Adapting to Changes in a Project's DNA

A Descriptive Case Study on the Effects of Transforming Agile Single-site to Distributed Software Development

Raoul Vallon*, Christopher Dräger*, Alexander Zapletal**, Thomas Grechenig*
*Research Group for Industrial Software
Vienna University of Technology
Vienna, Austria
{raoul.vallon,christopher.draeger,thomas.grechenig}@inso.tuwien.ac.at
**Research Industrial Systems Engineering R&D
Schwechat, Austria
alexander.zapletal@rise-world.com

***Abstract*—We present a 15-month descriptive case study on a real-world Scrum process transformation from a single-site to a distributed development environment in a medium-sized software development organization in Austria. The study describes what effects the scaling to a distributed development had on several key process indicators in one of the organization's major projects. An action research approach has been selected to generate results from an in-depth and first-hand research setting. To increase objectivity and separation of concerns, a two-cycle approach, practitioner-oriented and research-oriented, has been established that aligns with sprint iterations. Many possible adaptations to the Scrum process have been tested over the course of the study. Key findings include that constant customer shipments after each sprint were a turning point in supporting the process of integrating the different sites in the distributed development environment and that the retrospective was an invaluable tool to keep frustration levels low in an ever-changing process environment.**

Keywords*— Agile Software Development, Distributed Software Development, Scrum, Descriptive Case Study, Action Research

I. Introduction

Agile software development has been of great research interest in the last decade [1]. While originally designed for collocated teams, agile practices have gained ground on distributed software development (DSD) environments in recent years as several studies have reported (e.g. [2,3,4]). However, the area is still under-researched. DSD environments are more complex and several adaptations to process models for collocated teams are necessary [2,4], e.g. communication between team members needs other mechanisms [5]. Other areas for improvement [6] include virtual teams, transparency, commitment, planning, estimation, predictability, self-organizing teams and tools. This paper contributes to the research field of agile DSD by describing the effects of transforming a former single-site Scrum team to a distributed development environment.

The selected *action research* (AR) approach enables an extensive case description. AR uses "a spiral of steps, each of which is composed of a circle of planning, action, and fact-finding about the result of the action" [7, p. 38]. The term action research has been coined by Kurt Lewin in 1946 [7]. AR is, however, close to non-existent in SE research [8,9,10] although it provides the most realistic research setting and thus enables the researcher to gain an in-depth and first-hand understanding [9]. Usually the researcher will incorporate one or several feedback cycles [11]. Scrum provides a variety of feedback cycles which can be utilized for AR. Hence, AR aligns very well with Scrum, as the problem solving cycle is already part of the process. A parallel second research cycle can be established without altering the original process.

This descriptive case study provides the following contributions:

1. Report on a real-world Scrum adaptation from a single-site to a distributed development environment over an extensive period of time (15 months)

2. Descriptive study enriched by first-hand experiences from an action research setting

3. Establishment of two separate feedback cycles for practitioner's and researcher's goals, respectively, as well as a declared-in-advance theoretical framework to provide scientific rigor

The outline of the paper is as follows. First we present the case study's background and provide an overview of related work. Section 4 explains the case study design in detail. Section 5 provides the analysis including challenges and how they were met. Section 6 discusses results with regard to related work, lessons learned and threats to validity. Section 7 concludes the paper and provides an outlook to future research.

II. Case Background

Software development on this product started as an R&D project at the organization *Research Industrial Systems Engineering* on a single site for 2 years that evaluated several technologies for the interplay of hardware devices and a web administration. This case study begins with the 15-months product development phase. Figure 1 shows the full background timeline including the history before the beginning of the cases study. Table II lists the changes in project specifics.

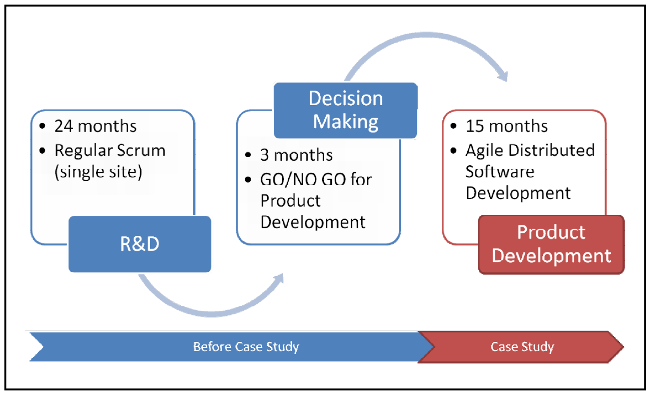

24 months
Regular Scrum (single site)
R&D
Decision Making
3 months
GO/NO GO for Product Development
15 months
Agile Distributed Software Development
Product Development
Before Case Study
Case Study

Fig. 1. *Case background timeline: The beginning of the case study period is separated from the R&D period by 3 months of management decision making on whether or not to proceed with full-fledged product development involving additional team members and the transformation to a distributed development environment.*

TABLE I. CHANGES TO THE PROJECT'S DNA

	R&D (before case study)	**Product Development (case study period)**
Development sites	1	2
Team members	9 (1 product owner, 1 scrum master, 6 developers and 1 tester)	19 (1 product owner, 2 scrum masters, 11 developers and 5 testers)
Time	24 months	15 months
Project Goal	Technology proof of concept and field test	Market-ready product (new feature development with focus on quality, performance and operational stability)
Process model	Scrum	Scrum extended for agile DSD

With the start of this 15-month case study period, the goal was to turn the R&D software status quo to a deliverable product. Additional team members had to be split over two sites due to office space constraints in the original R&D facility at the client. The two different development sites, *dev site* and *test site*, are located in the same city. Table II lists all project members with regard to role and site. The following Scrum roles were present: one product owner (PO) and two scrum master (SM).

