2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement

Agile Quality Requirements Engineering Challenges: First Results from a Case Study

Wasim Alsaqaf
School of Computer Science
University of Twente
Enschede, the Netherlands
w.h.a.alsaqaf@utwente.nl

Maya Daneva
School of Computer Science
University of Twente
Enschede, the Netherlands
m.daneva@utwente.nl

Roel Wieringa
School of Computer Science
University of Twente
Enschede, the Netherlands
r.j.wieringa@utwente.nl

***Abstract*— Agile software development methods have become increasingly popular in the last years. Despite their popularity, they have been criticized for focusing on delivering functional requirements and neglecting the quality requirements. Several studies have reported this shortcoming. However, there is little known about the challenges organizations currently face when dealing with quality requirements. Based on a qualitative exploratory case study, this research investigated real life large-scale distributed Agile projects to understand the challenges Agile teams face regarding quality requirements. Eighteen semi-structured open-ended in-depth interviews were conducted with Agile practitioners representing six different organizations in the Netherlands. Based on the analysis of the collected data, we have identified nine challenges Agile practitioners face when engineering quality requirements in large-scale distributed Agile projects that could harm the implementation of the quality requirements and result in neglecting them.**

Keywords— Empirical research method, Quality requirements, Agile, Requirements engineering, Interviews, Case study

I. INTRODUCTION

Engineering the requirements in Agile development methods (ADMs) is different from the way it is done in the traditional Waterfall approach. While the traditional approach aims to fully identify the requirements up-front by following sequential practices [1], requirements in ADMs are collected just in time (JIT) based on face-to-face communication sessions with the business representative [2] who communicates the elicited requirements towards the Agile teams. The Agile practices used to engineer the requirements are criticized for focusing only on functional requirements (FRs) and neglecting quality requirements (QRs) [3][4][5]. Neglect of QRs may result in systems that do not meet the user expectations. In small co-located projects, this can be repaired relatively easily by adapting the next batch of requirements and fixing the part of the product already delivered. This is however not possible in Agile large-scale distributed (ALSD) projects where the teams are spread over multiple locations and there are no possibilities for ad-hoc coordination and communication among team members and with clients. Our recent systematic literature review (SLR) [6] on engineering QRs in ALSD, has indicated twelve QR challenges in ALSD that could lead to compromising QRs and as a consequence not meeting the user expectation. Our SLR has also reported lack of empirical evidence on how Agile projects handle QRs systematically, in their entirety since there were no studies found which were dedicated to QRs as a whole in ALSD settings. In this paper we want to shed light on the challenges distributed Agile teams face when engineering the QRs in real-life settings. To this end, we have conducted an empirical study using semi-structured open-ended in-depth interviews to understand the current challenges that Agile practitioners cope with when it comes to QRs. We have interviewed eighteen practitioners with different expertise (e.g. testers, architects, scrum master, managers) and from different domains (e.g. Banking, Public transportation, Tax agency) working for Agile project organizations in the Netherlands. The main objective of our empirical study is to explore the challenges faced by distributed Agile teams regarding the engineering of QRs. Based on our objective we set out to answer the following research question (RQ): *What are the challenges Agile practitioners face when engineering the QRs in distributed large-scale settings?* If we gain insights into the challenges Agile practitioners face when engineering the QRs, then we will be able to understand better the problems that cause the neglect of QRs. The next step will be then to investigate how Agile practitioners currently deal with the identified problems. Based on that we will gather insights in how we could assemble an appropriate and cost-effective solutions to QRs problems in ALSD. We consider an Agile project 'distributed' if it consists of more than one team and its teams are distributed in terms of the distribution models described by Larman and Vodde [7], which are: 1) Multi-site teams - the teams work on different locations. Each team is single site. 2) Dispersed teams - the teams work on different locations. Each team is multi-site.

The remainder of the paper is organized as follows: Sect. 2 provides related work. Sect. 3 presents the research method, and Sect. 4 − our results. Sect. 5 discusses the results and Sect. 6 – validity threats. Sect. 7 concludes.

