Doing Scrum Rather Than Being Agile: A Case Study on Actual Nearshoring Practices

Franz Zieris, Stephan Salinger
Freie Universität Berlin
Institut für Informatik
Takustr. 9, 14195 Berlin, Germany
Email: zieris|salinger@inf.fu-berlin.de

***Abstract*—Previous research in the field of Agile Distributed Software Development often focused on the asynchronicity of working hours due to different time zones, as if this was the only risk when developing software in non-co-located environments. This case study reflects a nearshoring setting in which this primary impediment does not exist and investigates a broader range of risks now standing out more clearly. We observed two Polish Scrum teams working for a German company, which has been successfully applying Agile Methods for over four years. We present the actual process and practices of the external teams and contrast them to the intended way of proceeding.**

Main result: Agile nearshoring is feasible and may produce high satisfaction amongst Product Owners, but this satisfaction might be delusive if process deviations due to misunderstandings of what Agile development means go unnoticed.

Keywords*—*agile process, agile mindset, roles, practices, nearshoring

I. Introduction

The conceptual pair "Agile Processes" and "Distributed Software Development" shapes an area of both scientific interest and practical relevance. One the one hand, Agile Development methods with their approach to overcome *"the heaviness, commonly associated with traditional software-development methodologies"* [1] which relies on frequent and direct face-to-face communication, had their advent in the late 1990s and have been subject to numerous scientific studies since. On the other hand, the distribution of software development processes has become popular because of *"the three key advantages [. . .] (1) lower cost of labor, (2) capture talent not available locally, and (3) increase and decrease project size without layoffs"* [2].

Consequently, there is a strong attraction emanating from the combination of both. Practitioners ask how to transfer the advantages of Agile Development to a distributed setting and look for the recipe for success or at least a set of best practices. Researchers are tempted to debate on the immanent contradiction and its implications, and are striving for a better understanding of Agility itself (the distributed and the co-located variant).

Hence, one access to this field was opened through experience reports or, in a more systematic way, through case studies. Often these studies reflected extreme forms of distribution in which the involved persons were separated by up to 12.5 hours [3], [4]. Maybe this was due to the pure availability of settings, maybe the authors aimed at a maximized contrast for deeper insights. However, this sampling explains the strong focus on time-zone-related problems and how to solve (or circumvent) them, e. g. by using "strict communication plans" or by certain technical infrastructure (source code management, wikis, etc.) [3], [5], whereas other issues were rather incidental remarks. This is obviously the right way to start, since this temporal distance, at least in farshoring settings, is a prevalent characteristic. But as Berczuk [6] pointed out *"the distribution often amplifies existing process issues rather than being the issue itself"*.

We suppose Agile Development, and all the more Distributed Agile Development, to be difficult enough, even without the major impediment of shunted working hours. We conducted an exploratory case study which reflects a nearshoring setting in which no time zones but only corporate and national borders were crossed. This way, other issues stand out more clearly. Furthermore, the gained insights may also help farshored distributed and co-located Agile teams.

In their *tertiary* study on the Research on Agility in Global Software Engineering [7], Hanssen et al. noted a lack of common understanding of the concept of Agile. Surprisingly, only two out of twelve secondary studies turned out to be devoted to Agile. The first one, a systematic literature review by Hossain et al. [8], closes with a remark on the gap in literature when it comes to the actual Scrum processes in distributed software development. The other one, conducted by Jalali and Wohlin [9], praises the value of experience reports, but all the more demands for more rigorous research methods. The case study we present here aims at both narrowing the gap of actual process descriptions and applying a concise and comprehensible research process.

The remainder of the article is structured as follows. We will describe our methodology and the contextual background of this case study in Section II. The results are presented in two steps: Section III discusses the actual practices and contrasts them to their intended purposes; Section IV puts these pieces together to a conclusive picture to ease the understanding of the practices' interplay. We summarize our work in Section V.

II. Context of the Study and Methodology

The study presented here can be characterized as a case study with a distinctive exploratory orientation, which means in particular that it was not led by preconceived hypotheses. Rather, its purpose was to answer the following question: *"How does a Scrum-oriented software development process in a nearshoring setting work in practice?"*

In order to gather the appropriate data to answer this question, we utilized the GQM method [10]. One of our demands was to use a triangulation approach to incorporate multiple vantage points. We achieved this diversity by using different data sources and types, and by engaging multiple observers. The analysis of the collected data was conducted using a process following the principles of the Grounded Theory Methodology by Strauss and Corbin [11].

A. Context

The company at stake runs a large German online portal. The required software is developed and maintained by 180 in-house developers in the company's headquarter in Berlin. The developers almost exclusively work in small Agile teams (usually using Scrum) with up to six members. In accordance with the Agile Principles [12], the teams are granted with high responsibility. However, a smaller fraction of the development is out-sourced. At the time of our observation, three external Polish teams worked for the German company which were rented from a large IT Service Company (approx. 16,500 employees). These teams do not work in isolation from the in-house development. Rather they are embedded through a customized Scrum process, e. g. the Product Owners, who are situated in Berlin, take part in the teams' Daily Scrums using a videoconference system (we will describe the details of the applied practices later on, see Section III).

