interaction over following processes [25]. Therefore, agile teams need to establish well-structured and unambiguous communication channels to ensure the right implementation of the required QRs. The maturity of the involved teams is crucial for establishing and leading their own communication channels. [26] defines two models to coordinate the communication between teams: 1) Coordination between teams is done by a higher authority and 2) Coordination between teams is done by the involved teams themselves. The first option assumes that the teams maturity is not enough to coordinate the communication effectively and therefore a higher authority in the form of managers is required. PA6 explained: *"There was no one who banged his fist on the table and said: this is it. There were a lot of opinions, all of which were considered equally important"*. The teams' maturity should be examined up-front to determine the appropriate coordination model. In our multi-case study we did not observe any systematic process used by the agile organization to determine the appropriate coordination model. Choosing sporadically a coordination model that did not match the maturity levels of the teams, in turn, could result in missing QRs. In project P1, text documents had to be made available for end-users to search through. The documents were developed by one team and made available for end-users by another team. This is on the assumption that the documents were correct and accurate. PA1: *"We had agreements about, for example, the validity of the documents. We agreed to put the word "expired" in the name of the document when a document is no longer valid. If the communication between the teams has not gone well – what actually happened- the end-users could consult document which did not reflect the reality at that moment"*.

6) Customers are not interested in internal QRs

When it comes to requirements that are not visible to the end users, the PO as well as the customer are not interested. This moves the focus of the teams from invisible QRs to visible requirements. Besides, the pressure that organizations put on agile teams to deliver functionality together with the lack of interest of the customer in internal QRs gives agile teams a licence to neglect the internal QRs. PA2: "*It doesn't interest the PO, the customer at all, how you solve internal technical requirements"*. PA14: *"To enhance maintainability we had the possibility to switch to a newer version of the currently used framework. However, the customer rejected our suggestion since maintainability had in his perspective a lower priority"*. PA8: *"What you also saw in agile projects, that when the pressure on the teams is high to deliver functionalities, you saw that the teams remove invisible QRs from the PB to let FR get higher priority. They thought nobody will care about invisible QRs "*.

7) Project-wise thinking prevents the right implementation of QRs

In our multi-case study we observed that those agile organizations that are responsible both for the implementation of the software and for its maintenance, pay much more attention to internal QRs than agile organizations that were only hired to implement the desired system. This mentality was described by PA8 as 'project company versus product company', which in fact is translated into keeping a short-term perspective on the project versus a long-term perspective: *"What you actually try to do in agile, is to move on from project company to product company. In product company you think more in the future and determine early how you will treat QRs, since every crap you deliver will return back to you again. While project-wise working is more like after me the deluge. When the deadline is approaching, teams have the tendency to compromise QRs in favour of finishing the project in time"*.

8) Not clearly specifying test criteria

To ensure the right implementation of QRs, teams have to verify at some point the existence of the desired QRs. However, to verify the satisfaction of QRs, they need to be unambiguously specified. In agile projects where (conflicting) requirements emerge, agile teams together with the customer fail to specify clearly the scope of the desired QRs. In those cases, verifying the satisfaction of QRs became a matter of feeling instead of measurement. PA4: *"QRs are not specified, neither the happy flow, nor the alternative flows. I try to test both based on my knowledge and experience"*. Besides, the impact of conflicting QRs should be examined to specify the acceptable level of satisfaction of each QRs for the customer. By not doing that, agile teams actually leave the testers in the dark. In project P12 where security and capacity were conflicted, PA14 explains: *"after delivering the software we got calls from end-users telling us they could not log in. Investigating the log database showed that our security component was not able to handle the login of more than 50.000 users simultaneously. This issue was not tested, since it was not clearly specified. Hereafter we decided to limit the number of users that simultaneously can login to 50.000"*.