II. RELATED WORK

Our recent SLR [6] on engineering QRs indicated the lack of empirical research specifically devoted to the challenges that Agile practitioners face when dealing with QRs as a whole in ALSD [6]. However, there are studies reporting Agile RE challenges [5][8][9][3], which we summarize as follows: The SLR of Inayat et al.[5] focused on the differences between traditional and Agile RE and the challenges of Agile RE. These authors identified seven challenges brought by Agile, one of which was the neglect of QRs. The authors also called for more empirical research to collect evidence on the topic of QRs. Next, Ramesh et al. [3] investigated the ways in which requirements analysis was conducted in sixteen organizations involved in Agile software development. The study identified seven challenges − Problems with cost and schedule estimation, Inadequate or inappropriate architecture, Neglect of QRs, Customer access and participation, Prioritization on a single dimension, Inadequate requirements verification, Minimal documentation, that are posed by Agile practices. Käpyaho et al. [9] reported a case study that investigated whether prototyping can solve the RE challenges brought by ADMs. The study indicated that while prototyping can help with some challenges of Agile RE such as lack of documentation, motivation for RE work and poor quality communication, it does not help with other challenges such as not understanding the big picture and the neglect of QRs. Bjarnason et al. [8] performed a case study to investigate if and how Agile RE can mitigate the challenges of traditional RE and what new RE challenges Agile might pose. The study indicated that Agile addresses some RE challenges such as communication gaps and overscoping, but also causes new challenges, such as striking a good balance between agility and stability, and ensuring sufficient competence in cross-functional development teams. Furthermore, the study reported the need of further research on the impact of Agile in large-scale software development.

III. 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 the QRs is to explore the subject in real life settings. We conducted a qualitative exploratory multi-case study as described by R. Yin [10] to reach our research objective. The case study involves semi-structured open-ended in-depth interviews and is designed according to the guidelines of Boyce & Neale [11]: 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 to be included in the case study. To gain a solid understanding of the challenges of engineering QRs in Agile 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 is in line with research methodologists (e.g. [10][11]).

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 are improved and finalized based on the feedback received from the senior researchers. Thereafter, using the interview questions we conducted a pilot interview with an Agile practitioner to check the applicability of the questions in real-life context. No changes made to the interview questions after this stage. Interested readers can find our interview questions at this website: *https://wasimalsaqaf.files.wordpress.com/2017/07/interview-questions.docx.* We did not include the pilot interview in the case study because the respective project setting did not meet the requirement of project distributedness. The set of interview questions is composed of two parts. The first explores the settings to understand the project context, while the second focuses on the practices the participants experienced in engineering the QRs in one particular project of their choice.

Third, the data collection. The interviews were conducted by the first researcher. He 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 consultancy. They employ high skilled consultants and IT coaches specialized, among other things, in ADMs. One is a big government organization that has adopted an Agile large-scale framework for several years. The last two organizations provide customized IT services. One of them is specialized in providing Transport services and the other provides Administrative software. Both organizations use an ADM to develop their software for several years. Scrum [12] was the most used ADM. Some of the organizations use large-scale frameworks such as Scaled Agile Framework (SAFe) [13] and Scrum-of-Scrums [14]. The anonymized information about the organizations is summarized in Table 1. Due to confidentially agreements with the participants all data that refers to the participants and/or to the organizations employing them, are anonymized. The second column 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 I. CASE STUDY ORGANIZATIONS.

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

Table 2 presents the studied projects' settings. All the studied projects used Scrum as their ADM. One project (P13) fell into the dispersed team category, while the other 13 projects (P1-P12 and P14) were composed of multi-site teams. The second column of Table 2 shows the total number of team members and the number of Agile teams in the project. For example project P1 had 21 team members that formed 3 distributed teams. The third column shows which scaled-framework is used by the project. A cell with 'none' means that no framework was used. The rightmost column indicates the application domain.