TABLE II. TEAM SIZES DISTRIBUTED ACROSS TWO SITES.

Distributed Sites	**Dev**	**Test**	**SM**	**PO**	**Sum**
Dev Site	11	0	1	1	13
Test Site	0	5	1	0	6
Overall	11	5	2	1	19

Several fully distributed cross-functional teams, i.e. integrated over both sites, have been established, which is an effective approach according to Sutherland et al. [12].

Since this is an AR-based case study, one of the authors assumed the role of Scrum master at the test site during the whole case study period. The remaining authors supported the action researcher in the research and problem solving activities.

III. RELATED WORK

The case study relates to the following areas: agile distributed software development, distributed testing and the AR-based case study.

A. *Agile Distributed Software Development*

The research community agrees that Scrum has to be extended to work in DSD [3,14]. Sutherland et al. define three different types of distributed Scrum in [15]: Isolated Scrums (geographically isolated), distributed Scrum of Scrums (geographically isolated, but integrated by a regular Scrum of Scrums meeting) and fully distributed Scrums (cross-functional Scrum teams, team members geographically distributed). In our case we analyze a fully distributed Scrum implementation. However, fully distributed Scrum is not easily accomplished, as Vallon et al. show in their case study [2] of distributed Scrum involving two different organizations. Lessons learned include an increased effort for self-organizing teams in agile DSD. Based on the results of this study, the authors suggest cross-functional isolated teams instead of fully distributed ones.

Hossain et al. show that adapting Scrum helped practitioners overcome individual project challenges better than with previous plan-driven models in their multi-case study [3].

B. *Distributed Testing*

This paper deals with distributed testing due to the separation in dev site and test site. Collins et al. share experiences from a distributed testing team in an agile DSD environment in [16]. The authors conclude that distributed testing can function well in agile DSD if the following key lessons are considered: improved communication and coordination by allocating one person (test leader) as a link between teams, availability of key personnel in an online chat tool to clarify doubts, periodic physical meetings between development and test for building a common understanding of user stories.

Grechanik et al. argue in [17] that new approaches must be developed to deal with testing in DSD, including: traceability between different software artifacts, detection of semantic interference (merge conflicts), regression test selection and prioritization, coordination and awareness before a new revision is delivered.

C. *AR-based Case Study*

Dos Santos and Travassos report that AR studies represent only a small fraction in SE research [10] although AR is "the most realistic research setting" [9]. We have found two AR-based case studies that relate to our work in terms of case study design. Jolly et al. report an AR case in [18], which follows a standard case study routine: case description, related work, case study design, case study report (including lessons learned and threats to validity) and conclusion. We chose a similar design for this paper. Marshall et al.'s Action Research in practice case study [19] follows the dual imperatives of AR, as originally defined in [20]. The paper deals with finding a balance between the problem-solving and research interests. An overly strong focus on only one of the two can prevent a lasting solution for the other [19]. The dual imperatives approach is also part of this case study's design.

IV. Case Study Design

The research design is based on Yin [21], who established the case study as a rigorous and comprehensive research strategy including the logic of design, data collection techniques and specific approaches to data analysis. The theory is built upon the following focal points [21]: research question, propositions, unit of analysis, logic linking data to proposition (procedure) and criteria for interpreting findings. The case study design is based on [21] and action research design on [20]. AR is especially useful to investigate a phenomenon in its real-life context in depth.

The study is a descriptive study because it *attempts* to explore and explain, yet does not have the capacity to fully understand cause and effect in a broader context as would an explanatory study. There is yet no comprehensive process framework to fully cover what agile distributed software development is all about - on the contrary most research on the matter is still exploratory in nature with little possibilities for generalization. Hence, we devote this study to provide further insight into the matter and improve the description of characteristics of the phenomenon. It serves – among other related studies in the past years - as a basis for follow-up cases until enough evidence is collected to mature the field and conduct explanatory, probably multi-case, studies with generalizable results and implications of broader scope.

A. Research Question

The objective of the study is to enhance the body of knowledge of using agile development in DSD with a special focus on the transition from a collocated to a distributed Scrum implementation. We formulated the following research question (RQ):

RQ. *What are the effects of scaling Scrum from a single-site to a distributed software development environment and how can challenges be overcome?*

B. Declared-in-Advance Theoretical Framework

We address criticism on AR that the researcher is neither able to control processes and outcomes nor free to pick and choose problems [24] by providing a declared-in-advance theoretical framework as suggested by Blichfeldt and Andersen in [25]. To build a theoretical framework, we analyze the assumptions of regular Scrum supported by research on coordination theory in agile collocated environments [13] and compare where DSD violates those assumptions. As a result, we define up-front propositions prior to the execution of the study.