9) Focusing on QRs of a specific viewpoint and neglecting those of other viewpoints

Agile teams get their prioritised requirements from the PO. The PO is in the most cases a business representative with particular domain knowledge in his/her area. However, the PO is tasked to speak on behalf of all users from multiple domains, in a project. As the PO may not know all those domains equally well, he/she can pass bias into the prioritization:(s)he could prioritize the QRs of the viewpoint (s)he represents higher than those of other viewpoints. In this way, the PO could put pressure on the team to drop any QRs that (s)he did not request personally, especially in the face of a deadline. Hence, important QRs of different viewpoints could turn out to be in an impasse. P13 had to build a mobile native application for an insurance company. The PO formulated security and usability as the most important QRs. Since mobile devices had limited capacities, the agile teams chose to keep the data alive as a background mobile process for maximum five minutes (e.g. usability) after that a new login should be performed (e.g. security). The PO was satisfied enough with this solution, since the solution met the requirements of the viewpoint (s)he represented. However, users of old mobile devices were less happy since old devices claimed always the memory of applications as soon as they get in the background. Which meant that users of old devices lost their data and had to login repeatedly and more frequently then users of new devices had. Neglecting QR's of other viewpoints can also happen due to overlooking important stakeholders as indicates by P7: *"We had to identify the operational team early in the project as a stakeholder. In our project, the operational team got involved at the end of the project. That was a mistake in my opinion, since a lot of the operational requirements were actually QRs"*.

10) Adopting legacy architectural decisions complicate the implementation of QRs of the new system

Distributed agile teams who are re-implementing a legacy system get advised by the PO to collect important QRs from the existing legacy system. From the PO's point of view, since the new system needs to provide among others the same functionalities as the old one, agile teams have to collect the requirements -including QRs- from the existing system. The PO provides them only with new requirements for functionality that does not exist in the legacy system. By lack of documentation, collecting old requirements is sometimes interpreted by the PO as well as by the agile teams as cloning the current architecture and design. Cloning the legacy architecture could complicate the implementation of the new QRs of making them even infeasible. Project P13 was supposed to move a cross-platform mobile application to new platform-dependent (native) application. The old system used cloud storage to enhance security and reusability. The data on the cloud should be synchronised with that on the mobile application based on comparing hash codes. In the new native application, recalculating the hash codes to synchronize the data by every login resulted in a long login process which reduced the usability of the native system. The agile teams were forced to introduce a work-around solution since re-architecting the system was prohibitively expensive.

11) Moving to agile with a waterfall mind-set

In the waterfall approach, requirements (FRs and QRs) are collected up-front, documented and handed to the software developers to implement them. It is not up to the developers to discover the needed requirements nor to verify them. As R. Davies [27] describes the behaviour of waterfall developers: *"When requirements come from someone else and someone else tests my work, I don't need to know the environment that my software will be running in"*. On the other hand, agile does not make any differences between the kind of requirements (FRs and QRs) and defines both as user stories (the most used documentation technique as in [28]). Agile developers are expected to collect the needed requirement, elaborate them further in face-to-face communication sessions and implement them just in time. In our participants' projects, we observed that some of the agile developers still had the waterfall mind-set. They implement the user stories that the PO ask for, but do not bother themselves with figuring out what QRs might be needed unless the system begins to expose rigidity. PA1: *"It was not known to the developers that there is a system in between"*. PA8: *"I requested the test manager to give us more days to test the software. I told her the tests we run right now are more waterfall-like while we are doing scrum"*.

B. Mapping mechanisms to challenges.

We mapped the identified mechanisms in Sect. IV to the challenges reported in our previous study [6] by using Conklin's Dialog Mapping technique for qualitative data structuring [29]. It helped us organize the results of our data analysis from each interviewee, by forming challenge-mechanism-arguments structures. The challenges are those reported in [6], the mechanisms are those reported in the previous sub-section and the arguments are the interviewee's facts and reasoning that supported the mechanisms. Table 5 presents our mapping.