TABLE II. 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 [13]	Government
P4	30 / 3	none	Commercial
P5	50 / 5	Scrum of Scrums [14]	Banking
P6	175 / 25	SAFe [13]	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 [13]	Government
P12	24 / 3	SAFe [13]	Government
P13	13 / 2	none	Insurance
P14	200 / 22	Spotify [15]	Telecom

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). As it is common in qualitative exploratory studies [10], we included a broad variety of backgrounds, in order to explore the phenomenon of interest from multiple perspectives.

Next, Table 3 indicates the years of work experience each participant has in general in the field of Software Engineering and which role(s) (s)he performed in her/his respective projects which were described in Table 2.

TABLE III. 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 length of the interviews varied from 50 to 95 minutes. At the beginning of each interview, the research objective and the structure of the interview was explained to all participants. 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 by 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 described by Charmaz [16]. This method is suitable for qualitative exploratory research where theory should emerge from the data and where preconceived beliefs are not allowed. Thereafter the first two researchers (Alsaqaf, Daneva) read the transcripts separately and inductively applied descriptive labels (called codes) to segments of texts of each transcript. In the next step, the researchers involved in the analysis stage came together and discussed the descriptive codes they applied. Similar descriptive codes were combined in higher-level categories. Different descriptive codes were resolved by conducting an argumentative discussion [17] 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.

IV. RESULTS

Our qualitative analysis yielded nine QRs challenges on team level as well as project level. These challenges are described below. We illustrate our findings with quotations from the interview transcripts.

A. QRs infeasibility

In the experience of our participants, discovering that a needed QR is infeasible at an advanced stage of the development cycle may result in refactoring the software architecture and reimplementing the delivered functions. Project P1 was supposed to deliver a system that would have high availability (24/7). However, the system to be delivered should collect its input from an external system which is – due to security reasons, only available for a limited number of hours a day. The development team discovered this issue at an advanced stage of the development cycle which resulted in refactoring the system the team was delivering, in order to support on-line as well as off-line input collection.

B. Teams interaction

Our data indicated that QRs are usually not implemented in a single piece of code. Because the implementation of QRs could span the whole system, they fall under the responsibility of different teams. Therefore, our participants found that the communication between the teams and their members should happen in such a way that ensures the right implementation of the QRs. For example, 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 are correct and accurate. PA1 reports: *"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"*.

C. *Inadequate QRs verification*

This category refers to our participants' observations that QRs are difficult to model and therefore identifying and designing acceptance tests for them may be difficult (as e.g. in [18]). Besides, ADMs lack formal modelling of detailed requirements [3] which makes the process of verifying the QRs more difficult. PA2 describes this challenge: *"The tester has trouble with testing QRs, because there are mostly no clear-defined acceptance criteria or QRs have not been defined so precisely and verifiably"*.

D. *Integration test*

In the experience of the participants, integration tests are critical to the verification of the implemented QRs. This is due to the fact that if QRs must be globally implemented, they impact the entire system and not only the components separately. Therefore, the work of the development teams should be merged at some point to perform integration tests. These tests could happen late in the development period. If these tests reveal QR defects, this could result in extremely costly re-work and refactoring of the existing software architecture. For example, P3 was a large project that used SAFe to coordinate the work among the distributed Agile teams. P3 had sprints of two weeks and shippable increments every six sprints. At the end of each six sprints, the whole set of all shippable increments delivered by the distributed teams was merged and go through an integration test by a devoted integration team (DIT). The DIT needed other four weeks to complete the needed integration tests. QRs related issues discovered by de DIT went back to the particular teams to be resolved. PA5 reported this challenge: *"So what we have done now is actually saving all the work of six sprints and offering it to the integration team at once, while you could actually do the tests in advance"*. The development teams do simulate integration test as part of their own unit tests. However, simulating an integration test is not the same thing as doing a real integration test.

E. *Losing architectural overview*

Software architecture is intimately connected to the achievement of QRs [19]. Changes made to QRs at any time in the development cycle could result in costly changes in the software architecture whereby the earlier architecture become inappropriate for the new QRs [3]. Participant PA5 reports this issue: *"We have a number of developers who are already making changes closely to the architecture. Those developers who often have discussions with the software architect whenever he wants to make architectural changes which actually will undermine the overall performance. However, one time we choose for more performance and the other time not"*. The many changes in the software architecture could lead to fragmentation of architectural knowledge. The architectural knowledge of a particular system component could be limited to the team responsible for implementing the system component and the overall system architectural knowledge to the role of the software architect. Besides, due to minimal documentation and isolated knowledge the knowledge about previous architectural decisions can be lost. This could cause the justification of QRs trade-offs already made to be lost and the software to be less understandable and maintainable.