In the context of this study we observed two of the three external teams, hereinafter referred to as Team A and B, which have been working together with the German company for a little less than three years. We observed their daily work and sounded them on their current situation. At the time of our data collection, the teams' situations were as follows:

- Team A consists of five developers and one QA engineer, co-located in one workspace. The team has been in this form for several months, having only one developer passed over from Team B a few weeks ago. Thus, Team A can be characterized as rather stable.
  Team A was the only team working on a certain range of the German company's products. The team temporarily had two Berlin-based Product Owners which were in the midst of a handover.
  In their daily work, the team incorporated a Kanban-like board [13] for organizing their Stories which were broken down into Tasks. The team used different templates for Story cards and Task cards, both containing elaborated checklists for certain kinds of quality assurance. The applied practices include Continuous Integration (CI), Pair Programming (PP), Test-Driven Development (TDD) and Reviews (see Section III for details).
- In contrast to Team A, the history of Team B was more dynamic. At the time of our observation it consisted of four developers and one QA engineer, two of the developers being quite new to the team. The Scrum Master accordingly pointed out that *"the team is still not in the Performing state, it's still somewhere Storming/Norming"* and *"the team changes the way they work [...] it's still not working fully"*[1].
  Team B previously also worked on a project of its own. With this project being basically finished and waiting for further plans, Team B currently supported Team A with its products. Physically, both teams shared the same workspace. The team itself already had a Product Owner of its own, but because of practically working for Team A, the team also had contact with the two Product Owners of the other team.
  Just like Team A, Team B also used a Kanban-like board for its Stories and Tasks. However, its Task cards were merely hand-written sticky notes, without any checklists. The list of applied practices of Team B does not notably differ from that of Team A: CI, PP, and TDD.

Both teams share one Polish Scrum Master who is with the teams all the time. Prior to our observation, he was the local team leader of the three external teams and took over the role as the Scrum Master two weeks ago because the previous one left the external company. He was to fulfill the role in the interim until a new Scrum Master was hired.
In addition to this primary Scrum Master, there is a second "Scrum Master for External Teams" who is situated in Berlin. This additional role bears the overall responsibility for the cooperation between the internal and the external teams. Whereas the Polish developers spoke little or no German at all, using their mother tongue for internal, and English for external purposes, the additional Scrum Master was fluent in English, German, and Polish. The main Scrum Master was an employee of the external company; the additional one was employed by the German company.

B. Preparation and Data Collection

The research objective already sketched above reads as follows in its detailed form: *Characterization and preliminary evaluation of appropriateness, effectiveness, efficiency, consistency, maturity, acceptance, and completeness of the ongoing Scrum process as seen by the Scrum team. The focus of the evaluation is on the identification of problems and leadoff hypotheses regarding direct an indirect causes. Particular attention is to be directed on aspects related to the nearshoring situation.* Therefore, the actual state was to be observed and evaluated.

Following the GQM method, we broke down this goal into questions to be answered. These questions included the following:

Q1 What is the current situation of the project?

Q2 Which amendments to Scrum have been made (e. g. with other management techniques, process models, or development practices)?

Q3 How well does the process fit the project's conditions, i. e. the distribution?

Q4 Which problems were mitigated or eliminated by the actual applied process?

Q5 Which problems evolved due to the actual applied process?

Q6 How satisfied are the involved persons?

[1] According to the psychologist Bruce W. Tuckman [14], a group goes through several states including Forming, Storming, Norming, Performing.

After formulating these questions, we specified our tools for data collection and devised how to answer the questions. As stated above, we wanted our study to be exploratory, i. e. open-minded and unbiased. Therefore, we took care that our instruments allowed us to react on unexpected but interesting phenomena, e. g. leading to surveys containing a fair amount of 15 open questions (compared to 40 questions in total).

The following instruments were specified:

- We devised semi-structured interviews with individual developers, Scrum Masters, and Product Owners for topics from which follow-up questions are expected.
- We designed an online survey for covering simpler aspects and a larger number of people.
- "Fly on the Wall": In order to ease interpretation and combination of subjective data retrieved from the other instruments and to get aware of additional phenomena, we decided to passively observe the everyday work including the Daily Scrum, the Sync meeting (see Section III-F) and the developers' interaction with their working environment. In the same way we observed Sprint Retrospectives and Sprint Planning Meetings.

The actual data collection took place in Poland (two days) and Germany (two days). Additionally, we discussed our findings with the respective teams eight weeks later. We will refer to the observations we made and the feedback we got during these discussions throughout this paper.

The vast majority of the interviews and the observations were carried out jointly by at least two researchers. This served two purposes: We wanted to avoid missing of possibly important phenomena and to eliminate personal biases. All in all we gathered over 50 pages of notes and sketches, 5 $^1/_2$ hours of recorded interviews, and additional 3 hours of recorded discussions.

It should be noted that we already investigated two internal teams of the same company one year prior to the study presented here. Back then, one of these teams worked together with Team B on their – now finished – product. We also conducted extensive interviews with members of the management, which eased our comprehension of the current constraints and recent phenomena for the study at hand.

C. Analysis

The analysis of our collected data followed the principles of the Grounded Theory Methodology by Strauss and Corbin [11], especially for identifying and working out central process-related phenomena. As we did not want to formulate a general theory of Agile Nearshoring, not all GTM practices were employed. E. g. our "theoretical sampling" was limited to enquiries towards the German company if certain phenomena or constraints were unclear to us. Furthermore, instead of directly starting off with "open coding", our first step right after the collection of data was to summarize and consolidate our impressions. This led to preliminary hypotheses which we scrutinized on the basis of our actual data (see Figure 1 for Team B's initial hypotheses). However, not all of the hypotheses proved to be tenable, and not all insights presented here originate from these hypotheses. Nevertheless, this first model helped us to structure our research process and to avoid drowning in details.

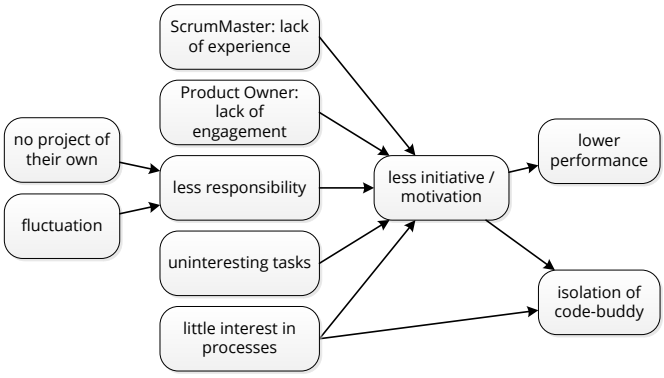

Fig. 1. Preliminary hypotheses for explaining the situation of Team B. Boxes are a rough description of complex phenomena, arrows can be read as "leads to". Both the boxes and the arrows are hypothetical and sometimes, e. g. in case of the "uninteresting tasks", turned out to be untenable.

