From Scrum to Agile: a Journey to Tackle the Challenges of Distributed Development in an Agile Team

Pernille Lous
IT University of Copenhagen
Copenhagen, Denmark
pelo@itu.dk

Paolo Tell
IT University of Copenhagen
Copenhagen, Denmark
pate@itu.dk

Christian Bo Michelsen
IT University of Copenhagen
Copenhagen, Denmark
chmi@itu.dk

Yvonne Dittrich
IT University of Copenhagen
Copenhagen, Denmark
ydi@itu.dk

Allan Ebdrup
Debitoor
Copenhagen, Denmark
aeb@debitoor.com

ABSTRACT

Background: Agile and distributed software development are two trends that continue to increase rapidly in today's software industry. Even though the benefits achievable by combining them are potentially many, the intrinsic challenges of such marriage often lead to severe complications that can jeopardize the successful completion of software projects. **Method:** To investigate empirically how these two trends can coexist without compromising on the agile core values and principles, we conducted an exploratory holistic case study. Focusing on the development team of a Danish SME having both distributed offices as well as teleworking arrangements, we showcase (the evolution of) their practices. **Results:** The case is an example of the effective application of the agile reflective culture that allowed the company to evolve to a level in which the collocation restrictions of agile software development are overcome by a continuously evolving software process geared towards reducing waste to achieve speed and simplicity. **Conclusions:** Even though results need to be considered carefully due to the single nature of the reported case, we highlight five elements that have been fundamental in such journey: agile servant-leader, agile team, trust, virtual work environment, inspect & adapt, and reduce waste. Extensive information is provided to frame the context and to allow meaningful future comparisons.

CCS CONCEPTS

• **Software and its engineering** → **Agile software development**; **Programming teams**;

KEYWORDS

case study; agile software development; distributed software engineering; software process improvement; continuous deployment; continuous software engineering

ACM Reference Format:
Pernille Lous, Paolo Tell, Christian Bo Michelsen, Yvonne Dittrich, and Allan Ebdrup. 2018. From Scrum to Agile: a Journey to Tackle the Challenges of Distributed Development in an Agile Team. In *ICSSP '18: International Conference on the Software and Systems Process 2018 (ICSSP '18), May 26–27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3202710.3203149

ICSSP '18, May 26–27, 2018, Gothenburg, Sweden

1 INTRODUCTION

Agile software development and distributed software development are two trends that are steadily increasing in popularity [33]. The more dynamic approach to rapidly changing markets and the ability to access a larger pool of skilled personnel—in some cases entire teams—outside of political boundaries are just some of the advantages that these two trends promise to deliver. This marriage, however, does not come without costs. Often businesses fail to effectively embrace them by not carefully addressing the challenges that this combination entails (e.g., [16, 31, 35]).

The root causes of these challenges are the opposite conditions required by the two approaches. Arguably, the main tenet of the agile methodology is the focus on communication among the people involved in the development of a software system. As the manifesto for agile software development states in the list of principles: "[t]he most efficient and effective method of conveying information to and within a development team is face-to-face conversation" [3]. The idiosyncratic absence of physical presence in distributed development, therefore, clearly hinders the ability of a development team to completely adhere to the principles defining agile software development. Therefore, the question arises of whether it is possible to combine these two approaches while reaping their benefits.

The case reported in this paper describes an example of a successful marriage, in which a distributed development team implements a continuously evolving agile process by persistently focusing on the core principles of agile software development and the lean principle of reducing waste.

Case Subject. Debitoor has 40 employees, and its business is based on an online invoicing and accounting software solution for small businesses and companies. Fourteen employees, which are distributed across four sites in three European countries, are directly involved in the development of the product and represent our case subject. Debitoor has reached a maturity level that allows the members of the development team to work from any location, since every practice is performed in a virtual environment. Additionally, the reflective company culture ensures that all practices

are continuously evaluated in pursue of a simpler process and an overall better work environment.

Research Objective and Contributions. Based on a year-long collaboration, we aim to present the journey that Debitoor has undertaken in the past five years. By detailing the context and the motivations driving the evolution of the practices, we present new insights and possible approaches to address the challenges of being agile in a distributed arrangement. Therefore, the study is driven by the following research question: *"How can a development team adopt an agile process without compromising on the core values and principles when facing the challenges of distributed development?"*

Outline. The remainder of the paper is structured as follows: Section 2 further presents the related work in the context of (global) distributed software engineering and agile software development; Section 3 contextualizes the study subject and details the design of the case study; and, Section 4 describes the evolution of the practices by decomposing the journey into phases; and, in Section 5, we discuss the results in light the themes that emerged from the study. Before concluding with Section 7, we discuss the threats to the validity of the study in Section 6.

2 RELATED WORK

Global distributed software engineering

Since the late 1980s, business agreements with foreign companies such as merges, acquisitions, and alliances have been often executed [5, 10]. These practices were the forerunner of what nowadays evolved into a well established software engineering approach known as Global Software Engineering (GSE).

GSE promises several benefits, such as, the ability to recruit talent potentially not available in-situ, reducing time to market, increasing quality and operational time, and reaching proximity to market, to name a few [9]. However, being the development of software a complex endeavour that requires intense human cooperation, the distribution of software engineering teams across multiple sites, time-zones, languages, and cultures [29] poses challenges whose interdependencies are (too) often underestimated [9, 29].

Even though some researchers prefer to focus on specific dimensions of cooperation (e.g., coordination [13]), the canonical framework used to categorize the elements negatively affected by the distances of GSE comprises communication, collaboration, coordination, and awareness[1] [11]. These dimensions represent an extremely useful model that is used often to frame challenges and solutions in terms of processes (e.g., [21, 26]), tools (e.g., [34, 39]), and architectural considerations (e.g, [14]).

Agile software development

Most likely the biggest change that the software development industry has seen in the last decades has been the birth of agile software development, which emerged as a reaction to what is often referred to as structured, plan-based, or traditional software engineering. This transformation, which culminated in the writing of the manifesto for agile software development in 2001 [3], defined the beginning of a shift in the development industry.

[1]In [12], Fuks expanded Ellis's model [11] to include awareness in what nowadays is accepted as the 3C collaboration model.