1) Assumptions for collocated Scrum: The agile manifesto [28] values "individuals and interactions", "working software", "customer collaboration" and "responding to change". Other more structured elements are mentioned in the agile manifesto but carry less weight: "processes and tools", "comprehensive documentation", "contract negotiation" and "following a plan". Strode et al. analyze in [13] how coordination is achieved in collocated agile development. Based on Strode et al.'s components, we assign the corresponding implementation in Scrum in brackets: *synchronisation activity* (daily scrum, sprint planning, sprint retrospective), *synchronisation artefact* (task board, sprint backlog, impediment list), *proximity* (daily scrum), *availability* (collocated teams), *substitutability* (agile generalist), *boundary spanning activity* (sprint review), *boundary spanning artefact* (task board, product backlog) and a *coordinator role* (scrum master). Their research shows that Scrum adresses these components very well in a collocated environment.

2) Violations of assumptions for collocated Scrum in DSD environments: Both the agile's values [28] and the components in agile coordination theory [13] rely heavily on the collocation of teams (especially: proximity, availability and synchronisation). Based on this rationale, we define the first a priori proposition as:

P1. *Since agile development has been designed for collocated teams, it has to be extended to work in DSD environments.*

By looking at collocated coordination theory that explains why Scrum works [13], we can see that most of the elements cannot be used in DSD without modification. Hence we define the second proposition as:

P2. *Agile DSD cannot achieve the same level of proximity, availability and synchronization as collocated development. Hence agile DSD does not strive to compete with collocated processes. It aims to find mechanisms to improve coordination by maintaining agile core values such as self-organization and synchronization, while acknowledging assets and drawbacks of distributing software development.*

DSD complicates direct human interaction which is a major concern for agile development. Hence, agile DSD needs tools to substitute missing face-to-face communication:

P3. *Due to the nature of distributed development, agile DSD needs to rely more heavily on processes, tools and documentation than in its collocated counterparts. In consequence, balancing the amount of structured and agile elements is a major concern in applying agile concepts to DSD.*

C. Unit of Analysis

"The essence of a case study, (...), is that it tries to illuminate a decision or a set of decisions: why they were taken, how they were implemented, and with what result" [22, p.12]. Hence, our unit of analysis is occurring problems and their corresponding decisions within the set time span of 15 months, i.e. from start of the product development phase in an agile DSD environment to a market-ready product.

D. *Procedure and Data Collection Strategy*

Based on [23], McKay and Marshall [20] propose a two-cycle feedback loop with one problem cycle to address the problematic situation and one research cycle to achieve scientific goals. This supports the separation of the dual imperatives of AR and enables scientific rigor. The stakeholders and participants own the problem-solving cycle, while the researchers own the research cycle [20]. Researchers and participants collaborate to meet respective goals.

Usually the researcher will incorporate one or several feedback cycles [11]. With Scrum we have the opportunity to include a feedback cycle each sprint. Hence, AR aligns very well with Scrum, as the problem-solving cycle is already part of the standard process. An additional research cycle can be added as pictured in Figure 2.

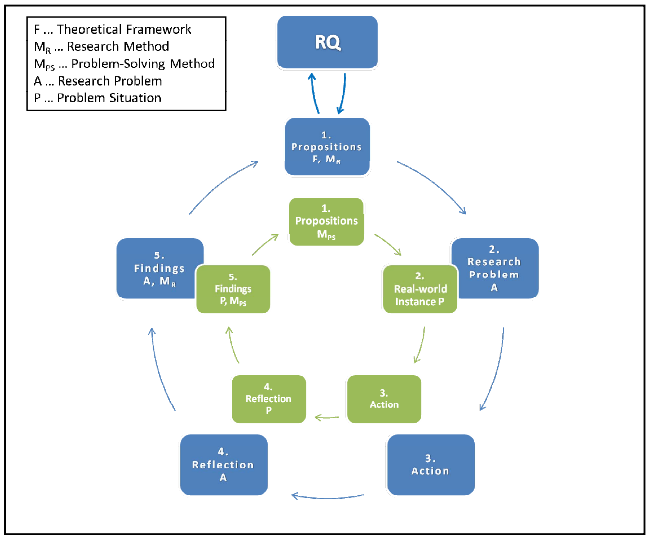


Fig. 2. *A dual imperative AR cycle (adapted from [20]): The inbound practitioners' cycle, designed to solve practical problems, provides input to the parallel outbound researchers' cycle, designed to gather knowledge on the research problem.*

The cycle starts with the research question RQ, a theoretical framework F based on propositions, a research method M_R and a problem-solving method M_{PS}. We have defined a research problem A, namely the possibilities, challenges and solutions of transforming single-site Scrum to a DSD. In our case study we analyze P, which is a real-world instance of our research problem A. By evaluating actions on P, we strive to derive results for A.

The problem-solving method M_{PS} is the regular Scrum process, where decisions on the process are taken and evaluated in the retrospective meeting at the end of each sprint in the real-world problem situation P. The research method M_R reflects on the problems and decisions (actions) in a separate cycle with regard to research problem A. This parallel two-cycle process enables the separation of problem-solving and research interests. The steps of the dual AR cycle of Figure 2 are further explained in Table III.