III. Aimed Objectives & Actual Practices

In this section we will discuss the actual practices we observed and contrast them to the intended way of proceeding. In each subsection we will first summarize the key points of the management's and/or the team members' (developer, Scrum Master, Product Owner) description of the intention of particular practices. We will use the notions of Schwaber and Sutherland's Scrum Guide [15] when our investigation yielded no particular indications of idiosyncratic conceptions. Afterwards, we compare this idealized notion to the actual observations we made. Occasionally, we will also differentiate between the two Teams A and B.

Please note that these differences are neither good nor bad *per se*. If possible, we will discuss the origins and the consequences of the differences. Most of these claims are neatly grounded in our data. However, some of them are rather hypotheses to explain certain phenomena. If possible, we close with an evaluation, based on the outcomes of the subsequent feedback phase.

A. Sprints, Starting and Finishing Ceremonies

Both external teams were performing **Sprints** with a length of three weeks. To support direct communication with the Product Owners, the teams stayed in Berlin for two consecutive days for finishing one Sprint (holding Sprint Review and Sprint Retrospective) and starting a new one (Sprint Planning Meeting).

A side-effect of these stays was that the Polish developers had the chance to meet their German colleagues face-to-face. One developer recalled: *"We were in Berlin, and there was a problem that we had, so I wrote an e-mail to a group and after 15 minutes a guy that I haven't seen before, came and said 'Hi, I can help you' "*. Furthermore, these personal contacts were likely to ease overcoming possible inhibitions to subsequently phone each other. According to the developers, Product Owners, and Scrum Masters, communication has improved a lot over the last year.

B. Scrum Masters

According to the Scrum Guide: *"The Scrum Master is responsible for ensuring Scrum is understood and enacted"* [15]. He is often described as a "servant leader" and he serves the development team, but also the Product Owner and the organization [15].

Aside from being fairly new in the role of the **Scrum Master** (he only fulfills it in the interim because the previous Scrum Master left the external company), the only "service" we observed was for the development team. In particular, the Scrum Master neither had contact with the other Scrum Masters working for the German company nor with the teams' Product Owners (aside from the Sprint Planning Meeting and Daily Scrums). Furthermore, throughout our observation his dealing with the developers lacked the typical traits of a Scrum Master. Probably both the missing integration into the German company and the inexperienced behavior were due to his newness, and indeed, both clearly improved until our discussion event eight weeks later.

Furthermore, there was a **Scrum Master for External Teams** who resided in Berlin. In the past, he was responsible for setting up the communication between the two sites, i. e. by *"building bridges where needed"*. To initially shape the flow of information, he channeled all communication, deliberately making himself a bottleneck. Afterwards he gradually withdrew to allow direct communication, intervening only if needed. E. g. the majority of the German company's electronic in-house communication was written in German, thus excluding the external teams from both formal and informal information flow, so the additional Scrum Master repeatedly reminded the internal teams to switch to English and took the lead in doing so.
In principle, the Polish Scrum Master on-site is the main point of contact for the German Product Owners in case the delivered quality is not satisfying. However, occasionally the Germans still simply talk to the External Scrum Master when meeting him in the hallway. And vice versa: The External Scrum Master supports the Polish Scrum Master, e. g. if he needs to get in contact with multiple German stakeholders at once.
However, over the past two years, the cooperation between the two sites improved so much that the External Scrum Master became able to serve an additional internal team. Nevertheless, he still bears the overall responsibility for the cooperation between the internal and the external teams and visits the external teams on-site every three to four weeks.

C. Code Buddy

Although the external teams worked on dedicated products, their work was not completely independent from the work of the internal teams. Since the German company worked with a considerable amount of legacy code, a perfect separation of concerns was nearly impossible.

Twelve months prior to our observation, the management introduced a new role named **Code Buddy** for facilitating the cooperation between the German and the Polish teams. Every other week one developer from each Polish team stays in Berlin for five days to work close by the German teams. In particular, the Code Buddy role is meant to serve the following purposes:

- He represents his team in Berlin. If a German developer needs to get in contact with one from the Polish teams, he can walk over to the Code-Buddy's desk and talk to him face-to-face instead of calling Poland.
- And vice versa: His Polish team members can ask him to talk to a certain German developer or to the in-house IT, who are difficult to reach by phone.

Hence, the Code Buddy can be viewed as a bi-directional proxy. That does not mean he needs to channel all communication, rather he introduces people to enable direct communication and is a fallback if this direct contact is temporarily not available.

In these regards, the portrayals of the management who introduced this role, and the developers who actually worked together with the Code Buddy (or fulfilled the role themselves) were in conformity. (We did not interview German developers though.)

Reflecting on these points, the Code Buddy role has two different aspects: technical ones and rather social ones (these aspects are also schematically shown in Figure 2).

On a *technical level*, the Code Buddy is supposed to facilitate the inter-team communication between his team and the internal German teams. In some way, the role therefore tries to fulfill the purpose of a (Distributed) Scrum of Scrums (linking Scrum teams to encourage *"communication, cooperation, and cross-fertilization"* [2]), but fails to do this with the same efficiency because of his communication events being purely need-driven rather than institutionalized. Nevertheless, there were plans at the German company to let the Code Buddy attend the Daily Scrums of other teams, thus approaching an institutionalized communication between internal and external teams.

On the *social level*, the mere presence of a Polish developer in Berlin should help the German developers to see the external developers as equal, which in turn should ease their integration into the company. According to the Product Owners, the appreciation of the Polish developers in Berlin indeed improved over time, at least partially because of the Code Buddy's physical presence. On the personal level, the Polish developers seemed to enjoy being in Berlin (e. g. the company rented an apartment, for the Code Buddy). We consider this enjoyment of the stays to be a healthy sign.