F. *Teams maturity*

All our participants indicated that in their perceptions, the success of Agile projects relies on the tacit knowledge embedded in the teams. They thought, experienced developers are more likely to make better architectural decision than junior developers (as in [20]). However, teams that are a mix of experienced and junior developers face the problem of transferring the knowledge from the more experienced to the less experienced in a way that allows both sides to share the same knowledge of the system and enhance the overall quality of it by implementing the right QRs in the right way.

G. *QRs identification*

Agile depends on the involvement of the stakeholders to iteratively collect the requirements. Face-to-face feedback sessions are planned to gather stakeholders feedback on the implemented requirements and to let new stakeholder's requirements emerge. However, to collect those requirements, all stakeholders representing the different viewpoints of the system should be identified [21]. Over-looking stakeholders representing any of the system viewpoints may lead to missing requirements and enhance the total project cost. PA7: *"If I look back at the whole project life cycle, I think identifying all the stakeholders and get feedback from them as soon as possible is still the biggest threat to the success of the project"*. QRs are by nature cross-cutting requirements which means that they may influence other requirements of different viewpoints. Accordingly, QRs should be discovered and introduced at the right development stage to avoid extensive rework to the system [22]. Participant PA9 reports this issue: *"Identifying the QRs was a problem for us. The most QRs were not identified in advance and were discovered in a later stage. By that time it was very complex to implement them"*. In addition to the previous observations, we observed that participants do not agree about the nature of QRs and how they should be treated. Participant PA8 does not believe in the distinction between QRs and FRs. Instead, he explained: *"There are only requirements, some of them are describing a change from situation A to situation B, while the others are constraining the change to certain options. Both changes and constrains might be of quality or functional natures and can be placed on the Product backlog as well as the Definition of Done"*. Participant PA11 does not agree with this statement. He sees QRs as constraint on FRs. According to his experience, for QRs to be meaningful, they should be always put in relation to some specific FRs. However, treating the QRs as part of FRs could result in neglecting the QRs if their related FRs have not been recognized as high priority [9]. For example, P2 is a project that should develop a new system to replace an old one. The new system should integrate within the environment where the old one perfectly operated. Therefore, according to participant PA2, the way of storing the data should be kept as it was in the old system to guarantee data consistency. The product owner (PO) of the project did not see changing the way of storing data as high priority. PA2 describing this challenge: *"I had to fight hard to keep the data storage process in the old way, because then the integration with the environment would be so much easier. That was a non-functional requirement, which was not known by the PO"*.

H. QRs visibility

QRs can be broken down into two categories External and Internal [23]. External QRs (EQRs) are visible to the stakeholders and describe how the system should perform the desired function to be of acceptable quality (e.g. security, performance, availability). The stakeholders are very interested in those QRs. Participants PA2 describes the importance of EQRs: *"Yes, performance was very important for the stakeholders"*. Internal QRs (IQRs) describe the ease of understanding, maintaining and extending the system (e.g. maintainability, modifiability, extensibility) and contrary to EQRs are in the first instance barely visible to the stakeholders. In the end they are visible to stakeholders, namely by means of increased maintenance cost. Participant PA8: *"IQRs get attention only when it is really needed and when the system begins to crack"*.

I. Teams coordination

Large Agile projects with multiple teams, face the problem of organizing and coordinating the teams around the so-called Product Backlog Items (PBIs). The PBIs are all the desires that might be needed in the product and are listed in an ordered way in the "Product backlog" [12]. The "Product backlog" is the single source of requirements for any changes to be made to the product. Our participants used various approaches to this situation: (1) Component teams: the teams were organized around particular components of the system such as a database, user interface, etc. (2) Feature teams: the teams were cross-component ones and organized around particular customer features such us login, log processing, etc. (3) Functional teams: the teams were organized around a single function such as a test team or an architecture team. Depending on the context and the system to be implemented, one of the approaches or a combination of two or more could be used. However, since each of them has advantages as well as drawbacks, our participants thought that teams should be careful with their choice because a suboptimal choice could affect the overall quality of the system.