TABLE 5: MAPPING CHALLENGES TO UNDERLYING MECHANISMS

Challenges	Mechanisms
Teams coordination and communication challenges	-Implementing QRs based on unstated assumptions. -Priority assigned to conflicting QRs turns out to be suboptimal -Uninformed choice of an inappropriate communication model impedes the implementation of QRs -Focusing on one's own component and losing the big picture.
Quality Assurance challenges	-Customers are not interested in internal QRs -Not clearly specifying test criteria -Focusing on QRs of specific viewpoint and neglect those of other viewpoints -Moving to agile with a waterfall mind-set
QRs elicitation challenges	-Customers are not interested in internal QRs -Focusing on QRs of specific viewpoint and neglect those of other viewpoints -Project-wise thinking prevents the right implementation of QRs
Conceptual challenges of QRs	-Implementing QRs based on unstated assumptions -Uninformed choice of an inappropriate communication model impedes the implementation of QRs -Customers are not interested in internal QRs -Focusing on QRs of specific viewpoint and neglect those of other viewpoints -Moving to agile with a waterfall mind-set
Architecture challenges	-Priority assigned to conflicting QRs turns out to be suboptimal -Focusing on one's own component and losing the big picture. -Emerging QRs are hard or impossible to implement in the chosen architecture -Adopting legacy architectural decisions complicate the implementation of QRs of the new system

Therein, the first column shows the categories of challenges as reported in [6]. The second column reports the mechanisms which could result into the challenges of the respective category in the first column. Each mechanism could produce one or more challenges. For example: the lack of interest of the customer regarding QRs (e.g. Customers are not interested in internal QRs) can lead to QRs test specifications and user acceptance challenges (e.g. Quality Assurance challenges). Besides, it also can produce QRs elicitation challenges since the customer does not pay enough attention to internal QRs. Moreover, it can further produce a mix of different QRs specification approaches (e.g. Conceptual challenges of QRs). On the other hand, each challenge can be produced by one or more mechanisms. For example, architectural challenges could be produced by insufficient prioritizing of conflicted QRs, having the agile teams focus on their own components, and losing the big picture of what the whole system needs to be. Furthermore, adopting architectural decisions of legacy system in the new system and making unchangeable architectural choices based on limited knowledge early in the development cycle could also produce architectural challenges.

C. RQ2 – What are the practices agile teams currently use to mitigate the impact of the reported challenges.

1) Maintaining an assumption wiki-page

In situations where the PO cannot provide clarity about the specification of QRs, agile teams make their own assumptions. These assumptions are also made when the teams think that a QR is self-evident and not worth to be verified. Those assumptions are collected in an accessible wiki-page and implemented together with FRs in different sprints. PA2: *"Halfway through the project we created a wiki-page where we put our assumptions about what might have been QRs. Those were user stories that the PO could not specify enough like how specific the performance should be, whether we have to integrate the current system with other systems already used by the customer, or how to treat versioning. Those were specific technical issues which did not interest the PO".* The assumed QRs could be rolled back if the PO or the customer did not accept them after implementation.

2) Use multiple product backlogs to include requirements of different viewpoints

In the experience of our participants, some agile teams explicitly acknowledge that different viewpoints have different interests and QRs. Distributed teams could use different PBs to document the requirements of different viewpoints. Project P6 had three different PBs. One was the responsibility of the PO and was filled up with user stories that represented the customer's business desires. Another was filled up by the software architect and represented architectural related QRs and the last one was the responsibility of the operation manager and filled up with FR and QRs related to the process of continuous delivery. PA8 explains: *"To avoid any conflicts and useless discussions with the business representative we decided to divide the cake. Hence, we maintained three PBs. The sprint velocity was also divided to reflect the three PBs".*

3) Use automated monitoring tools

Agile teams make use of tooling (e.g. SONAR[1]) to monitor the quality of their software under development. QRs that are related to the internal operation of the software were implemented as rules in the monitor tool. The tool then raises a warning when the defined rules are overridden. PA11: *"We tried to automate the Definition Of Dones (DoDs)* [15] *as much as possible and implement them in Sonar. For example we agreed to use where possible in our code the java keyword Final. It is a small technical detail that can improve the performance. Hence, we implement a Sonar rule that checks de code and gives a sign if it finds no Final in a place where it should". PA9: "We used Sonar as quality application. Code quality rules were implemented in Sonar and any violations of those rules was reported by Sonar".*