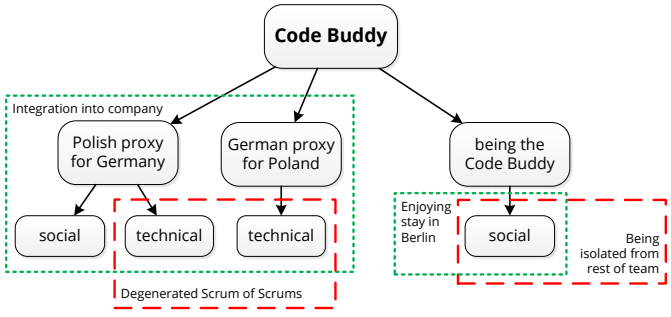

Fig. 2. Technical and social aspects of the Code Buddy role, including both purposes and side-effects. The color of the frames characterizes the favorableness of certain (group of) phenomena (red/dashed = undesired, green/dotted = welcome). See Subsection III-C for details.

However, the Teams A and B differ in various ways when it comes to the details of the role "implementation":

- The Product Owners of Team A regarded the Code Buddy as a normal developer working on the same Stories as the rest of his team.
  Yet, during the Daily Scrum the Code Buddy seemed slightly isolated because he did not know which Tasks to work on. The rest of his team reacted adeptly by sending him a list of open Tasks related to their current Stories – the Code Buddy was to pick one and inform the team about his choice.
  As stated by both developers and Product Owners, the Code Buddy provided technical support during the acceptance tests (e. g. changing database entries directly in order to see the changes in the user interface) and hence profits from immediate feedback (see the "War Room" practice in Subsection III-G for a similar effect).
- In contrast, the Product Owner of Team B emphasized the first proxy property mentioned above *("The Code Buddy is here for me")*, and was said to be working on Stories of his own.
  During the retrospective the Code Buddy of Team B proclaimed he felt isolated from the rest of his team during his stay in Germany. This also had technical consequences: the Code Buddy started a new Story on his own because the team saw no possibility to integrate him into their actual development process.
  The acceptance tests were conducted at a stretch at the end of the Sprint, and without assistance of the Code Buddy.

We will summarize the evaluation of Code Buddy role in Subsection IV-D.

D. Sprint Retrospective

Scrum Guide: The **Sprint Retrospective** *"is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint"* [15]. During this meeting, the *"Scrum Master encourages the Scrum Team to improve [. . .] its development process and practices to make it more effective and enjoyable for the next Sprint"* [15].

In practice, the Product Owners were not present during the Retrospectives, which were, for the developers' convenience, usually conducted in their mother tongue Polish. However, for the purpose of our observation, the teams switched to English.

Since both teams shared the same Scrum Master, their Retrospectives had some things in common. They started with the team members (including the Scrum Master himself) individually writing "good" and "bad" aspects of the recent Sprint onto separate cards. Each member shortly explained his cards and stuck them on a flip chart, grouping them on-the-fly. The Scrum Master afterwards determined the *"most important points"* without consulting the rest of the team. However, some additional cards were chosen on developers' request. For each of these points, the team discussed if and how this point should be approached and usually to-dos including a responsible person were formulated. However, we do not know what happened with these to-dos afterwards. Furthermore, there was no evaluation of the previous Retrospective in means of determining whether the former goals have been achieved.

Aside from these similarities, the Retrospectives of the two teams still showed a number of differences:

- Team A clearly demonstrated awareness of self-defined rules, e. g. by recalling *"not to discuss one-time-things"*. The formulated to-dos had the potential to actually improve the next Sprint (e. g. *consider effort for bug-fixes because of belated tests during the last Sprint*, or *split Stories with too long acceptance criteria*). Finding responsible persons went smooth.
  The departure of the previous Scrum Master was mentioned by almost every developer, revealing a high appreciation.
  The atmosphere was relaxed and left room for humorous cards like *"Good: Cookies during meeting"*.
- Team B showed no interest in institutionalizing practices (*"We don't want to do War Rooms for every Story like Team A. We decide this by gut instinct"*).
  For many of the discussed points, the team did not see the need for formulating a to-do because they simply *"will take care in the future"*. Additionally, finding a responsible person for once formulated to-do proved difficult.
  The previous Scrum Master was not mentioned once.
  The atmosphere was a bit more strained than Team A's, though not tensed. All of the positive cards were purely technical.

Roughly speaking, throughout the Retrospective Team A clearly showed more interest in taking responsibility and improving their process, whereas Team B simply wanted to get the technical stuff done.

E. Sprint Planning

Scrum Guide: During a **Sprint Planning** meeting, the *"Product Owner presents ordered Product Backlog items to the Development Team and the entire Scrum Team collaborates on understanding the work of the Sprint"* [15]. This description held for both external teams: The items presented by the Product Owners ("Stories") were discussed, and changes to the acceptance criteria were made until everyone was satisfied.

In contrast, the next step was not in the scope of the Sprint Planning Meetings we attended: *"After the Development Team forecasts the Product Backlog items it will deliver in the Sprint, the Scrum Team crafts a Sprint Goal"* [15]. Both teams only estimated the effort for the Stories presented by their Product Owner(s) rather than also committing to a subset of them, which was caught up in a "Sprint Planning II" we did not observe. The estimation was done by means of "Story Points", a non-standardized metric for estimating effort, instead of an absolute metric like ideal working days.

We observed a number of differences between Teams A and B in the way of holding these meetings, including the following:

- At the end of the meeting the Product Owners of Team A summarized all Stories discussed in the meeting, and

reiterated the priorities of the Stories, thus drafting a "Sprint Goal light".
The Product Owner of Team B did nothing comparable.

- During the discussions in Team A, the developers recalled an issue from their previous Retrospective (the team agreed on asking the Product Owner for crucial aspects in the Story acceptance criteria in order to adjust their Smoke Tests accordingly).
  In contrast, Team B showed no such "reflective" behavior – lacking appropriate to-dos from their Retrospective after all.