The manifesto for agile software development [3] is a list of four values and twelve principles that describe the philosophy behind the methodology and, through its construction, clearly positions itself in contrast with prior methods: "while there is value in the items on the right, we value the items on the left more". Agile methods are characterised by a clear attention to the dynamics of team work, the strive for working software, the involvement of the customer, and the ability to react to an ever changing environment.

Even though, at least initially, management feared such revolutionary approach, as soon as results and benefits became more obvious, agile methods have been adopted by companies of all sizes and involved in product targeting a variety of application domains—including more regulated ones [21]. Among the agile methods, Scrum, XP, and their combination are by far the ones used the most [41]. In fact, they are extremely easy to combine due to their different approach to implementing the agile values and principles: the former provides management practices (i.e., a set of ceremonies, artifacts, and roles), while the latter focuses on programming practices (e.g., pair programming, TDD). Interestingly, in the recent years, also Kanban has been adopted widely, while XP seems to have lost momentum [41]. However, as shown in [21], hybrid methods are increasingly becoming the practice, and a combination of Scrum and Kanban (or Scrumban) using XP practices is quite common.

Distributed agile software development

Despite the distributed software engineering idiosyncratic characteristic of being physically distributed and the numerous values and principle explicitly requiring co-location, agile methods have been applied also in distributed arrangements (e.g., [2, 19, 23]). Researchers looked at whether the two approaches could be combined (e.g., [35]) and, interestingly, in some cases it was found that the use of agile methods can help reduce partially the issues related to the distances in distributed environment (e.g., [4, 18]). However, the combination of the two trends is all but a solved problem, and articles are constantly reporting challenges that hinder the use of agile methods in globally distributed teams (see, e.g., [6, 23, 25, 41]). More recently, strategies to overcome such issues have been described and suggested. These often result in discussing virtual teams, refinement of roles within teams, presenting technical solutions, or even presenting higher levels of integration of management, e.g., using Scrum-of-Scrums or complex agile frameworks like Large-scale Scrum (LeSS), Scaled Agile Framework (SAFe), or Disciplined Agile Delivery (DAD) [23]. Unfortunately, the majority of these approaches appear not to be in line with the agile values and principles as they tend to introduce heavier management practices; and, as the CTO of Debitoor stated when discussing SAFe during an interview, *"at every point in which you would expect a feedback loop, instead there is a person in charge—CTO."*

Even though it is recognized also by Cockburn [7], among others, that distribution makes agile development harder, this fallback strategy towards heavier practices should not be—we claim—the first and only solution, as it is in direct contrast with the initial motivations behind agile software development of speed and simplicity [15, 27]. The study at hand contributes to the body of knowledge by providing experiences based on the journey that a Danish SME

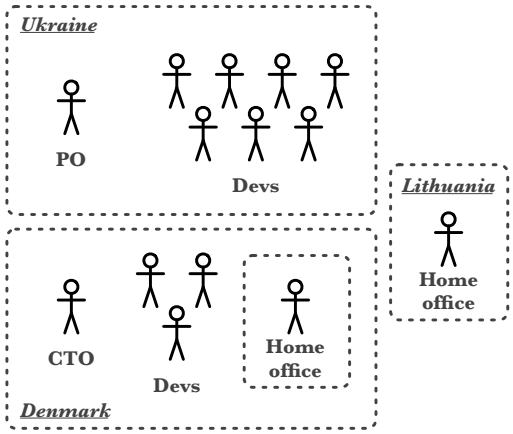

Figure 1: Development team overview.

Table 1: Characterization of the empirical context.

Attribute	Value
Year	2016-2017
Empirical focus	Empirically based
Empirical background	Industry
Industry sector	Accounting
Subject of investigation	Practitioners
Study results	Successful practices
Empirical research method	Case study
Source of empirical evidence	Observations, interviews
Location	Offshore
Legal entity	Outsourcing (insourcing)[2]
Geographic distance	Distant (within Europe)
Temporal distance	Small (1 hour max)
# of sites	4 (2x Denmark, Ukraine, Lithuania)
Team size	14 (12 developers, 1 PO, 1 CTO)

underwent to successfully adopt the agile methodology in its distributed arrangement.

3 RESEARCH DESIGN

The research objective, i.e., to investigate the use of agile practices in a company having both distributed offices as well as teleworking arrangements, has been investigated via an exploratory holistic case study [42] making use of both qualitative and quantitative techniques.

3.1 Organizational Context

The case organisation, Debitoor, is a small Danish company created in 2012, which headquarter is located in Copenhagen (Denmark). Debitoor provides an online invoicing and accounting software solution for small businesses and companies. Users of Debitoor's system are located in 33 European countries. Debitoor consists of forty employees in total with multiple national backgrounds. Between 2012 and 2015, before becoming an independent enterprise, Debitoor was part of a larger Danish company and was already allocated to the design and development of the current product. The results presented in this study will include events that occurred since 2012, and are based on the collaboration with Debitoor that started in 2016.

Results and insights presented in this study are based solely on the analysis of the single development team. Figure 1 provides an overview of the development team, which consists of fourteen people in total: five located in Denmark, eight in Ukraine, and one in Lithuania. The product owner (PO) is located in Ukraine and travels regularly between Denmark and Ukraine. Located in Denmark, the CTO is considered to be a part of the development team as his daily work is heavily in service of the development team. Interestingly, one developer in Denmark works remotely from his home office and rarely visits the office in Denmark due to perfume allergy, hence, considered as an independent site. In terms of expertize, all but one member, which is an intern, are senior developers and have years of experience working in distributed agile teams.

Table 1 presents a summary to characterize the empirical context to ease future comparisons. The taxonomy has been adapted from the one suggested in [40].

3.2 Data Collection

The data collection took place between November 2016 and October 2017 during twelve visits to Debitoor. During the study, we were given access to the entire tool ecosystem used by the development team; this included: Slack, Google Sheet/Slide, GitHub, and the Waffle board. The data collection techniques used during the visits included semi-structured interviews and observations (see Figure 2 for a detailed breakdown of the activities performed during the visits) [36]. Details about these techniques follow.

Observations. All nine observations were conducted at the Danish site. Six of them were full-day observations. During these visits, we observed the 9:15 stand-up meetings, the retrospective meetings in two instances, and participated to the daily work life of the team; this included joining the team during lunches and coffee breaks, which allowed us to ask questions to clarify specific observations captured throughout the day while the memory of the event was still fresh, but without disrupting the team's activities. Three visits, which lasted less then 30 minutes, included short observations of only the stand-up meetings. Field notes were taken during all observations and were re-written after the observation ended in line with the process described in [28].