TABLE III. THE DUAL ACTION RESEARCH CYCLE AND ITS IMPLEMENTATION IN SCRUM AS USED IN THIS STUDY

Cycle Steps	**Problem-solving Activity**	**Research Activity**
0. Definition of RQ	-	Define RQ
1. Definition of F,M_R,M_{PS}	(Re-)Define M_{PS} (Sprint)	(Re-)Define F, M_R
2. Update problems	Update problems of P (Daily, Retro)	Update observations of A
3. Take action	Act on problems of P (Sprint)	Collect and analyze data
4. Reflect on success	Reflect on success of actions (Retro)	Draw conclusions for A
5. Update findings	Update findings to P, M_{PS} (Retro)	Update findings to A, M_R
6. Proceed with step 1 until exit criterion is met		

A well defined process is very important to establish recoverability [23,10]. Since this a descriptive case study, our exit criterion is time-boxed, i.e. the end of the product development phase.

Data collection for research involves the following activities: explore possibilities for accessing project data in the issue tracking tool (beginning of study), update project diary with field notes and photos (daily), extract data from issue tracking tool (each sprint), discuss problems and solutions with practitioners and record actions (sprint retrospective as well as informal discussions during sprint, possible due to action research setting), track results of sprint planning/review to analyze teams' performance as well as results of sprint retrospective to collect problems and solutions (each sprint).

E. *Criteria for Interpreting Findings*

This study will incorporate both qualitative and quantitative methodologies to broaden and strengthen the study [26]. The metrics described in Table IV are used to interpret findings and evaluate answers to the research question, as they are significant to the collaboration of the dev and test site. Bug count and release count can be extracted from the issue tracking tool. Impediments are maintained in a wiki and need to be processed manually by the authors.

TABLE IV. METRICS IN USE

Metric	**Description**
Bug Count	Number of bugs
Release Frequency	Frequency of bug fix or feature releases
Impediments Count	Number of impediments

V. ANALYSIS

We analyze 27 sprints over the course of 15 months. The sprint length was two weeks, although some sprints were prolonged to three weeks to cope with holidays or adjust to upcoming customer's deadlines. Due to the descriptive nature of the study, we describe changing characteristics of the agile DSD process over the course of the study. In retrospective, we found several major phases regarding progress in the integration of the two sites, which we present consequently.

A. *Kick-Off: Isolated and Unsynchronized Sites*

The **initial process** at the start of the case study period was **fully focused on the dev site**, since the test site was the additional new component that was added to the project after the two year R&D phase. We expected problems due to the R&D history, which all of the dev site members were part of. The dev site worked in sprints without the direct involvement of the test site, which resulted in **test-ready features at the end of the sprint instead of deployment-ready ones**. The test Scrum master took part in the following Scrum meetings on behalf of the test team: sprint planning (first day of sprint), review and retrospective (both at last day of sprint). The rest of the sprint no meetings were scheduled. **Specification was written at dev site** and testers used it to deduce test cases. This procedure also served as a validity and sanity check of requirements. The goal was to have **test cases (i.e. a double-checked specification) ready before stories** could be pulled into a sprint, which was **rarely accomplished**. The specification consisted of a written document accompanied by design mock-ups. Test cases were written in Gherkin language to support behavior driven development [27]. Resulting scenarios could either be used for manual testing or selected for test automation.

Daily Scrum meetings were skipped due to the separation in micro teams where people worked very closely and also in pair programming at times. Figure 3 shows a typical distribution of team members during this case study. A micro team would typically consist of 2-3 developers and 1 tester working on one feature. Since these **developers needed to collaborate very closely in micro teams** to implement a story within these micro teams, **daily Scrums were regarded as superfluous**. In reality, a tester would write test cases according to available specification and test the story once it is delivered without regular communication with developers.

Other problems not directly related to the distribution of sites include **low release frequency**, **definition of done not well defined**, **stories too big to fit in one sprint**, **highly volatile specification resulting in frequently necessary rework of test cases** (overhead) and **legacy stories from R&D project badly documented**.

B. *Progress: First Steps towards Cross-functional Teams*

The process implementation led to various serious problems: A story was regarded as accepted once it was developed. After that it was released for test, but the **focus was on implementing new stories instead of fixing "old" ones**. As a direct consequence a customer deployment failed badly due to the **low quality of software**. That was the turning point after sprint 7 to **include the test site in the process** more properly. Release frequency also increased: the interval for bug fix releases increased to three days and features were released as soon as they were finished.