The Sprint Plannings of the teams did not differ much on the content level on which they were both carried out effectively. However, Team A's meeting seemed more enjoyable for all participants and the reasons for this are probably beyond the scope of this study.

F. Daily Scrum & Sync Meeting

Scrum Guide: *"The Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours."* It *"is not a status meeting, and is for the people transforming the Product Backlog items into an Increment."* Finally, it *"eliminates other meetings"* [15].

The **Daily Scrums** of Teams A and B were held via a videoconference system and were attended by all members of the Scrum team, i. e. all developers (including the Code Buddy), the Scrum Master, and the Product Owner(s). The External Scrum Master only attends when he resides in Poland – he does not participate when in Germany. This practice is meant to reduce the intra-team communication risks as it brings the two sites together. The attendance of the Product Owners in the Daily Scrum is not a novelty due to the distribution, it is rather the usual way to conduct these meetings in the German company. The Daily Scrum is the only time of the day when the cards on the teams' boards are altered or moved.

In addition to the commonly known Daily Scrum both teams applied a practice they called **Synchronization Meeting** ("Sync" in short). The Sync starts 15 minutes ahead of the Daily Scrum and has the following properties:

- The purpose of the meeting is to clarify technical aspects of the team's work that the Code Buddy needs to know before starting the actual Daily Scrum. The idea is to keep technical details out of the Daily Scrum.
- The Product Owner does not participate. For the developers attendance is not mandatory, but usually everyone is present.
- There are no rules comparable to those of the Daily Scrum: neither are there predefined questions to answer, nor any need to say something at all. The whole meeting is driven by the developers' needs and continues until no more questions pop up. Hence, there is no time-boxing: If need be, the Sync will be resumed right after the Daily Scrum, practically occupying all developers.

The Sync meeting contrasts with the Daily Scrum in two ways:

- The commonly held (i. e. including the Product Owner) Daily Scrum was intentionally introduced from corporate management to reduce the communication risks, whereas the Sync meeting evolved from the teams themselves as a consequence of the Code Buddy being separated from the rest of his team and feeling the need to stay in touch. However, both teams did not recall how, when, and by which team this practice was initially introduced.
- Thus the Daily Scrum follows commonly known and accepted rules, whereas the Sync meeting clearly lacks deliberate consideration.

First of all, the Sync Meetings obviously did not answer all of the Code Buddy's questions since the Code Buddies of both teams did not know on which Task/Story to work until their respective Daily Scrum. Apart from that, according to the developers, the Sync meeting somewhat replaced the Daily Scrum, which in turn degenerated into a plain status report for the Product Owner (*"just to show the Product Owner that we do stuff"*) instead of being a planning meeting. This clearly contradicts the Scrum Guide and, much worse, is disliked by the developers because they already know almost everything they are going to hear. But instead of dunning this kind of Daily Scrum, they accept it because they feel as an *"external company that needs to show progress"*.

One the one hand, the Product Owner gets the impression to be well informed and is satisfied because she always gets a furbished report of what is going on. On the other hand, this might make the teams consider themselves safe because they always manage to keep the 15-minute-time-box, whereas the actual purpose of this limit is undermined by the rule-abolished Sync meeting.

If both introduction and execution of practices are subject to conscious reflection, the evolvement of new practices is an indication of a healthy Agile process. However, we consider both the current Daily Scrum and the newly introduced Sync meeting to be undesired deviations.

G. The War Room

The **War Room** is a practice the teams introduced some iterations prior to our visit (and again, the teams did not recall by whom it was introduced). Before a Story is given to the Product Owner for acceptance, the whole development team sits together and tries to "break" the new feature, thus discovering several failures earlier in the process without requiring the Product Owner's time.

The Product Owners claimed to receive much better results since the teams had applied this practice because *"they think about their products from the end-users' perspective"*. While in a co-located team the Product Owner might directly explain or show the fault he found, in a distributed setting he uses the phone, e-mail or bug tracking system. By discovering the failures earlier during the process, the need for this indirect communication drops. Thus, the teams' overall velocity did not decrease because of this additional task; effectively, it increased because of the saved feedback loops between the developers and Product Owner.

Overall, the War Room is a practice to avoid communication between the two sites by cutting off the underlying needs.

H. Pair Programming & Test-Driven Development

The developers stated they are *"encouraged to do Pair Programming and Test-Driven Development"* since it increased the code quality, difficult problems could be solved more easily, and the overall velocity did not drop, in fact, it maybe was even higher. Most of the work is said to be done in pairs, excluding only small tasks. The decision whether to pair or not is said to be made during the Sync meeting.

During the Retrospectives and the interviews, and in the online surveys multiple developers of both teams praised pair programming as a *"really cool"* practice. In the teams' workspace we counted more chairs than developers, thus easing spontaneous pairings, which actually happened more than once. During our field observation we did not recognize differences worth mentioning in the occasions and applications of this practice. However, Test-Driven Development is perceived as a more ambiguous practice, mainly because of resentments against ultra-high code coverage.

Interestingly, the teams additionally apply practices they both call "Simultaneous Programming", which have slightly different meanings depending on the applying team. Both practices involve multiple (more than two) persons. In Team A, there are three persons working on two computers (two fixed, and one oscillating between them). It is a practice they seldom apply and which is probably rather a spontaneous extension of Pair Programming sessions which a third developer, working on a task of his own, interrupts by pulling out one of the pair members. In Team B, three or four developers crowd in front of one big screen, e. g. to deal with hard-to-understand legacy code. They said they do this quite often, and despite initial doubts, it *"really works well"*. However, neither of these variants could be observed in action and the economic efficiency is, if not precarious, at least worth considering.

IV. Putting the Pieces together

The previous section discussed the various practices the two teams applied in their daily work, focusing on the differences between the intended way and the actual process. So far, these practices were discussed in isolation, whereas in this section we will continue investigating their interplay.

Albeit we observed two external teams, we found the most interesting issues combined in only one of them: Team B. Therefore, this section will primarily talk about Team B's situation, using our findings from Team A merely for a better contrast.