4) Reserve part of the sprint for important QRs

QRs which in the opinion of the PO do not have business value could turn out to be neglected more easily than others. The neglect of those QRs may result in inflexible and difficult to maintain system. In project P1, agile teams found a way to work around this PO behaviour. They collected the internal QRs in a different PB and agreed with the PO to implement one of those QRs each sprint as long as time permits. PA2: *We said to the PO, you have to give the teams the space to implement one technical QRs each sprint. The teams did use this space to improve the quality of the software. The PO doesn't even know which QRs we were implementing*. Other agile teams reserved part of the sprint velocity to be dedicated to important QRs. PA14:*" We have reserved 30% of each sprint for important QRs, such as Maintainability, Performance and Security".*

5) Sprint allocation based on multiple PBs

Distributed agile teams could use multiple PBs to include QRs of different viewpoints. Hence, they divide the sprint according to the number and importance of the different PBs. Project P6 had three PBs representing the business requirements, architectural QRs and continuous delivery requirements. The sprint capacity was distributed among the different PBs, respectively (40%, 40% and 20%). For example if an agile team has a sprint velocity of 20 story points then the sprint will have 8 story points of business requirements, 8 story points of architectural related QRs and 4 story points of continuous delivery requirements. PA8 explains: "*40 % of the sprint velocity was devoted to business user stories, 40% was dedicated to architectural QRs and 20% for the pipeline requirements*".

6) Establishing preparation team

A preparation team is a team that consists of senior information analysts, senior software architecture and business representatives. This team works in parallel to the other distributed agile teams and is responsible for drafting the PB, making the PB items ready for implementation, defining the overall software architecture (e.g. big design up-front [30]) and assigning ready items from the PB to the distributed agile teams. The preparation team begins from a so-called 'sprint zero' with gathering the most important requirements and defining the overall architecture. The team continues with collecting the needed requirements, making them ready to implement and refining the software architecture during all the sprints after. The team distributes the defined user stories among the distributed teams based on the nature of the user stories and the available skillsets within the different teams. PA9 explains: *"We had specialized teams. Every team was responsible for a specific component (e.g. user screens team, end-user's letters team). Besides, we worked with scenarios that touch every component of the system"*. The preparation team made the user stories outflew from the scenarios ready to implement and assigned them to the right team with the needed skills. PA12 used in project P10 the same approach, however, they called it 'Readiness Team' instead of 'preparation team'.

7) Establishing components teams

As we already indicated, the ALSD projects in our multicase study organize distributed teams around particular components. In our participants' experience, component's teams develop an ownership feeling about the components that they are responsible for. This feeling enhances the internal quality of the individual component and hence the quality of the whole system. PA9: *"At the beginning of the project we assigned a complete scenario to each team to implement. We saw that each component suffered from ambiguity and poor internal quality, since each component could be modified by each team. Then we decided to assign components to teams and make each team owner of a particular component. The internal quality of the components significantly improved".*

8) Establishing QR specialists' teams

We have observed that ALSD projects dedicate the ownership of important QRs to teams with solid knowledge about that particular QR. For example, if security is an important QR for the project, a team with security specialists will be put together and will be assigned the ownership of security requirements. This team should ensure the implementation of the security requirements across the distributed teams. PA13: *"Performance and usability were dedicated to other teams within the organization. security was dedicated to an external team"*. Moreover, in cases that the system fails to meet the customer expectation regarding a particular QR, a team with solid expertise in that particular QR could be set up in an ad-hoc manner to resolve the failure. PA5: *"The previous version of the project did not meet the performance expectation of the customer. We put together a performance team who should analyze and resolve the problem"*.

9) Innovation and Planning Iteration (IP)

IP is a term that is used in SAFe [16]. It is a time period equal to one sprint that an agile team can request in order to work on activities other than delivering user stories with business value. In our study, we have observed that distributed teams request IP to resolve technical debts introduced in previous sprints. PA14 explains: "*Important QRs which had business values (e.g. security, performance) were implemented, other QRs were neglected. We had to fight for an IP to resolve technical debts"*.

[1] https://www.sonarqube.org/

D. Mapping practices to challenges

We mapped each of the reported practices in Sect. IV, to the reported challenges in our previous study [6]. For this purpose, we used the data structuring technique [29] mentioned in Sect. IV.B. Table 6 summarizes this mapping. The first column of the table represents the reported categories of challenges as in [6]. The second column shows the currently used agile practices in distributed context.