Interviews. Besides discussing general aspects of the work practices at Debitoor, all these interviews—apart from the first, which was the project kick-off—were all organized with key members of the development team to explore recurring insights captured during the observations and explore the process that led to the practices observed. Semi-structured interviews were chosen to ensure the openness of the conversation on the one hand, while on the other allowing the topics of interest to be fully explored [37]. In total, seven semi-structured interviews were conducted. One with the PO, one with a developer from Ukraine, and five with the

[2]Note: Debitoor's business arrangement with the Ukraine part of the team is outsourcing. However, all members of the development team consider the relation much more in line with what would be expected in an insourcing arrangement.

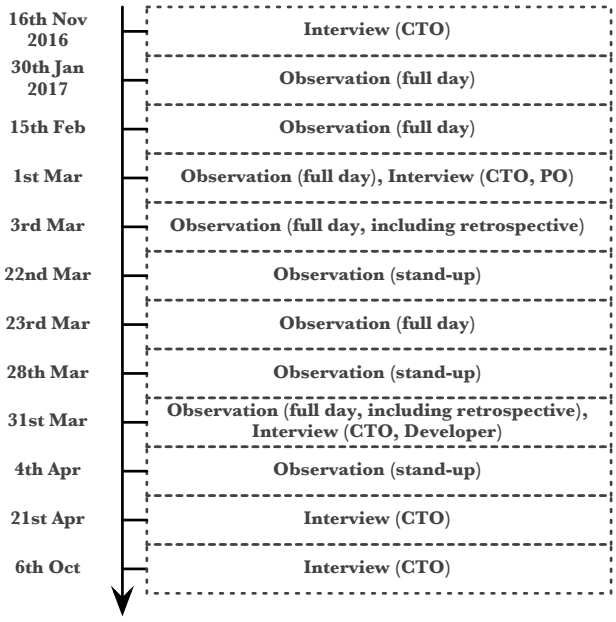

Figure 2: Overview of the research activities performed at Debitoor.

CTO. All interviews were conducted using an interview guide [42], lasted on average roughly one hour, and were recorded for post analysis. All but one interview, which was performed through a Slack-call (Ukraine developer), were conducted face-to-face in a meeting room.

Access to the tool ecosystem of Debitoor. Throughout the study, we had access to the entire tool ecosystem used by the development team. This included access to: Slack—their communication hub; Google Slides—used to support various meetings; Google Sheet—their knowledge repository and collaboration tool during the retrospective meetings; Waffle.io—hosting their task board; GitHub; as well as the slides used by the PO during the grooming sessions.

Off-site Data Collection. Finally, throughout the study, the main contact at Debitoor was available via email and Slack (while on-site), and this connection was leveraged several times for clarification as suggested in [28].

3.3 Data Analysis

Audio recordings from the interviews and selected observation parts were transcribed manually by two researchers. Together with the field notes, all material was coded using Atlas.ti for data analysis. The data underwent several coding iteration performed by two researchers along the study. The coding eventually stabilized around clusters, which were then abstracted into themes (see Section 5).

4 RESULTS

Figure 3 provides an overview of the major process changes that Debitoor underwent over the last years. After describing the method previously used, one close to Scrum, we will present the set of practices that are currently used by Debitoor and their evolution over time focusing, when relevant, on the motivation that triggered specific modifications.

Previous use of Scrum (Figure 3-A). Since the beginning, Debitoor was practising distributed agile, for the Ukraine members were already part of the team. The method followed—Scrum—was adopted due to company-wide standards. Of interest in this context is that the team used GitHub to version the code base and Jira to manage tasks and backlog. The process included quality assurance (QA) in the form of mandatory formal code reviews and a dedicated QA team. Additionally, continuous integration was a practice deeply rooted in the team, hence, team members were familiar with unit testing and code quality in terms of the application of company wide standards (e.g., variables naming, methods length, etc.). The sprints length was two weeks, and, in terms of Scrum ceremonies, Debitoor did not have sprint reviews at this stage, but was practising daily stand-up meetings, sprint retrospectives, and sprint planning meetings. Tasks size was estimated using planning poker involving the entire team. The team comprised of one team leader (TL), two POs, and a CTO that was not very involved in the daily work of the team.

New team leaders (Figure 3-B). Two new team leaders got assigned in Debitoor: a primary in Denmark (chief TL) and a secondary in Ukraine (Ukraine TL). The previous TL got reassigned, and, later, one PO got dropped. This was a crucial point in the history of Debitoor, after which all the process improvement activities were designed and, most importantly, experimented and implemented fearlessly. Even though full independence arrived only after the merge operation of the mother company, Debitoor's team was autonomous already at this point, and the chief TL was fully trusted by management—so much so that he later became the CTO of Debitoor. Three major changes in the process were experimented at this stage: regular one-on-ones, the pull request approach that lead to continuous deployment, and new stand-up meetings. These will be thoroughly explained in the following relevant sections.

One-on-ones (Figure 3-C). One-on-ones were introduced as a regularly practice for the development team. These are personal talks between the chief TL and a team member. The conversations can cover anything from topics related to the company itself to private issues in the employee's private life. Their main purpose is to ensure the employee's well-being and satisfaction. They last approximately thirty minutes, and there is an extra half an hour that is scheduled as a buffer in case needed. Their frequency is not a constant, and it might vary between one week and three weeks depending on whether there is a pressing necessity from individual employees to discuss specific topics. For a long period, this practice was performed by the team leader in Denmark and involved all team members—including the ones distributed. In the case of Danish employees, such conversation would happen either in a meeting room or during a walk outside of Debitoor's premises. While, with the remote team members, the meeting would be performed over a Slack call. As the CTO admits: *"[...] it is much easier to do it in person. It is sometimes a bit hard with the Kiev-guys. I mean you have the video feed and you can sit and talk, and you can get quite intimate and trusting each other—if you are honest and open about*