A. Hanging of Team B

The product Team B previously worked on was basically finished and the decision whether to implement additional features or to start a new project was still open. Furthermore, a previously announced decision deadline was not met due to unsettled organizational assignments, resulting in an even longer lean period. However, the project of Team A continued and aid work was possible, so the decision was made to let Team B support Team A, leaving Team B *"on stand-by"* until the decision is finally made.

We will now emphasize some aspects of this situation:

- This constellation had a drawback which originates from the company's culture to rather statically assign teams to their Product Owners. Since Team A already had two Product Owners (in the midst of a handover) and both teams now basically worked on the same project, Team B actually worked together with *three* Product Owners, complicating both coordination and communication.
  During the events we attended both teams only had contact with their direct Product Owners. However, according to a Team B developer, during the recent Daily Scrums his team had contact with all three Product Owners *"and it happens that there are really strange people that I don't know"*.
- Consequently, there was no clear separation of the topics and the sovereignties of the two Teams A and B. E. g. during the Sprint Planning of Team B, developers of Team A needed to be called in to clear up the dependencies between two Team-B-Stories.
- Additionally, Team B's dedicated Product Owner, who at her own judgment already had *"eight irons in the fire"*, now was in charge of a foreign topic. E. g. during the same Sprint Planning Meeting, the Product Owner could not speak with full confidence, and referred to what she heard from the Product Owners of Team A.
- Finally, Team B was not provided with a clear vision, which was also criticized but not really discussed in their Retrospective (*"messy roadmap"*, *"no own product"*). The Product Owner's omission of a Sprint summary during the Sprint Planning was a further indication of her not feeling at home in this topic.

From our point of view, this kind of constellation has two major drawbacks. Both of them lack clear empirical evidence, and are merely hypotheses which could guide further studies. Nevertheless, we still find them important enough to be discussed here.

The first problem is a lack of *intrinsic motivation* [16]. The developers of Team B complained about their uncertain future, which indicates how difficult it may be for the Product Owner to formulate a clear vision under these circumstances. The exceeded decision deadline makes it harder for the team to preserve their trust in the company's will to alter these circumstances. With intrinsic motivation falling apart, only the extrinsic motivation (i. e. professionalism) remains. A team like this might still perform well for a while, but not for long.

The second problem is closely related to the first: A lack of *responsibility*. A team without a dedicated product has no good reason – professionalism aside – to feel responsible for it. With responsibility being a core ingredient of Agile processes, this poses a considerable risk for the team's long-term performance. A possible manifestation of this problem might be the reluctance to take over responsibility during the Retrospective.

Confronted with this point of view during the final discussion, Team B and their Product Owner claimed that (1) there had not been a state of being "on stand-by" (despite the statements in the interviews and the Retrospective) and (2) either way, the team did not have the choice to do anything differently. Given this mindset, we saw no opportunity to

discuss this topic any further.

In a non-distributed setting, the odds for this to happen might be lower because the moans we heard when being on-site were probably too subtle to be heard in the upper organizational levels in Germany.

The secondary study of Hossain et al. [8] revealed the practice of quarterly presentations of the Product Vision to be useful in some projects. Maybe the Product Owner's strong need to be able to present such a vision could start an upward cascade, seizing the levels of organization that caused this long lean period.

B. Possible Performance Misconception

During our analysis we stumbled upon an interesting discrepancy. One the one hand, the Product Owner of Team B was very proud of the team, praised their velocity using superlatives and even recalled a recent leap forward. The developers themselves were proud to always deliver what they had committed to.

On the other hand, there had been recent personnel changes in Team B – members left, new ones joined. The Scrum Master accordingly pointed out that the team is not in the "Performing" state yet.

Given these circumstances, technically both the perceived consistently high performance and the recent performance boost are unlikely to be accurate.

We presented an alternative explanation for the Product Owner's impression during our discussion session: The development team worked on a different topic than before, hence the velocity is expected to drop. The Product Owner herself also worked in a different field, which tends to decrease the performance evaluations' accuracy. Furthermore, the Product Owner worked on far more than one topic and naturally had less time for both providing valuable guidance and evaluating the performance. Hence, the satisfaction might have been delusive and the team might not have performed as good as perceived by the Product Owner.

The Product Owner tried to weaken our points with two arguments. Firstly, the effort was measured by using Story Points which were *"independent of the topic"*.
The problem of measuring effort with a relative measure like Story Points is obvious: The team needs experience in the particular field of work to estimate accurately. Story Points are a highly subjective metric, and are *not* independent of the topic.

Secondly, she admitted that she had indeed worried about dropping performance upon fluctuation, but in the past that had never actually happened.
But even if the second point was true, two problems remain. First, the developers might struggle hard after a fluctuation to sustain their performance level (e. g. in order to not jeopardize the cooperation contract). This point was not fully rejected during the discussion session. Second, even if the developers do not struggle to maintain their level with fewer employees, the question remains why they do not perform higher when fully staffed. Again, in a non-distributed environment, the situation would be slightly different, since the Product Owner would probably scent the risen exertion after a personnel change.

From our point of view, the whole team (not only the developers) was lacking a pro-active approach towards the issue of fluctuation, performance, and expectations towards performance.

C. Little Sensitivity for Process Issues

Throughout our observation, Team B's focus was primarily adjusted on technical issues. Although the ability to concentrate on the actual work is important to achieve a high velocity, it is an important ingredient of Agile processes to "inspect & adapt" [12]. The developers need to step back, switch the perspective, and think about the process itself.

The very Scrum ceremony for this is the Retrospective at the end of each Sprint, which, at least in the case of Team B, was clearly dominated by technical issues. The interim Scrum Master did not shift the focus, instead he afterwards said *"the Retrospective was this technical because I'm a technical person"*. This is a dangerous statement: The overall direction of the Retrospective should be set by the whole team. The role of the Scrum Master is to enable the team to be Agile, i. e. to think about the process.