TABLE 6. MAPPING PRACTICES TO CHALLENGES

Challenges	Practices
Teams coordination and communication challenges	-Maintain an assumption wiki-page -Establish preparation team
Quality Assurance challenges	-Use automated monitoring tools -Establish QR specialists' teams -Innovation and Planning Iteration (IP)
QRs elicitation challenges	-Establish components teams -Establish preparation team -Reserve part of the sprint for important QRs
Conceptual challenges of QRs	-Use automated monitoring tools -Use multiple product backlogs to include requirements of different viewpoints -Maintain an assumption wiki-page -Sprint allocation based on multiple PBs
Architecture challenges	-Establish preparation team -Establish QR specialists' teams -Innovation and Planning Iteration (IP)

Each of the reported practices in Table 6 could (partially) mitigate the impact of one or more of the reported challenges. For example, the practice of establishing a preparation team is used to solve coordination issues. It helps coordinating the collaboration between the distributed teams by eliminating any ambiguity that could originated by team's miscommunication. Moreover, establishing a preparation team is used to control architectural changes and to prevent unmanaged architectural changes from being happened. Besides, the impact of the reported challenges could be mitigated by implementing one or more practices. For example: the challenges of the category "QRs elicitation challenges" can be mitigated by implementing different practice namely 1) Establishing component's teams to enhance the internal quality of the different components, 2) Establishing a preparation team to collect the requirements and release the PO from being the only source of the requirements, and 3) Reserve part of the sprint to be used for implementing important QRs.

V. DISCUSSION

We have observed that though currently used agile practices could mitigate the challenges reported in distributed ALSD [6], they could also introduce other challenges. In our multi-case study, the ALSD projects with multiple distributed teams organize the teams around defined subsystems or components (e.g. payment component, registration component). This approach to the organization of the work was perceived as "something new" in the organizational settings where our participants were employed. In the experiences of our participants, when a particular team has the ownership over a particular component, the team also takes actions to guard the quality of its own component. This is because the technical debt detected in a particular component will eventually return back to the responsible team. Plus, each component will show stability and clarity since only one team is allowed to touch the code belonging to that particular component. Therefore components' teams [31] could be an effective mitigation against poor internal quality and ambiguity. However, the customer who requested the system, is not interested in the individual components but in the business values delivered by the whole system (made up of these components). If the delivered business value fails to meet the expectation of the customer, the component's teams would not take the responsibility of the failure since each of them is only responsible for a particular component (and no one is responsible for the system as a whole). Moreover, the component's teams need together to agree on well-defined communication protocols to ensure the delivery of the desired business values (e.g. inter-faces). In our multi-case study we have observed that teams are not always capable to steer the communication between component's teams in the right direction especially if the teams suffer from lack of maturity. In case of lack of maturity within component's teams, a higher management role is needed to coordinate the collaboration between the teams [26]. Feature's teams [31] is another approach to organize distributed agile teams. The idea behind the feature's teams is to organize the teams around requested business values (e.g. system features). A feature team is allowed to touch all existing components to ensure the right implementation of the required feature. Feature's team could be the answer to the challenges of using component's teams [31]. However, according to our participants feature's teams lack the responsibility over individual components which could result in delivering subsystems of poor internal quality. PA9 explains: *"When we started the project we divided the teams into scenario teams. Each team was responsible for the implementation of a whole scenario from the user interface through the database layer. We saw then that each component suffered from ambiguity and unclear guidelines. Because each team had its own way of working and there was no ownership for the components, spaghetti code began to arise and it was difficult to understand the structure of the different components"*.