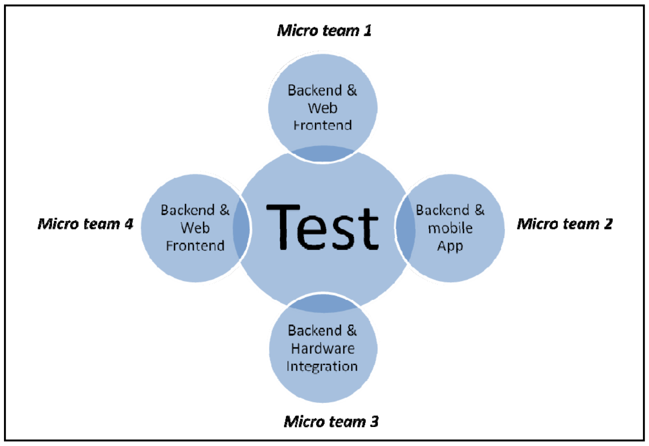
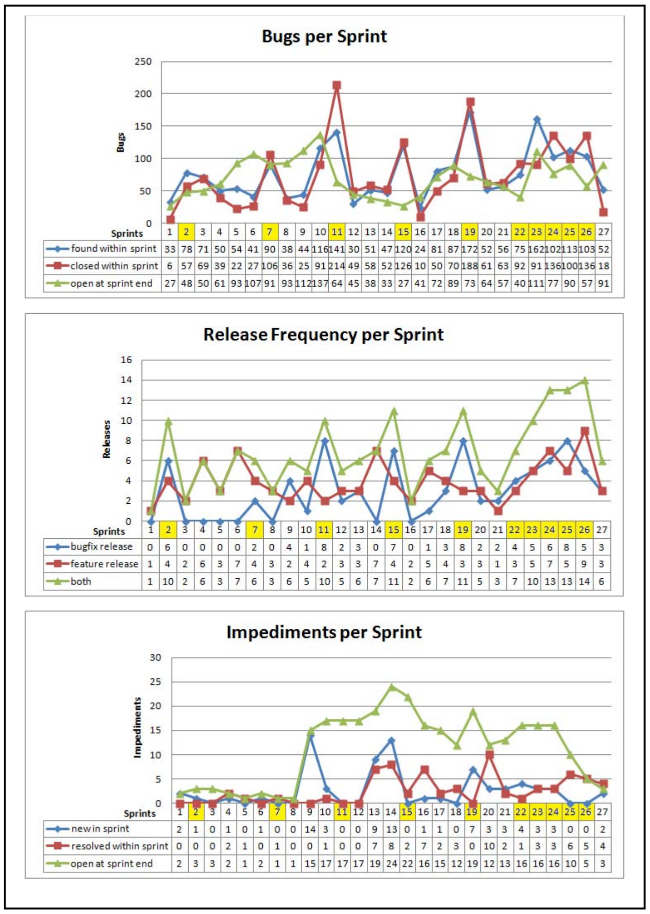


Fig. 3. *Separation in four Scrum micro teams: The split in micro teams was necessary due to a very specialist environment with very different components (web, app and hardware devices) interacting in a security-critical system. The figure shows the good case scenario towards the end of the case study. In the first few months of the case study the test was isolated and formed a component team.*

However the process was still inefficient and did not produce the necessary quality in software, as the **steadily increasing bug count** shows and the **jump in the impediment count** in sprints 9 and 10 (cf. Figure 4). Reasons were a **huge technical complexity in the interplay of hardware devices and software** that had been underestimated. As a result, **a dedicated bug fixing iteration** had to be conducted, in which over 200 bugs were closed, to improve quality for the customer shipment in sprint 11.

C. *Achievement 1: Establishing Cross-functional Teams*

Further measures were taken to improve quality in a continuous process without the need for bug fixing iterations. **Contact visits increased**, especially in the **second week of each sprint the test team joined the dev team at the dev site for an intense feature testing and bug fixing session**. To make room for the testers, some developers moved either to the test site or work in home office for these days. A **better common understanding that bugs need to be fixed within the sprint** has been established. The **test site** became an integral part of the process and **has the right to reject stories**.



Sprints	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
found within sprint	33	78	71	50	54	41	90	38	44	116	141	30	51	47	120	24	81	87	172	52	56	75	162	102	113	103	52
closed within sprint	6	57	69	39	22	27	106	36	25	91	214	49	58	52	126	10	50	70	188	61	63	92	91	136	100	136	18
open at sprint end	27	48	50	61	93	107	91	93	112	137	64	45	38	33	27	41	72	89	73	64	57	40	111	77	90	57	91

Sprints	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
bugfix release	0	6	0	0	0	0	2	0	4	1	8	2	3	0	7	0	1	3	8	2	2	4	5	6	8	5	3
feature release	1	4	2	6	3	7	4	3	2	4	2	3	3	7	4	2	5	4	3	3	1	3	5	7	5	9	3
both	1	10	2	6	3	7	6	3	6	5	10	5	6	7	11	2	6	7	11	5	3	7	10	13	13	14	6

Sprints	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
new in sprint	2	1	0	1	0	1	0	0	14	3	0	0	9	13	0	1	1	0	7	3	3	4	3	3	0	0	2
resolved within sprint	0	0	0	2	1	0	1	0	0	1	0	0	7	8	2	7	2	3	0	10	2	1	3	3	6	5	4
open at sprint end	2	3	3	2	1	2	1	1	15	17	17	17	19	24	22	16	15	12	19	12	13	16	16	16	10	5	3

Fig. 4. *Bug, release and impediment count measured per sprint: sprints 2, 7, 11, 15, 19 and 22-26 denote shipments to the customer. In these sprints we can see a rise in release frequency and closed bugs (positive) but also a rise in new impediments (negative), which shows that these sprints put the process to a test. The regular shipments starting with sprint 22 allowed for a more continuous flow.*

Although the integration of the two sites was improved, testers had to reject many stories because non-functional aspects such as performance and stability were not valued in the commitment. This behavior led to **many stories that were only 80% finished** according to the testers and thus rejected. **Transparency and focus decreased drastically**. These huge problems set the incentive to further improve the process.