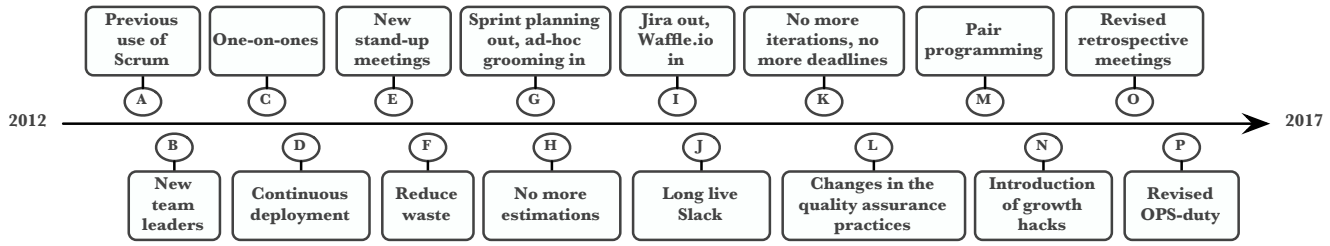

Figure 3: Overview of the changes that Debitoor underwent in their software development process between 2012 and 2017.

things. But it is just—you know—a bit better when you are there [in person], but unfortunately that is not an option—CTO." During the last visit to Debitoor, we were informed that the responsibility of performing one-on-ones with the Ukraine employees is now given to the Ukraine local team leader.

Continuous deployment (Figure 3-D). At the same time that the two TLs got promoted in the team, one of the developers started experimenting with pull requests on GitHub. The approach triggered a strong interest in the two TLs. As the CTO states when remembering the events: *"this is going to change everything—CTO."* The decision was therefore taken of moving towards continuous deployment. This was motivated by the wish of the TLs to achieve a more parallelized and faster implementation of tasks to production. The idea had already been discussed previously, but was never implemented due to more pressing matters constantly being prioritized.

As mentioned before, the team was practising continuous integration, and the missing part was the automation of the deployment pipeline. Within the following two weeks, the team managed to move to continuous deployment. Yes, there were still a few hiccups and some tasks had to be performed manually, but the infrastructure was there and tasks could be deployed independently. Debitoor was already at this point able to deploy more than ten times a day.

New stand-up meetings (Figure 3-E). This ceremony is most likely the one that has been modified the most. Already at the beginning, once the continuous deployment pipeline was operational, the chief TL pushed the development team to design a new format for the stand-up meetings. Besides following the regular three questions (i.e., what I did, what I will do, and what impedes me), the meeting was hosted using 'Join-me'-screen sharing, in which everyone looked at the same Jira-board with each developer sitting at their own personal computer with their own headset. The rational that motivated such request was that the previous (standard) format was repetitive, boring, and did not engage people. The chief TL decided to begin an experimentation period to change the practice in which groups of two people would take turns for a sprint: *"two guys. You get the responsibility of coming up with a new format for the stand-up and the rule is that everybody has to participate with a maximum of 15 minutes. It has to be fun—CTO."* This was the setup, and such were the simple rules. At the very first experiment in the first week, the couple in charge lead the stand-up meeting using Google slides *"We ran the experiment for quite a while, and we tried some different things [within this format] and then in the end: "Lets stop experimenting". Then, we voted on the stuff we tried and came up with what we are doing now—CTO."*

The format that emerged from this experimentation is rather peculiar. *"So, basically this whole "stand-up as a service"-thing is something we build [...] where there are process issues you normally could just go and talk to people or do a whiteboard or something. We have tried to solved it in this way by automating some of this stuff [Google slides] and putting it on slides and then we call it "Stand-up As A Service"—CTO."* The meeting was still conducted with each developer sitting at their own personal computer wearing a headset. The three standard questions were dropped, and instead Google Slides were used as the main tool to drive the meetings. The duration of the meetings was carefully tuned to be always less then 15 minutes. The initial slide was automatically generated, e.g., using calendars data to make people aware of vacation periods, or data from GitHub. In case of recurring error messages in the production logs, another slide would be generated to alert the team about a potentially critical issue in the system. Finally, developers could add a personal slide to present interesting aspects of currently active tasks for knowledge sharing purposes: *"[...] so one of the principles in Debitoor is that we are not enforcing people a lot of things, right, they should share things by themselves—Ukraine developer."*

Reduce waste (Figure 3-F). Inspired by the lean principle of reducing waste and to ensure that team members could feel as productive as possible, rather then overwhelmed by constant meetings, the chief TL re-evaluated every practice that would require team members to spend time on potentially unproductive activities. As he often states: *"when I started as team lead, I went on the mission to kill all those meetings. We had so many meetings, and we simply got rid of those."* Today, meetings are held when necessary and mainly when employees request them. As one stated: *"It is more when we need it [that we are doing meetings]—Danish developer."* According to the CTO metrics, *"developers have about 3% of their time allocated to meetings, so that is 300 minutes every month—CTO."*

The repercussion of this continuous strive, which is still one of the core drivers of the CTO's modus operandi, will be discussed within the context of the next practices.

Sprint planning out, ad-hoc grooming in (Figure 3-G). As one of the first process improvement activities geared towards reducing waste, the TLs decided to remove the lengthy sprint planning meeting. They realized that the content of these meetings was not of interest to the entire team, and, given their duration, planning meetings represented a waste that had to be reduced. *"We do not do team*

grooming at all. There is no reason for everyone to groom every task. I think it is a huge waste of time—CTO."

While taking such decision, the TLs discussed the implications of such drastic change, e.g., reduced knowledge sharing and awareness among team members. Nonetheless, with an experimental mindset, they also realized that the worst that could happen would be to be forced to re-introduce them—eventually this did not happen. Instead of the planning meetings, they designed a process in which the PO would groom specific prioritized tasks together with either single individuals or sub-sets of the team that would have worked on them.

No more estimations (Figure 3-H). Another important practice that was removed together with the elimination of planning meetings was task estimations. The planning poker technique was too lengthy, and tasks were already groomed together with the team members that would have implemented them to something small enough that could be performed within a day. Since the people involved in the grooming were the ones that, on the one hand, had the necessary expertise to assess the tasks and, on the other hand, had to complete them eventually, the need for (accurate) estimations diminished and in the end were simply dropped.

Jira out, Waffle.io in (Figure 3-I). After having removed estimation and burndown charts, as they could not be computed anymore, it was decided to try to replace the project management tool. Jira, the tool used since 2012, was too heavy and required too much attention to configure to fit the needs of the team. Therefore, after evaluating several alternatives, Waffle.io—an automated project management tool for GitHub—was chosen to be used as a virtual task board. This event coincided with the removal of Scrum iteration in favour of a lean process closer to Kanban (see later). At this point, the number of columns on the task board decreased from seven to only three: 'the next awesome thing', 'in progress', and 'in production'.