We have also observed that agile teams, specially less mature ones, still need a higher management role that coordinate the collaboration between the distributed teams which is totally against the spirit of agility [15][25]. The agile spirit advocates the self-organizing team concept which should be capable to organize the needed activities itself to perform the required work. In our multi-case study we found that agile teams use practices that move the elements of self-organizing to a higher management team (e.g. preparation team, ready team). This results in agile teams that are not self-organized anymore. Establishing a preparation team mitigates some challenges, e.g. unmanaged architectural changes, coordinating the communication between the distributed teams and improve the requirements elicitation and specification processes. However, establishing a preparation team takes the flexibility of agile a step backward toward waterfall since the distributed agile teams are stripped out from their self-organizing elements. Besides, the use of a preparation team could introduce other challenges such as making definitive architectural choices early in the process that prevent emerged QRs from being implemented. P7 explains this challenge: *"We made at the beginning of the project an architectural choice. This choice include the use of small massages in an even-driven architecture to enhance the performance. I think it was not a very good choice. However, now we cannot change it anymore and we just have to work with it"*.

An interesting observation from Table 1 is that some of the mechanisms are not specific to ALSD projects, but are known from waterfall projects, e.g. that customers have less interest in internal QRs, which in turn create QRs elicitation challenges (see the third raw in Table 1). However, the solution practices that agile teams come up with in order to counter these challenges are different compared to those in waterfall projects. While waterfall projects may resort to increasing the number of documentation (e.g. using standards, checklists and explicitly defined QRs-specific terminology [32]), ALSD projects put emphasis on people and propose solution practices that are organizational in nature – e.g. establish component teams, and preparation teams.

Another interesting observation is that agile teams use different approaches to document QRs (e.g. assumption's wiki, multiple PBs, rules in monitoring tools, etc.). This observation exposes the struggle that agile teams experience when it comes to QR's specification and documentation. It also shows that agile teams find creative solutions to cope with the inability of user stories to document QRs [33][34].

VI. Limitations of this Study

We are aware of a few important limitations of our multi-case study, which we examine below by using Yin's checklist [12] concerning generalizability, reliability and construct validity. Because our research does not seek to establish any causal relationships, we do not discuss threats to internal validity.

Generalizability. Would the mechanisms be observed if we would interview other practitioners in other countries? And would the practices resolving the challenges experienced by our participants, be observable in other contexts? We can only claim that the reported mechanisms and possible solutions occurred in the projects reported by our participants and that it might well be possible that they occur elsewhere too − if there are contextual similarities between our participating organizations and other organizations. But it could be the case that other organizations' political contexts may differ from those in this multi-case study, which in turn put at play different mechanisms compared to those in this paper. It could be also possible that organizations experiencing the present mechanisms come up with completely different solution practices that are working equally well in countering the challenges. Therefore our list of mechanisms for the challenges and our list of possible solutions are living documents that may change with future case studies. We do expect that similar contexts in similar organizations [35] (e.g. in the same business sector in the same country) may create the same mechanisms, which in turn might bring agile teams to come up with similar solution practices to the experienced challenges. Our generalization is not that the mechanisms we identified occur in all ALSD projects, but that they may occur more often in ALSD, and are important to understand, prevent and mitigate. Our research provides evidence that these mechanisms are important, because practitioners reported them as such; whether they occur more often requires more research.

Reliability. It might be possible that the researchers passed occupational bias [38] into the research process due to their own business experience (that they accumulated prior to their academic careers). We countered this threat by taking four specific actions: (1) we had the interview protocol and questions reviewed by experienced and senior researchers; (2) a pilot interview was done to test the applicability of the interview questions; (3) all the interviews were audio-recorded and shared with the senior researchers; and (4) the audio files were transcribed by a professional agency.

Another reliability threat is our participants' bias due to a possible lack of honesty [36]. We countered this limitation by taking three actions: (1) we assured that all the participants were volunteers and had the right to refuse answering any question at any time or even leave the interview at any stage without giving a reason; (2) we assured the participants that all information remains confidential and anonymous; (3) the interviewer started each interview by explaining the objective of the research to the participants and the importance of giving accurate and honest answers to the validity and reliability of the research.

Construct validity. We used four measures to mitigate construct validity threats: (1) triangulation of "multiple sources of evidence" (as recommended by Yin [12]), (2) member-checking [12], e.g., during the interviews, the interviewing researcher restated or summarized the information provided by the interviewee and then used follow-up questions to the respective practitioner in order to determine accuracy; (3) peer debriefing [37], which happened through the involvement of senior researchers, and (4) disclosure of researcher bias [37], which was achieved by maintaining a reflective journal. After every interview, the interviewing researcher spent time noting

his immediate observations, thoughts and interpretations before he subjected the data to coding.