D. Achievement 2: Establishing a Continuous Flow

Our analysis showed that **a continuous flow could not be achieved so far**: bug count and impediment count steadily increased in between customer shipments and only dropped shortly before the shipment, usually in the last sprint, in a last minute effort to improve quality. Based on our findings we proposed to **drastically shorten intervals of customer shipments to each sprint**, starting with sprint 22. It **helped to put greater effort into a more realistic sprint planning and commitment and improve quality as a consequence – with the customer shipment always in mind**. The metrics show a continuously high release frequency and fewer impediments in sprints 22-26. The **bug count was also more stable**. The relatively low "closed bugs" count in the last sprint was due to the fact that only "hot fix" bugs have been resolved for the market release.

E. Achievement 3: Keeping Team Spirits High and Maintaining the Willingness to Change

Both top management and team members knew it would be hard to establish a working distributed development environment. During the 15-months of agile DSD, **the retrospective turned out to be an invaluable tool, especially at high-stress times, to keep process improvement going and thus keep frustration levels low** as all team members could speak their mind and propose solutions.

The building of trust was an important prerequisite for the success of the process improvement. When trust was not established in the beginning, finger pointing (between the two sites) was more common than in later sprints. Trust was established with **contact visits and the online availability of all team members (instant messaging and email)**. **Instant messaging became the equivalent to face-to-face coordination that is available only to collocated teams**.

F. Challenges while Performing the Analysis

In sprint 9 the issue tracking tool was changed due to its limited querying capacities. This posed a threat to the case analysis as old data had not been migrated from one tool to the other. Hence, reporting facilities of neither tool could be used as they only represented part of the whole time span. For the sake of a complete analysis, we extracted data from both tools to a spreadsheet.

VI. Discussion

In this longitudinal 15-month case study many problems occurred, decisions have been made and actions have been taken. We strive to answer the following research question:

RQ. *What are the effects of scaling Scrum from a single-site to a distributed software development environment and how can challenges be overcome?*

Table V shows an extensive overview of practical problems and decision taken and their learning for the research problem. We will discuss our findings *A1-A7* (cf. Table V) with regard to our a priori propositions:

P1. *Since agile development has been designed for collocated teams, it has to be extended to work in DSD environments.*

We find evidence for *P1* in all findings *A1-A7*. Extensions to the process include (for a full list, cf. Table V):

- Instant messaging as an equivalent to face-to-face communication
- Contact visits to build and maintain trust among team members from different sites
- Extensive use of electronic issue tracking tools to keep track of updated requirements

- Keep documentation up-to-date (including relevant informal enquiries) to share knowledge across the sites

P2. *Agile DSD cannot achieve the same level of proximity, availability and synchronization as collocated development. Hence agile DSD does not strive to compete with collocated processes. It aims to find mechanisms to improve coordination by maintaining agile core values such as self-organization and synchronization, while acknowledging assets and drawbacks of distributing software development.*

The case study shows that agile DSD cannot compete with collocated Scrum. It is slower in adapting to problems (*A7*) and in keeping team members synchronized (*A1*). We found several mechanisms to bring agile values to agile DSD (see *P1* and Table V).

P3. *Due to the nature of distributed development, agile DSD needs to rely more heavily on processes, tools and documentation than in its collocated counterparts. In consequence, balancing the amount of structured and agile elements is a major concern in applying agile concepts to DSD.*

We experienced a greater need for both up-to-date documentation in agile DSD (*A4*) and electronic tools (*A2*) which is underlined by the necessity to track results from informal enquiries and meetings (*A5*). Documentation and the use of electronic tools also help with synchronizing the sites (*A1*). The findings from the case study support the initial proposition.

Apart from the propositions, we found *A3* and *A6* to be important contributions to the improvements of the process in case study.

A. Case Study Validity

We set the study on a robust theoretical framework by examining related work and creating propositions at the design phase to establish external validity within a single-case study [21]. Internal validity was achieved using an adapted AR approach and a construct table (Table V) to show causal relationships. We increased construct validity by using both qualitative and quantitative data and had key team members (Scrum master, senior developer) review the paper draft. To achieve reliability we maintained a project diary during the whole 15 months period and kept records of collected data.

B. Limitations

Threats to validity are introduced by the fact that this is a single case study. Other case studies may produce different results. We aimed to reduce investigator bias by following an adapted two-cycle AR approach.

C. Evaluation against Related Work

Stettina et al. [29] research ways of handling documentation in agile teams. The authors conclude that documentation is often seen as a burden in agile teams as it does not directly contribute to the development of software. In DSD, however, it is very important to maintain documentation for knowledge sharing.

Vax et al. also conclude in [30] that you need the right expertise and team for distributed Scrum. We support that claim as it took over 20 sprints to establish a working process.

Paasivaara et al. mention the possibility of having an "ambassador/rotating guru" in [31], who is sent to other sites for a longer period of time. We saw several key roles pay contact visits instead of permanent rotations.