Long live Slack (Figure 3-J). To reduce the set of tools used for communication, Slack was integrated in the tool ecosystem. All kind of communication at Debitoor happens on Slack. This includes synchronous and asynchronous as well as textual, audio, video, and screen sharing. After carefully designing the channels and educating all employees at Debitoor, Slack had become the one and only tool used to communicate internally. This change included the complete removal of email for internal communication.

No more iterations, no more deadlines (Figure 3-K). "We were doing 14 days sprints and having a lot of pain, [as] when the sprint finishes you have to have everything in the done column before you can deploy. And one period, we experienced a delay of three weeks, in which people were starting the next sprint. And, if you are three weeks delayed then it is also the next sprint that gets delayed. It was just awful, the entire coordination to do that—CTO." The pressure on the team was simply not worth, and the TLs decided to remove iterations. With the introduction of continuous deployment, in fact, adhering to a pre-defined pace was not anymore necessary. At the same time and for practically the same reasons, the TLs decided to shield the development team from the pressure imposed by the use of deadlines. It was recognized that deadlines were demotivating, stressful, and unnecessary for the developers. Customers and partners of Debitoor do not receive specific dates for the delivery of features; the only exception are major marketing campaigns in which larger financial consequences are at stakes, and these are necessary to synchronize PO and CTO with the marketing team. However, also for these occasions, deadlines are not communicated to the development team and the responsibility to meet them falls entirely on the CTO.

Changes in the quality assurance practices (Figure 3-L). After having fully implemented continuous deployment, the quality assurance (QA) team quickly became a bottleneck to the continuous flow of tasks to production. Therefore, the QA team and the mandatory code reviews were removed: *"it turns out it was way more effective when you need to find someone to review your work. It was super super important that we did not make these swim lanes where things would end up stuck with each hand-over—CTO."*

Together with this change, it was made clear that the responsibility of a task would have lied with the developer(s) that owned the tasks. In other words, a significant amount of trust was given to the judgement of developer(s), which had to decide whether a specific piece of code required a more structured review. *"[O]nce they are done, it is up to the developers to do code review and ask for code reviews and do automated testing and do whatever it take to make sure that this thing [the code] is production quality. We do not have any testers or QAs so they [the developers] make sure that the quality is great—CTO."* It is also important to note that at this point in the history of Debitoor everything that could be automated was either already automated or in the process of being automated. This includes: static code checking, verification of code styles, and alike.

In terms of code reviews, these are arranged as follows. Nowadays, developers are encouraged to seek a reviewer through Slack. And, an unwritten rule exists that ensures that reviews are performed within roughly 15 minutes. Because these are not mandatory, it is well understood by all team members that, if a review is requested, it is important. It should be noted that, according to Debitoor's internal culture, interrupting a colleague is not an acceptable behaviour; and, code reviews are the only exception to this internal code of conduct. During the observations, we saw no predominance of choosing co-located team members over remote ones when selecting the reviewer. As the CTO confirmed, the selection criteria is purely based on complexity and expertize of a specific area of the code base. *"There might be a few people who prefer certain people for code review, but I mean that is people. That is not Copenhagen/Kiev [Denmark/Ukraine]. That is a personal thing—CTO."* Finally, contrarily to expectations, it is important to mention that the number of bugs reported by users with this new QA process went down drastically: *"we reduced the amount of bugs by 44%—CTO."*

Pair programming (Figure 3-M). Among the QA techniques, pair programming is certainly a proactive one. This was performed from the very beginning, and still is performed among team members at Debitoor regardless of physical location. If the practise is performed across sites, screen sharing and Slack calls are used. Especially if the team members are working on closely related tasks, pair programming happens frequently. However, following the self-organizing agile principle, the decision of whether to do it or not is left entirely to the developers, which rely on this technique to

ensure quality when facing a more challenging task. *"Some people like to do pair programming, some people like to work alone, some people like to do pair programming a bit and then work alone—CTO."*

Introduction of growth hacks (Figure 3-N). Inspired by the continuously evolving practices experimented by the two TLs and comforted by the fact that proposals advanced at the retrospective meetings were taken seriously, some team members suggested the introduction of growth hacks: 20% of the employee's time dedicated to the development of personal projects related to the company product. This initiative was feared by the TLs at first, but, in line with the experimental culture they were advocating, they accepted. For a long period of time, Fridays became dedicated to personal projects. Afterwards, again triggered by developers during a retrospective meeting, growth hacks became two days with a bi-weekly cadence. Interestingly, some developers preferred initially to use such time to complete delayed tasks. However, after more or less a year and after the removal of sprint iterations and deadlines, all developers nowadays use this time for developing creative ideas, many of which become part of the backlog after being prototyped during the hacks and accepted for inclusion.

Revised retrospective meetings (Figure 3-O). Following the same setup used during stand-up meetings, all developers—even when co-located—sat at their personal computer wearing a headset and accessed the same Google Sheet. Lead by the developer in Lithuania, the meeting comprised five steps:

- assessing results based on the previous retrospective;
- sharing good and bad thoughts since the last meeting;
- generating ideas on how to solve the bad thoughts identified in the previous step;
- defining a plan on how to implement the ideas; and,
- (eventually) assigning tasks to developers if necessary to ensure their execution.

Initially, developers suggested to change the cadence of the retrospective from two to four weeks. Even though retrospective meetings had been successful for years, the CTO challenged the meeting format: *"I mean, they work pretty well and do great results, but I just felt to try something new after having done the exact same formula for some years. So we had hired an external facilitator [...] —CTO."* An external facilitator was hired to try to improve the meeting; but, after trying different formats, nothing suggested was to the team's liking. Eventually, the external facilitator was dropped after taking a vote in the development team, as the team was happy to revert to the old format.

Revised OPS-duty (Figure 3-P). OPS-duties at Debitoor are split among the team members from all sites. That is, no OPS-workers are employed in the company, and the duties are a rotating responsibility. Initially, OPS-duties switched daily to a new developer, but this was reported to be tiresome and frustrating for the developers. Instead, weekly duties were introduced in which developers were responsible for the duties for an entire week.