VII. Conclusion

In our previous study [6], we have identified 13 challenges which were divided into five categories that distributed teams in ALSD projects face regarding the engineering of QRs. In this paper we have identified 11 mechanisms behind those challenges and nine practices currently used by ALSD teams to mitigate the impact of the reported challenges. Furthermore, we have mapped the identified mechanisms and practices to the reported challenges. Based on this mapping we have found that each mechanism can produce one or more challenges and each challenge can be the result of one or more mechanisms.

This study shows also that a particular practice could mitigate some challenges but also could introduce other challenges. Therefore we advise agile teams to carefully evaluate the used practices and get insight in the challenges that could be mitigated or introduced by the use of that particular practice. Only then agile teams could benefit optimally form the reported practices.

We want also to advise agile organizations to get insight in the maturity of their involved agile teams as early as possible. This insight will actually help an organization to define an appropriate communication model as described in [26]. Agile teams that lack maturity need a higher management role to coordinate the collaboration between the teams, while mature agile teams are self-organized teams, and would be less productive if a management role is involved to supervise their work.

This paper also shows that it is well possible that agile teams have the waterfall approach in their minds. Agile teams have to be careful when moving from waterfall to agile philosophy, since the wrong implementation of both will prevent the teams from gaining the benefits of either approach.

VIII. Further Research

In this paper we have identified the mechanisms behind the challenges we have reported in our previous work [6] and the practices agile practitioners use to cope with the challenges. To identify these, we have analysed the data collected previously [6]. This analysis enabled us to map the identified mechanisms and solution practices to the previously identified challenges. However, we think that it is also worthwhile investigating the possible mapping between the solution practices and the mechanisms. We assume that multiple mechanisms might lead to the use of a particular practice. Also, it might be possible that one mechanism might call for a combination of multiple solution practices. To realize this mapping however more interviews with agile practitioners need to be done since the current data does not provide enough evidence for establishing a reliable and credible mapping. This is the focus of our immediate future research.

References

[1] M. Paasivaara and C. Lassenius, “Scaling Scrum in a Large Distributed Project,” ESEM 2011 pp. 363–367, 2011.

[2] W. Alsaqaf, M. Daneva, and R. Wieringa, "Quality requirements in large-scale distributed agile projects - A systematic literature review", REFSQ'17 pp. 219-234, 2017.

[3] M. Käpyaho and M. Kauppinen, “Agile Requirements Engineering with Prototyping: A Case Study,” RE2015, pp. 334–343, 2015.

[4] V. Sachdeva and L. Chung, “Handling non-functional requirements for big data and IOT projects in Scrum,”. Cloud Comput. Data Sci. Eng., pp. 216–221, 2017.

[5] B. Ramesh, L. Cao, and R. Baskerville, “Agile requirements engineering practices and challenges: an empirical study,” *Inf. Syst. J.*, vol. 20, no. 5, pp. 449–480, 2010.

[6] W. Alsaqaf, M. Daneva, and R. Wieringa, “Quality Requirements Challenges in the Context of Large-Scale Distributed Agile: An Empirical Study,” REFSQ2018, pp. 1–16, 2018.

[7] K. Dikert, M. Paasivaara, and C. Lassenius, “Challenges and success factors for large-scale agile transformations: A systematic literature review,” *J. Syst. Softw.*, vol. 119, pp. 87–108, 2016.

[8] W. R. Fitriani, P. Rahayu, and D. I. Sensuse, “Challenges in Agile Software Development : A Systematic Literature Review,” *Icacsis*, pp. 155–164, 2016.

[9] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa, “Requirements Engineering Challenges in Large-Scale Agile System Development,” REFSQ 2018, pp. 6–8, 2018.

[10] K. Petersen and C. Wohlin, “The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study,” *Empir. Softw. Eng.*, vol. 15, no. 6, pp. 654–693, 2010.

[11] K. H. Rolland, “‘Desperately’ Seeking Research on Agile Requirements in the Context of Large-Scale Agile Projects,” XP2015, 2015.