In related publications we can see a growing interest in bringing agile to (globally) distributed software development [32], [33]. One of the conclusions over several case studies reviewed by Hossain et al. in [32] is that Scrum needs to be extended to work in a distributed setting, which corresponds to our proposition P1.

VII. Conclusion

In this paper we presented a longitudinal 15-month case study based on action research. By implementing a parallel feedback cycle we were able to separate actual problem solving from generating research outcomes. The case study provided various insights into the complexities involved with transforming Scrum to distributed environments. Adaptations to the process take longer to take effect because solutions need to be propagated across several sites. Synchronization of team members needs mechanisms such as instant messaging and contact visits to substitute on-site availability. Documentation and knowledge management is more important in agile DSD since even informal enquiries (between a developer and the product owner e.g.) need to be communicated to the other site. The retrospective served as a mood barometer for team members of both sites and was a great driver for process improvement. In the last months of the case study, software has been successfully shipped to the customer each sprint, which underlines the working process adaptations for the given distributed project environment.

Future research interest includes the exploration of agile DSD in further projects to be able to enhance and test findings of this study and find recurring problems and solutions.

TABLE V. RESULTS SUMMARY.

Problem P	**Root Cause**	**Decision**	**Agile DSD Learning A**
Test-ready features at the end of the sprint instead of deployment-ready ones	• Isolation of test site • Focus on dev site	• Give test site the right to reject stories • Increase contact visits (for face-to-face specification enquiries) • Review of test cases for critical user stories by a developer • Constant availability of team members in instant messaging	**A1.** ***Fully distributed cross-functional teams need to synchronize and work towards the same goal in a sprint. Synchronization can be achieved with contact visits, instant messaging, an up-to-date ticket management system and test case or code review.***
Transparency for both sites not accomplished	• Dev site members did not keep issue tracking system up-to-date • Commitment not met (waste in commitment) • Overhead due to rising bug count	• All relevant informal communication needs to be appended to the feature ticket in the issue tracking tool • Contact visits to improve trust and team spirit	**A2.** ***Informal face-to-face communication needs substituation in DSD. Tool support can help keep track of changes in requirements. Instant messaging can become the equivalent to face-to-face coordination that is available only to collocated teams.***
Low software quality Lack of focus in sprint	• Feature rush (many 80% ready features valued higher by product owner than fewer 95% ready ones) • Huge technical complexity in the interplay of hardware devices and software • Definition of done not well defined • No pressure to deliver potentially shippable code	• Incremental inclusion of test site • Bugfix iteration (should be avoided) • Sunshine cases should work when a story is passed to the tester, so the tester can focus on corner cases • Build all aspects of a story including non-functional requirements (e.g. stability, performance) • Continuous deployment to customer every sprint	**A3.** ***Continuous customer deployments can help establish focus in DSD and lead to more releastic sprint plannings and commitments. Continuous flow means more reliable estimations and higher software quality.***
Volatile specification	• Legacy stories from R&D project not defined • Stories not detailed enough before sprint • Stories written at dev site only • Stories too big • Volatile specification	• No informal story updates • Meetings with customer before sprint • Involve customer more in the prioritization • Small managable stories • Increased up-front planning and specification to identify problems, corner cases and impact on existing software early and improve estimation of team • Staffing: new Business Analyst	**A4.** ***While up-to-date documentation does not substitute direct interaction, it plays a bigger part in agile DSD than in on-site Scrum since direct communication is harder to manage.***
No up-to-date test cases	• Test cases not ready before stories get pulled into sprint • Specification wrongly interpreted • Informal specification adaptations without the other site's knowledge • Wrong effort estimation due to open questions in specificaton	• No story updates during sprint • On-demand specification meetings with members from both sites	**A5.** ***Results from informal on-site enquiries or meetings need to be made available to other sites, either in terms of updated documentation or updated tickets in the electronic ticket management system. Contact visits can further be used to review test cases for critical stories.***
Process adaptation in DSD is slower and more difficult than in regular collocated Scrum	• Harder to propagate changes over multiple sites • More variables and complexities to take into account	Use retro as a driver for continuous process improvement	**A6.** ***Keeping the retrospective as a constant in an ever-changing process environment provided a valuable asset. The retrospective meeting gathers members from all sites to reflect on the process and discuss improvements. It further functions as an important mood barometer across all sites.*** **A7.** ***Process adaptation in DSD is slower and more difficult to achieve than in regular collocated Scrum.***

References

[1] Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: Towards explaining agile software development. JSS 85(6), 1213–1221 (2012)

[2] Vallon, R., Strobl, S., Bernhart, M., Grechenig, T.: Inter-Organizational Co-Development with Scrum: Experiences and Lessons Learned from a Distributed Corporate Development Environment. In: Proceedings of the 14th International Conference on Agile Software Devel-opment, XP 2013, Vienna, Austria, pp. 150-164 (2013)

[3] Hossain, E., Bannerman, P.L., Jeffery, R.: Towards an Understanding of Tailoring Scrum in Global Software Development: A Multi-case Study. In: Proceedings of the 2011 International Conference on Software and Systems Process, ICSSP 11, Honolulu, HI, pp. 110-119 (2011)