In conclusion, guided by the lean principle of reducing waste and following the agile philosophy, Debitoor clearly went through a significant journey supported by a culture of experimentation to continuously reflect and improve both their development process and their work environment: *"it is a culture itself, that we can change things as we want. We remove the annoying aspects without problems—PO."* These process improvement activities have resulted in the creation of a virtual work environment in which the challenges of distributed development are overcome. In the next section, we will abstract and further discuss these results.

5 DISCUSSION

In this section, we abstract the practices of Debitoor previously presented to answer the research question that drove this study. We will describe how Debitoor succeeds in adopting an agile process even when facing the challenges of distributed development without compromising on the core values and principles.

5.1 "How can a development team adopt an agile process without compromising on the core values and principles when facing the challenges of distributed development?"

Agile methods fundamentally rely on human interactions. However, in distributed software development, distance (i.e., geographical, temporal, cultural, and linguistic [1, 29]) makes such interactions challenging by putting pressure on one of the most basic principles of agile software development. A considerable share of the literature suggests strategies to overcome such issues. These often result in discussing virtual teams, refinement of roles within teams, presenting technical solutions, or even presenting higher levels of integration of management, e.g., using Scrum-of-Scrums or complex agile frameworks like Large-scale Scrum (LeSS), Scaled Agile Framework (SAFe), or Disciplined Agile Delivery (DAD) [23]. Unfortunately, the majority of these approaches appear not to be in line with the agile values and principles as they tend to introduce heavier management practices.

To perform continuous software process improvement activities, which lead to hybrid approaches that cannot be related to any specific method [21], is a healthy practice that should be encouraged. Ollson and Bosch [30] defined a research-based road-map describing five stages a company would naturally have to journey through to reach a maturity that is often referred to as Continuous Software Engineering (CSE): traditional development, R&D organization all agile, continuous integration, continuous deployment, and R&D as an innovation system. Debitoor already was in the third stage of this journey, and we presented the practices that allowed them to transition towards continuous deployment and, partially, R&D as an innovation system.

We argue, however, that to pursue these final stages without compromising on the speed and simplicity—goals of the agile philosophy—additional enabling elements need to be carefully considered. In the following, we introduce the themes that emerged from the empirical work at Debitoor (see Figure 4): *agile servant-leader* and *agile team* focus on the organization, *trust* and *virtual work environment* describe the environment, while *inspect & adapt* is the engine enabling a reflective culture pursuing the *reduce waste* goal to achieve speed and simplicity. We claim that these are important elements that should be considered to ensure healthy transitions into a distributed company following the agile core values and principles.

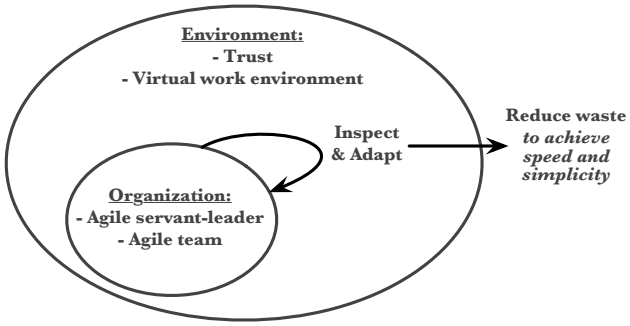

Figure 4: Schematic representation of the themes that support the continuous reflective culture at Debitoor.

Agile servant-leader. The CTO has been, and still is, instrumental at Debitoor for enabling a true culture change towards simplicity and speed. As the Scrum master in Scrum, he is the Agile master in Debitoor's processes. His role within the team has a long history: he started as a team member, became a team leader, and eventually the CTO. He is respected both within the team and by management for his technical and social abilities. Additionally, he has been practising agile software development and has seen its benefits for years. Since becoming team leader, this combination has allowed him to successfully implement changes without 'much' resistance from either side.

The agile servant-leader is a key figure within a team/organization that wants to climb the stairway to heaven [30]. Without a leader that has the management support changes can be proposed and discussed, but will hardly be implemented. This for instance was the case with continuous deployment at Debitoor(Figure 3-D): the topic had already been discussed, but had never been prioritized. Similarly, such key figure must be a servant to the development team to ensure that any impediments, any suggestion for improvement emerging from the team, or any little automation that can improve the working condition of the team is removed, heard, or implemented. Examples of this attitude are many in the results, but certainly the most visible and relevant are the constant one-on-ones. These meetings exemplify the continuous dedication to the well-being of the team.

Agile team. It should not come as a surprise that the development team is a crucial pillar in an agile software development organization. It must be self-organizing, states [3]. And, we argue, it is fundamental to make the combination of agile development and global software development succeed. The challenges with achieving a self-organised team can be many, e.g., differences in culture [32], lack of reflective behaviour [43], or lack of trust [18]. At Debitoor, we have observed how rooted this principle is. The members of the team are fully empowered to take decisions on how to perform a development task from the beginning till the end. Pair-programming, additional grooming, QA, and, of course, deployment are just few of the most visible techniques and practices that each team member can independently decide on. Three major factors were observed that allowed the self-organising team to flourish: easy access to information, high amount of responsibility placed upon the team members, and an undisputed level of trust. First, the virtual work environment established at Debitoor allows the team to easily get access to all information required, and the communication infrastructure supports all communication needs. Second, code reviews, grooming, pair programming, and the OPS-duty are just some of the examples showcasing how the development team has complete responsibility over the development activities. Lastly, and arguably most importantly, trust. Each team member is trusted to be able to pick a task and see it through completion.

Trust. In 1979, Luhmann already noted that "one should expect trust to be increasingly in demand as a means of enduring the complexity of the future which technology will generate" [24]. "Trust has become widely acknowledged as a crucial factor in inter-organizational relationships[, for it] affects a wide range of relationship qualities, from increasing relationship stability to lowering transaction costs for the trusting parties" [20]. Trust has been widely explored in (globally) distributed development [17, 38]. It is an extremely delicate secret ingredient to cooperation, which is crucial to work as friction-less, effective, and efficient as possible within a team.

At Debitoor, trust permeates the organization. Most notably, the CTO and the development team competences are fully trusted by each other, by the management, and by all other departments in the company. This allows them to work efficiently while feeling part of an organization that believes in their capabilities. And as Luhmann [24] foresaw, we observed how trust clearly improved the distributed cooperation, since the coordination that would be required to (micro) manage individuals was not needed.