[12] R. K. Yin, *Case Study Research Design and Methods*, Sage, 2013.

[13] C. Boyce and P. Neale, “Conducting in-depth interviews: A Guide for designing and conducting in-depth interviews,” *Evaluation*, vol. 2, no. May, pp. 1–16, 2006.

[14] Z. Eisikovits and C. Koren, “Approaches to and outcomes of dyadic interview analysis,” *Qual. Health Res.*, vol. 20 12, pp. 1642–55, 2010.

[15] K. Schwaber and J. Sutherland, “The Scrum Guide,” *Scrum.Org*. p. 17, 2016.

[16] D. Leffingwell and R. Knaster, *SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering*, 1st ed. Pearson Education, 2017.

[17] C. Larman and B. Vodde, *Practices for Scaling Lean & Agile Development*. Addison-Wesley Professional, 2010.

[18] H. Kniberg and A. Ivarsson, “Scaling Agile @ Spotify - with Tribes, Squads, Chapters & Guilds,” 2012.

[19] K. Charmaz, *Constructing grounded theory: a practical guide through qualitative analysis*, Sage, 2006.

[20] D. Hitchcock, “The Practice of Argumentative Discussion,” *Argumentation*, vol. 16, no. 3, pp. 287–298, 2002.

[21] R. Matavire and I. Brown, “Profiling grounded theory approaches in information systems research,” *Eur. J. Inf. Syst.*, vol. 22, no. 1, pp. 119–129, 2013.

[22] M. Kassab, O. Ormandjieva, and M. Daneva, “An ontology based approach to non-functional requirements conceptualization,” *ICSEA 2009,* pp. 299–308, 2009.

[23] T. E. Fægri and N. B. Moe, “Re-conceptualizing requirements engineering: Findings from a large-scale, agile project,” *XP2015 - XP '15 Work.*, pp. 1–5, 2015.

[24] C. M. Robert and M. Micah, *Agile Principles, Patterns and Practices in C#*. Prentice Hall, 2006.

[25] Agile Alliance., *Manifesto for Agile software development*. 2001.

[26] J. Apperlo, *Management 3.0 Leading agile Developers, Developing agile Leaders*. Pearson Education, 2011.

[27] R. Davies, “Non-Functional Requirements: Do User Stories Really Help?,” *DevOpsDays*, 2009. [Online]. Available: http://www.methodsandtools.com/archive/archive.php?id=113.

[28] J. D. R. V Medeiros, D. C. P. Alves, A. Vasconcelos, C. Silva, and E. Wanderley, “Requirements engineering in agile projects: A systematic mapping based in evidences of industry,” *CibSE*, pp. 460–473, 2015,.

[29] J. Conklin, “Dialog Mapping: Reflections on an Industrial Strength Case Study,” *Vis. argumentation*, pp. 1–15, 2003.

[30] M. A. Babar, P. Kruchten, and P. Abrahamsson, “Agility and

Architecture: Can the coexist?", IEEE Software, vol. 27, no. 2, pp.16-22, 2010

[31] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, First Edit. Pearson Education, 2011.

[32] M. Daneva, A. Herrmann, and L. Buglione, "Coping with Quality Requirements in Large, Contract- Based Projects," *IEEE Softw.*, no. 6, pp. 84–91, 2015.

[33] M. Daneva *et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1333–1353, 2013.

[34] J. Nawrocki, M. Ochodek, J. Jurkiewicz, S. Kopczyńska, and B. Alchimowicz, "Agile Requirements Engineering: A Research Perspective," in *SOFSEM*, 2014, vol. 8327, pp. 40–51.

[35] S. Ghaisas, P. Rose, M. Daneva, K. Sikkel, and R. J. Wieringa, "Generalizing by Similarity : Lessons Learnt from Industrial Case Studies," *CESI*, 2013, pp. 37–42.

[36] N. King and C. Horrocks, *Interviews in Qualitative Research*. SAGE Publications Ltd, 2010.

[37] S. B. Merriam, *Qualitative Research and Case Study Applications In Education*. Jossey-Bass Publishers, 1998.

[38] P. N. Adler, *Membership roles in field research*, First edit. SAGE Publications Inc, 1987.