V. DISCUSSION

We have observed that Agile practitioners struggle with approaching the QRs. While user stories are the most used technique in ADMs to document the customer desires [24], our practitioners did not agree on whether the user stories are equivalent to traditional requirements (TR) or not. In our study, participant PA8 experienced the user stories as equivalent to TR and therefore they could be both FRs as well as QRs. In contrast, PA11 believes that user stories always represent FRs, while the QRs are constrains on the FRs and they cannot be specified in isolation from their related FRs. This is in line with Pammi's online article[1] that used the term "3C" to denote the Agile requirement. The term "3C" refers to Card (written user story), Conversation (user story discussion) and Confirmation (user story acceptance criteria). In Pammi's opinion, the 3C is an equivalent to TR and each 'C' cannot be treated separately. Pammi's suggestion is in alignment with [25]. Bjarnason et al. [25] reported based on empirical study the use of user stories complementary with acceptance criteria to formally document Agile requirements. An another interesting observation is about the exact point in time, when to identify the QRs. Participant PA9 experienced not identifying QRs in advance as a challenge for his project. This observation contrasted with the spirit of Agility where requirements emerge throughout the development cycle. However, identifying crosscutting requirements at the wrong stage could result in costly rework.

As part of this work, we also compared our findings with the twelve challenges reported in our 2017 SLR [6]. Concerning the identification and documentation of QRs, our SLR found the following challenges: (1) ADMs do not provide a widely accepted technique for gathering the QRs, (2) the inability of user stories to document QRs and their dependencies, and (3) dependence on the product owner as the single point to collect the requirements Our present results overlapped the SLR findings and revealed that the identification of stakeholders and their QRs at the right development stage is a challenge. Besides, agile practitioners lack of agreement about the nature of QRs makes the process of specifying the QRs unclear. However, more research needs to be done to investigate whether the lack of documenting the QRs is due to the inability of the user stories to document them (as in our SLR) or due to the lack of agreement about the nature of QRs.

Our results also overlap with other previously reported challenges [6], namely, (1) Focusing on delivering functionality at the cost of architecture flexibility, (2) Ignoring predictable architecture requirements, (3) Insufficient requirements analysis, (3) Validating QRs occurs too late in the process and (4) Product Owner's lack of knowledge

Next, we did not find evidence that the challenges: (1) Product Owner's heavy workload and (2) Insufficient availability of the Product Owner, reported in [6] in anyhow threated the success of the projects in our case study.

Last, this paper revealed new challenges that were not reported in our SLR, namely: (1) Teams interaction, (2) Teams coordination, (3) QRs visibility and (4) Teams maturity.

VI. THREATS TO VALIDITY

The first author has an Agile software engineering background, therefore, some occupational bias could be passed to the interview questions as well as the interviews themselves.

[1] *https://www.scrumalliance.org/community/articles/2013/september/agile-user-stories*

This type of bias was reduced by (1) having the interview protocol and questions reviewed by experienced and senior researchers (the second and third authors); (2) conducting a pilot interview to ensure the applicability of the interview questions; (3) recording all the interviews and having the audio files reviewed by the senior researchers; and (4) having the audio files transcribed to a written version by a professional external organization. King et al. [26] reported a lack of honesty that the participants show in their answers to be possible a weakness in interview techniques. To reduce this threat we took the following measures (1) 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; (2) All the participants were ensured that all information will be confidentiality and anonymously treated; (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; and (4) The participants had different backgrounds, disciplines and were of different application domains. This diversity allowed us to investigate and evaluate the same phenomena from different points of view. An another possible weakness of interview techniques reported in [26] is the tendency for the interviewer to ask leading questions. However, this threat is minimal since we conducted a pilot interview to ensure the applicability of the interview questions after having the interview questions reviewed by the senior researchers.