Virtual work environment. A challenge often reported in case studies of companies combining agile development and global software engineering is the one of feeling united as one team (e.g., [18, 22, 35]) [23]. The feeling of being separated as multiple teams can have different motivations, and team unity is often reported to be a problem even when communication is supported by online channels such as e-mail, instant messages, calls, or virtual rooms [18]. Physical tools such as a task boards, post-its or burndown charts placed on-site are artifacts difficult, if not impossible, for the off-site team to follow and appreciate [16]. Additionally, physical meetings held on-site might create a feeling of being excluded [19]. When teams are not feeling united, important aspects of agile development such as trust and knowledge sharing become a problem; "the project information should be available with details to all members" [8]. A virtual team which is missing a shared understanding of tasks or goals can end up being an inefficient virtual team [20].

At Debitoor, it was decided that rather than working around distance, the development team would have embraced it to allow for a complete inclusion of all sites. As described in the previous section, all meetings happen online, while sitting at each own work station. The virtual work environment at Debitoor fosters the creation of a cohesive team culture, thereby increasing trust and integration within the team.

Inspect & Adapt. Inspect and adapt emphasises the reflective behaviour of the development team to improve the software process [3]. The engine that has been used in Debitoor to enable software process improvements has been mainly the retrospective and the

one-on-one meetings. These represent the core practices that enable the Debitoor's reflective culture to software processes. Triggered sometimes as a result of the one-on-one meetings, almost every change is discussed and democratically voted during the retrospective meetings. At Debitoor, we saw two clear triggers: a more top-down approach, in which the CTO would propose changes; and, a bottom-up one, in which the team would be the initiator. Both of these triggers have been important in the shaping of the current software process followed at Debitoor and, we argue, they represent a healthy balance in an agile team, which should be pursued.

To support this reflective culture, experimentation is encouraged at Debitoor to ground the decision making process on facts. This culture is essential to ensure that the outcome of the inspect and adapt activities can be tested, and—why not—if not achieving the sought outcome, reverted. In other words, to fail must be a possible outcome, and failure must be accepted since: on the one hand, the experience undergone provides knowledge; on the other hand, because of to the fast feedback cycles, failure is detected fast and recovery can be achieved quickly.

Reduce waste. Finally, reduce waste. Using the CTO's words: *"I went on the mission to kill all those meetings."* Since becoming a team leader, the CTO has been following the lean principle of reducing waste as a goal to guide the decision making process. As represented in Figure 4, this is the purpose that drives all the process improvement activities at Debitoor to eventually reach a work environment in which the development team thrives *"by being empowered to do its job without having to ask permission and by being happy. We tried to be a start-up, as effective as you would be by working in a group of two people on something than you can be super effective—CTO."*

To summarize, our empirical material hints at the importance of a few elements that need to be guaranteed to succeed in performing agile development while being distributed. First, a strong agile backbone (i.e., an agile servant-leader and an agile team). Second, team independence—guaranteed by relations based on trust; and, individual independence—guaranteed by a fully virtual work environment. Third, a culture of experimentation in which rapid feedback cycles allow failure to be a learning exercise rather than an issue. Fourth, a constant strive to achieve simplicity and speed by reducing waste.

All these together create a reflective culture in which the inspect and adapt agile principle can be applied constantly in a distributed agile team to perform process improvement activities that lead to speed and simplicity.

6 THREATS TO VALIDITY

In this section, we will discuss the threats to the validity of this study following the basic scheme that distinguishes among construct validity, internal validity, external validity, and reliability [42]. *Internal validity* will be skipped, for with this empirical study based on qualitative data we are not attempting to proving logical relationships among the concepts presented.

Construct validity. Threats to construct validity were minimized by using different sources of data including observations, semi-structured interviews, project artifacts, and off-site data collection. As suggested by [37] and [28], this triangulation process allowed us to clarify and confirm insights by accessing a wide range of perspectives.

Reliability. To ensure reliability [42] and to reduce the risk of single-researcher bias, all field activities, but three interviews, were performed by at least two researchers. With regards to the data analysis, this was initially performed by two researchers until the coding scheme stabilized and, afterwards, meetings were arranged including often up to four researchers to discuss, clarify, and proceed in the generation of clusters and themes. Additionally, an open communication with the company contact was kept to verify constantly the findings, hence, decreasing the risk of misinterpretation.

External validity. The main limitation of single case studies is their generalizability [42]. This study is no exception; however no statistical significance or generalizability of the results were sought. Our results and discussions are grounded on the case analyzed and are extremely affected by the specific context. We have therefore provided detailed information of the context to allow future studies to replicate or compare results.

7 CONCLUSIONS

Agile and (globally) distributed software development are two trends that are relentlessly growing in momentum. Unfortunately, due to their intrinsic opposing requirements, solutions to combine them tend to (too easily) compromise on the agile core values and principles by introducing heavier management practices to cope with the complexity introduced by geographical, temporal, cultural, and linguistic distances.

This paper reports on a case study of the software development team of a Danish SME that successfully adopts agile practices while being distributed across several locations. Through a thorough presentation of the practices followed, we showcase how a distributed team succeeded in dealing with the known problems of distributed development without compromising on the motivations behind agile software development: speed and simplicity. Via a continuous reflective introspection, the team has modified the software development process to a maturity that: (i) ensures adherence to the core agile philosophy; (ii) guarantees mechanisms to continuously improve the process both from the top (i.e., CTO) and from the bottom (i.e., the team members); and, (iii) allows team members to work from any location, since every practice is performed in a virtual work environment.

Our empirical material hints at the importance of a few key elements that have been fundamental in such journey: agile servant-leader, agile team, trust, virtual work environment, inspect & adapt, and reduce waste. In the future, we plan to validate these findings by widening the study scope to more organisations.

8 ACKNOWLEDGMENTS

The academic authors would like to thank Debitoor and, in particular, all the employees that actively participated to the study.

REFERENCES

[1] J Ågerfalk and Brian Fitzgerald. 2006. Flexible and distributed software processes: old petunias in new bowls. In *Communications of the ACM*. ACM.