From the developers themselves we got statements like *"there are no process problems in such small teams"* and *"there is no need for continuous improvement."* The overall conviction was like: *"We as developers are not responsible for the process. That's other people's business."* That attitude is even more dangerous because it is the very duty of the *whole* team to forge the Agile process, and not a task which is confined to a particular person or role, like the Scrum Master.

Throughout the final discussion, Team B reinforced their attitude, e. g. by seconding their former statements. Finally, the session itself was less a vivid discussion than a presentation of hypotheses from our side, and shielding and rejecting from the team's side, revealing little critical faculties (see a further discussion on this in Subsection V-C).

Team B did all the Scrum practices (Daily Scrum, Sprint Planning, Review and Retrospective), and introduced new ones (e. g. Sync Meeting), but at least three of them did not serve their intended purpose – or even lack a well thought out intention. These deviations did not attract much attention because their effect on the performance – however big or small – was not clearly perceived. Neither were they the subject of internal inspections by the team itself. Therefore, the team somehow *did Scrum*, but *was not Agile*.

This lack of reflection (or at least lack of sensitivity for process issues) might also explain why the possible overburdening of Team B (especially after fluctuation, see previous subsection) was not recognized or at least not brought up. Another reason might be that both teams (not only Team B) considered themselves too much as a service provider for the German company instead of an equally treated, self-organizing Agile team. The silent acceptance of the "Status" Daily Scrum, the *"need to show progress"*, and the negation of an unpleasant "hanging mode" affirm this hypothesis. Either way, an open discussion between the involved parties would probably improve the overall situation.

D. Isolated Code Buddy

The very practice that reflects the distributed setting of this case study is the Code Buddy role. Therefore, the observed phenomenon of Team B's Code Buddy feeling isolated described above is of particular interest.

During the Daily Scrum, the Code Buddy could not follow the line of thought of the rest of his team (*"Which Stories are you talking about?"*) and also stated that he did not know what to work on. Eventually, the team decided to start a new Story solely for the Code Buddy, thus decreasing the need for intra-team communication.

In the Retrospective one week later, this exclusion from the team was discussed shortly without yielding any solution. The developers explained to their Scrum Master that any attempt to improve the communication with the Code Buddy would be in vain, because the Code Buddy sat *"next to the Product Owners yelling at their phones"*. This stalling position was defended during the final discussion as we proposed to give Distributed Pair Programming (e. g. using Saros[2] or a similar tool) a try to improve the integration of the Code Buddy into his team.

Ironically, the Code Buddy role was meant to facilitate communication, but he himself suffered from a shortage of communication. However, we did not hear anything like this from the other team which applied the same practice. The only difference worth mentioning lies in the history of the role implementation: In Team A the same developer fulfilled the Code Buddy role for a long period of time, whereas in Team B the role rotated. Even if this led to more intense personal contacts in Berlin, we were not able to explain how this difference affects the cohesion of the team. Either way, during the closing discussion Team B made clear, that for them being the Code Buddy multiple times in a row was out of question.

In summary, the Code Buddy role is clearly a neat practice for ice-breaking and warming up. The absence of direct team members forces the Code Buddy to get in contact with the internal teams himself. Nevertheless, a year after its introduction the role conception itself might be worth reconsidering. The company's management should ask itself what they expect of this role, and what this role is capable of. Maybe it is no longer the best practice to achieve the actual objective of integrating internal and external teams.

V. Conclusion

In the first subsection we will answer our the questions raised in Section II-B and then summarize our insights. Furthermore, we provide additional Lessons Learned on both the content level and regarding the research process.

A. Answering the GQM Questions

Q1 *What is the current situation of the project?*
We investigated a nearshoring setting in which two Polish Scrum teams (A and B) worked for a German company. In principle, the external teams worked on dedicated products, and each team had a so-called "Code Buddy" who facilitated the residual integration work by spending five consecutive days in the company's headquarter. However, Team B's work was done for now, so it supported Team A until further notice. This situation and the yet unclear future of Team B may have resulted in weakened motivation and responsibility. Recent personnel changes in Team B and a partially isolated Code Buddy might have lowered the team's potential. In addition, working in a foreign topic (for both developers and Product Owner) may have hampered the accuracy of the performance assessment so that an unrecognized underachievement seemed possible.

Q2 *Which amendments to Scrum have been made?*
In addition to local practices borrowed from eXtreme Programming (e. g. Pair Programming and Test-Driven Development), there were several practices reflecting the distributed nature of the projects. First of all, the Daily Scrums were conducted in a distributed manner, also involving the Product Owner via a videoconference system. Furthermore, there were two additional roles, namely the "Code Buddy" (as described above), and the "Scrum Master for External Teams" who bears the overall responsibility for the cooperation. A home-grown practice named "Sync Meeting" – which is conducted without the Product Owner – reflected the Code Buddy's need to stay in touch with the rest of his team.

Q3, Q4, & Q5 *How well does the process fit the project's conditions? Which problems were mitigated or eliminated by and which evolved due to the actual applied process?*
The Sync Meetings lacked the rules other events have (e. g. time-boxing) and took over the planning trait from the Daily Scrums, which in turn degenerated into a demonstration of progress to the Product Owner. Hence, the Product Owner might had gotten the impression of being in the thick of the actual process (which the Product Owners of internal teams really are), but in fact retrieved only a furbished status report. Combined with the seemingly constant performance (measured in obfuscating Story Points) this might have conveyed the delusive feeling of mitigated nearshoring risks.
Team B's attitude was maybe co-determined by the position of being a service provider, so they felt no urge to change themselves, since the final outcome appealed to the purchasing company. Accordingly, the process-critical Sprint Retrospective was minimally constructive and revealed little potential to improve the subsequent Sprint. The yet inexperienced Scrum Master was not able to draw the developers' attention to process-related issues. Overall, Team B technically did Scrum, but without being Agile.