VII. CONCLUSION

Previous studies have reported the engineering of QRs in Agile as a challenge [3][4][5][8][9]. Other studies indicated the lack of empirical studies regarding Agile QRs [5][6]. This study has identified nine main challenges Agile practitioners cope with when dealing with QRs based on a qualitative exploratory case study. which is the primary contribution of our research. Moreover, this study shows that there is actually a conceptual problem when it comes to the identification of QRs. Practitioners have no clear concept of what a QR is, or have different concepts; this lack of agreement makes it easy to miss QRs. Moreover, it seems to us that the challenges do not look like being caused by Agile, but in fact it seems to be challenges that teams struggle with when trying to implement Agile. We make the note that due to space limitation, we did not include all the results. Our next step is therefore to continue with our analysis further. This includes to understand the role of the distributed context in the ways in which the challenges are experienced. Second, we plan to categorize the possible causes for each problematic phenomenon that we presented in Sect. 4. This would lead us to a set of hypotheses that we could evaluate in future research. This analysis is needed in order to better diagnose the problems of QRs in Agile. Only then we could propose a treatment for these problems, namely finding mitigations for the challenges.

VIII. REFERENCES

[1] M. Kassab, "An Empirical Study on the Requirements Engineering Practices for Agile Software Development," in SEAA, 2014, pp. 254–261.

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

[3] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," Inf. Syst. J., 20 (5), pp. 449–480, 2010.

[4] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," in WET ICE, 2003, pp. 1–6.

[5] I. Inayat, L. Moraes, M. Daneva, and S. S. Salim, "A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed," in XP Workshops, 2015.

[6] W. Alsaqaf, M. Daneva, and R. Wieringa, "Quality Requirements in Large-Scale Distributed Agile Projects – A Systematic Literature Review," in REFSQ2017, 2017, vol. 10153, pp. 219–234.

[7] C. Larman and B. Vodde, Large-scale Scrum more with less. Pearson Education, 2016.

[8] E. Bjarnason, K. Wnuk, and B. Regnell, "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," AREW, pp. 1–5, 2011.

[9] M. Käpyaho and M. Kauppinen, "Agile Requirements Engineering with Prototyping: A Case Study," in RE2015, 2015, vol. 23, pp. 334–343.

[10] R. K. Yin, Case Study Research Design and Methods, 5th Revise. Sage Publications Inc, 2013.

[11] C. Boyce and P. Neale, "Conducting in-depth interviews: A Guide for designing and conducting in-depth interviews," Evaluation, 2(5), pp. 1–16, 2006.

[12] K. Schwaber and J. Sutherland, "The Scrum Guide," Scrum.Org. p. 17, 2016.

[13] 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.

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

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

[16] K. Charmaz, Constructing grounded theory: a practical guide through qualitative analysis, vol. 10. 2006.

[17] D. Hitchcock, "The Practice of Argumentative Discussion," Argumentation, 16 (3), pp. 287–298, 2002.

[18] P. Bourque and R. E. D. Fairley, Guide to the Software Engineering Body of Knowledge, Version 3.0, vol. 3.0. 2014.

[19] R. Kazman and L. Bass, "Toward Deriving Software Architectures From Quality Attributes," Softw. Eng. Inst., no. August, pp. 1–44, 1994.

[20] B. Boehm, "Get ready for agile methods, with care," Computer, 35(1), pp. 64–69, 2002.

[21] I. Sommerville, Software Engineering. Pearson Education, 2011.

[22] P. Rodríguez, A. Yagüe, P. P. Alarcón, and J. Garbajosa, "Some findings concerning requirements in agile methodologies," in PROFES, 2009, vol. 32 LNBIP, pp. 171–184.

[23] C. Mario, Executable Specifications with Scrum. Pearson Education, 2013.

[24] 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," in CibSE, 2015, pp. 460–473.

[25] E. Bjarnason, K. Wnuk, and B. Regnell, "Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering," Inf. Softw. Technol., 54 (10), pp. 1107–1124, 2012.

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