[2] Yehia Ibrahim Alzoubi, Asif Qumer Gill, and Ahmed Al-Ani. 2016. Empirical studies of geographically distributed agile development communication challenges: A systematic review. *Information & Management* 53, 1 (2016), 22–37.

[3] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for agile software development. (2001). http://www.agilemanifesto.org/

[4] Sarah Beecham, John Noll, and Ita Richardson. 2014. Using agile practices to solve global software development problems–a case study. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE-Workshop)*. IEEE, 5–10.

[5] Erran Carmel and Ritu Agarwal. 2001. Tactical approaches for alleviating distance in global software development. *IEEE software* 18, 2 (2001), 22–29.

[6] Tsun Chow and Dac-Buu Cao. 2008. A survey study of critical success factors in agile software projects. *Journal of systems and software* 81, 6 (2008), 961–971.

[7] Alistair Cockburn. 2006. *Agile software development: the cooperative game*. Pearson Education.

[8] Daniela S Cruzes, Nils B Moe, and Tore Dybå. 2016. Communication between developers and testers in distributed continuous agile testing. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 59–68.

[9] Daniela Damian and Deependra Moitra. 2006. Guest editors' introduction: Global software development: How far have we come? *IEEE software* 23, 5 (2006), 17–19.

[10] Christof Ebert and Philip De Neve. 2001. Surviving global software development. *IEEE software* 18, 2 (2001), 62–69.

[11] Clarence A Ellis, Simon J Gibbs, and Gail Rein. 1991. Groupware: some issues and experiences. *Commun. ACM* 34, 1 (1991), 39–58.

[12] H. Fuks, A. Raposo, M.A. Gerosa, M. Pimentel, and C.J.P. Lucena. 2007. The 3c collaboration model. *The Encyclopedia of E-Collaboration, Ned Kock (org)* (2007), 637–644.

[13] James D Herbsleb. 2007. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*. IEEE Computer Society, 188–198.

[14] James D Herbsleb and Rebecca E Grinter. 1999. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*. ACM, 85–95.

[15] Jim Highsmith and Alistair Cockburn. 2001. Agile software development: The business of innovation. *Computer* 34, 9 (2001), 120–127.

[16] Rashina Hoda, Norsaremah Salleh, John Grundy, and Hui Mien Tee. 2017. Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology* 85 (2017), 60–70.

[17] Helena Holmström, Eoin Ó Conchúir, J Agerfalk, and Brian Fitzgerald. 2006. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 3–11.

[18] Helena Holmström, Brian Fitzgerald, Pär J Ågerfalk, and Eoin Ó Conchúir. 2006. Agile practices reduce distance in global software development. *Information systems management* 23, 3 (2006), 7–18.

[19] Samireh Jalali and Claes Wohlin. 2010. Agile practices in global software engineering-A systematic map. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 45–54.

[20] Frens Kroeger and Reinhard Bachmann. 2013. Trusting across boundaries. (2013).

[21] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, et al. 2017. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In *Proceedings of the ACM International Conference on Software and System Process (ICSSP)*. ACM, 30–39.

[22] Lucas Layman, Laurie Williams, Daniela Damian, and Hynek Bures. 2006. Essential communication practices for Extreme Programming in a global software development team. *Information and software technology* 48, 9 (2006), 781–794.

[23] Pernille Lous, Marco Kuhrmann, and Paolo Tell. 2017. Is Scrum fit for global software engineering?. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE Press, 1–10.

[24] Niklas Luhmann. 1979. Trust and power. *John Willey & Sons* (1979).

[25] Santiago Matalonga, Martín Solari, and Gerardo Matturro. 2013. Factors affecting distributed agile projects: a systematic review. *International Journal of Software Engineering and Knowledge Engineering* 23, 09 (2013), 1289–1301.

[26] Martha L Maznevski and Katherine M Chudoba. 2000. Bridging space over time: Global virtual team dynamics and effectiveness. *Organization science* 11, 5 (2000), 473–492.

[27] Renée McCauley. 2001. Agile development methods poised to upset status quo. *ACM SIGCSE Bulletin* 33, 4 (2001), 14–15.

[28] M.B. Miles, A.M. Huberman, and J. Saldaña. 2013. Qualitative Data Analysis. SAGE Publications.

[29] John Noll, Sarah Beecham, and Ita Richardson. 2011. Global Software Development and Collaboration: Barriers and Solutions. *ACM Inroads* 1, 3 (2011), 66–78. https://doi.org/10.1145/1835428.1835445

[30] Helena Holmström Olsson and Jan Bosch. 2014. Climbing the "Stairway to Heaven": evolving from agile development to continuous deployment of software. In *Continuous software engineering*. Springer, 15–27.

[31] Maria Paasivaara and Casper Lassenius. 2006. Could global software development benefit from agile methods?. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 109–113.

[32] Maria Paasivaara, Casper Lassenius, Ville T Heikkilä, Kim Dikert, and Christian Engblom. 2013. Integrating global sites into the lean and agile transformation at ericsson. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 134–143.

[33] Efi Papatheocharous and Andreas S Andreou. 2013. Evidence of agile adoption in software organizations: An empirical survey. In *European Conference on Software Process Improvement*. Springer, 237–246.

[34] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. 2012. Tools used in Global Software Engineering: A systematic mapping review. *Information and Software Technology* 54, 7 (2012), 663–685.

[35] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu. 2006. Can distributed software development be agile? *Commun. ACM* 49, 10 (2006), 41–46.

[36] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

[37] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

[38] Darja Šmite, Claes Wohlin, Tony Gorschek, and Robert Feldt. 2010. Empirical evidence in global software engineering: a systematic review. *Empirical software engineering* 15, 1 (2010), 91–118.

[39] Igor Steinmacher, Ana Chaves, and Marco Gerosa. 2010. Awareness support in global software development: a systematic review based on the 3C collaboration model. *Collaboration and Technology* (2010), 185–201.

[40] A. R. D. R. Techio, R. Prikladnicki, and S. Marczak. 2015. Reporting Empirical Evidence in Distributed Software Development: An Extended Taxonomy. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. 71–80.

[41] Version One. 2016. The 10th Annual State of Agile Report. (2016). http://stateofagile.versionone.com

[42] Robert K Yin. 2013. *Case study research: Design and methods*. Sage publications.

[43] Franz Zieris and Stephan Salinger. 2013. Doing scrum rather than being agile: A case study on actual nearshoring practices. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 144–153.