Q6 *How satisfied are the involved persons?*
Instead of yielding a simple answer, our research led us to further questions: What does satisfaction mean, i. e. how does a Product Owner know that his team actually performs well? Clearly, measuring effort with Story Points cannot yield accurate performance ratios, and Product Owners need to be aware of this deficiency. In the end, they have to decide whether they want to look under the hood (but hereby must not fool themselves) or if they trust the developers as a self-organizing team.
Throughout the observed events, the developers of Team A seemed more relaxed and enjoying their jobs, whereas Team B seemed a bit tensed, as if the unanimously positive descriptions

[2] An Eclipse plug-in for Distributed Collaborative Editing (see http://www.saros-project.org)

of their current situation (e. g. in the surveys) were telling only half the story. But that might be an error in "measurement", because the silent observation of their daily work revealed no according indications. Maybe the developers did not felt themselves in a valuation situation when communicating informally using their mother tongue (see the remarks on the research process below).

B. Further Lessons Learned

The principle of "inspect & adapt" should be applied on multiple levels. Not only should the teams evaluate their own decisions and react accordingly, but also should the higher organizational levels which introduced the roles of the Code Buddy and the External Scrum Master, and have kept them since. Both had their particular purposes and achieved beneficial effects. However, one should not lean back but reconsider these decisions (e. g. within the scope of high-level Retrospectives), and draw conclusions if need be.
Admittedly, in a perfect Agile World, such decisions from the outside would not exist. The teams themselves would decide upon the necessity of new roles and their implementation details. However, it is well comprehensible that certain foundations need to be laid by a higher authority (i. e. renting an apartment for the Code Buddy is no decision to be made by the teams on their own). But maybe the time has come to grant the teams more decision-making competence, starting with making them aware of the fact that they can contribute to these kind of things instead of only being the receivers.

C. Summary

In this case study we presented a nearshoring setting in which two Polish Scrum teams worked for a German company. We contrasted the process elements self-imposed by the German company's management and the teams with the actual process and the applied practices.

The overall goal of the study was to devise a broad characterization and preliminary evaluation of the actual Scrum process regarding different aspects (*appropriateness, effectiveness, efficiency, consistency, maturity, acceptance, and completeness*), to make proposals for improvements, and to guide potential specialized follow-up studies.

Agile nearshoring is feasible and may produce high satisfaction amongst Product Owners since the deliverables meet the expectations regarding functionality and quality (*effectiveness*). However, the analyzed constellation did not ensure an actual Agile process (*consistency, completeness, maturity*). The involved teams need to develop "process awareness" or "agility" (*acceptance, consistency*) in order to achieve the desired project goals (*appropriateness*) and to keep hidden costs low, which are entailed by a lack of self-reflection and hence adaptability (*efficiency*).

To determine the actual magnitude of friction loss in the investigated setting, a longitudinal observation of the teams and their environment would be required.

Let us close by sharing an additional insight we gained during data gathering. The observed teams ought to be well informed about what is going on. We got the impression, especially since we worked with external teams, the teams sometimes felt evaluated "from above" instead of perceiving our questions and feedback as an opportunity to improve their own process. It might be a good idea to let the management clarify that the examination is without consequences for the team members. Otherwise, we as researchers are possibly perceived as a threat, which might explain the impression of some persons keeping a low profile.

Acknowledgment

The authors would like to thank the observed teams for their patience and feedback.

This work was partially supported by a DFG grant.

References

[1] J. Erickson, K. Lyytinen, and K. Siau, "Agile modeling, agile software development, and extreme programming: The state of research," *Journal of Database Management*, vol. 16, pp. 88–100, 2005.

[2] J. Sutherland, G. Schoonheim, and M. Rijk, "Fully distributed scrum: Replicating local productivity and quality with offshore teams," in *Proceedings of the 42nd Hawaii International Conference on Systems Sciences*, 2009, pp. 1–8.

[3] B. Drummond and J. F. Unson, "Yahoo! distributed agile: Notes from the world over," in *Proceedings of the Agile Conference*, 2008, pp. 315–321.

[4] E. Therrien, "Overcoming the challenges of building a distributed agile organization," in *Proceedings of the Agile Conference*, 2008, pp. 368–372.

[5] M. Vax and S. Michaud, "Distributed agile: Growing a practice together," in *Proceedings of the Agile Conference*, 2008, pp. 310–314.

[6] S. Berczuk, "Back to basics: The role of agile principles in success with an distributed scrum team," in *Proceedings of the Agile Conference*, 2007, pp. 382–388.

[7] G. K. Hanssen, D. Šmite, and N. B. Moe, "Signs of agile trends in global software engineering research: A tertiary study," in *6th International Conference on Global Software Engineering Workshop (ICGSEW)*, 2011, pp. 17–23.

[8] E. Hossain, M. A. Babar, H. Paik, and J. Verner, "Risk identification and mitigation processes for using scrum in global software development: A conceptual framework," in *Proceedings of the 16th Asia-Pacific Software Engineering Conference*, 2009, pp. 457–464.

[9] S. Jalali and C. Wohlin, "Agile practices in global software engineering a systematic map," in *Proceedings of the 5th International Conference on Global Software Engineering*, 2010, pp. 45–54.

[10] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Transactions on Software Engineering*, vol. 10, no. 6, pp. 728–738, Nov. 1984.

[11] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[12] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," http://www.agilemanifesto.org, 2001, last access: 2013-02-22.

[13] D. J. Anderson, *Agile management for software engineering: Applying the theory of constraints for business results*. Prentice Hall, 2004.

[14] B. W. Tuckman, "Developmental sequence in small groups," *Psychological Bulletin*, vol. 63, no. 6, pp. 384–369, 1965.

[15] K. Schwaber and J. Sutherland, "The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game," http://www.scrum.org/Scrum-Guides, October 2011, last access: 2013-02-18.

[16] J. E. Barbuto, Jr. and R. W. Scholl, "Motivation sources inventory: Development and validation of new scales to measure an integrative taxonomy of motivation," *Psychological Reports*, vol. 82, no. 3, pp. 1011–1022, 1998.