[4] Batra, D.: Modified Agile Practices for Outsourced Software Projects. Communications of the ACM 52(9), 143-148 (2009)

[5] Dorairaj, S., Noble, J., Malik, P.: Effective Communication in Distributed Agile Software Development Teams. In: Proceedings of the 12th International Conference on Agile Software Development, XP 2011, Madrid, Spain, pp. 102-116 (2011)

[6] Vallon, R., Bayrhammer, K., Strobl, S., Bernhart, M., Grechenig, T.: Identifying Critical Areas for Improvement in Agile Multi-Site Co-Development. In: Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2013, Angers, France, pp. 149-156 (2013)

[7] Lewin, K.: Action research and minority problems. Journal of Social Issues 2(4), 34-46 (1946)

[8] Glass, R.L., Vessey, I., Ramesh, V.: Research in Software Engineering: An Analysis of the Literature. Information and Software Technology 44(8), 491-506, (2002)

[9] Sjøberg, D., Dybå, T., Jørgensen, M.: The Future of Empirical Methods in Software Engineer-ing Research. In: Proceedings of the 2007 Workshop on the Future of Software Engineering, ICSE 2007, Minneapolis, MN, pp. 358-378 (2007)

[10] Santos, P., Travassos, G.H.: Action research use in software engineering: An initial survey. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, Lake Buena Vista, FL, pp. 414-417 (2009)

[11] Davison, R.M., Martinsons, M.G., Kock, N.: Principles of Canonical Action Research, Info Systems Journal 14, 65–86 (2004)

[12] Sutherland, J., Viktorov, A., Blount, J., Puntikov, N.: Distributed Scrum: Agile Project Man-agement with Outsourced Development Teams. In: Proceedings of the 40th Hawaii Interna-tional Conference on System Sciences, HICCS-40, pp. 274a (2007)

[13] Strode, D.E., Huff, S.L., Hope, B., Link, S.: Coordination in co-located agile software development projects, Journal of Systems and Software 85, 6, 1222-1238 (2012)

[14] Paasivaara, M., Durasiewicz, S., Lassenius, C.: Distributed Agile Development: Using Scrum in a Large Project. In: Proceedings of the 3rd IEEE International Conference on Global Soft-ware Engineering, ICGSE 2008, Bangalore, India, pp. 87-95 (2008)

[15] Sutherland, J., Schoonheim, G., Kumar, N., Pandey, V., Vishal, S.: Fully Distributed Scrum: Linear Scalability of Production between San Francisco and India. In: Proceedings of the 2009 Agile Conference, AGILE 2009, Chicago, IL, pp. 277-282 (2009)

[16] Collins, E., Macedo, G., Maia, N., Dias-Neto, A.C.: An Industrial Experience on the Applica-tion of Distributed Testing in an Agile Software Development Environment. In: Proceedings of the 2012 IEEE Seventh International Conference on Global Software Engineering, ICGSE 2012, Porto Alegre, Rio Grande do Sul, Brazil, pp. 190-194 (2012)

[17] Grechanik, M., Jones, J.A., Orso, A., van der Hoek, A.: Bridging gaps between developers and testers in globally-distributed software development. In: Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, Santa Fe, NM, pp. 149-154 (2010)

[18] Jolly, S.A., Garousi, V., Eskandar, M.M.: Automated Unit Testing of a SCADA Control Software: An Industrial Case Study Based on Action Research. In: Fifth International Confer-ence on Software Testing, Verification and Validation, ICST 2012, Montreal, Canada, pp. 400-409 (2012)

[19] Marshall, P., Willson, P., de Salas, K., McKay, J.: Action Research in Practice: Issues and Challenges in a Financial Services Case Study. The Qualitative Report 15(1), 76-93 (2010)

[20] McKay, J., Marshall, P.: The dual imperatives of action research. Information Technology & People 14(1), 46-59 (2001)

[21] Yin, R.K.: Case study research. Sage Publishing (2009)

[22] Schramm, W.: Notes on case studies of instructional media projects. Working paper for the Academy for Educational Development (1971)

[23] Checkland, P., Holwell, S.: Action Research: Its Nature and Validity. Systemic Practice and Action Research 11(1), 9-21 (1998)

[24] Baskerville, R.L., Lee, A.S.: Distinctions among different types of generalizing in information systems research. In: Proceedings of the IFIP TC8 WG8.2 International Working Conference on New Information Technologies in Organizational Processes: Field Studies and Theoretical Reflections on the Future of Work, St. Louis, MO, pp. 49-66 (1999)

[25] Blichfeldt, B.S., Andersen, J.R.: Creating a wider audience for action research: Learning from case-study research. Journal of Research Practice 2(1), D2 (2006)

[26] Yin, R.K.: Mixed Methods Research: Are the Methods Genuinely Integrated or Merely Paral-lel? Research in the Schools 13(1), 41-47 (2006)

[27] North, D.: Behavior Modification. The evolution of behavior-driven development. Better Software Magazine, March Issue (2006)

[28] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001. Manifesto for agile software development. Retrieved 24 February 2014 from http://www.agilemanifesto.org

[29] Stettina, C.J., Heijstek, W., Faegri, T.E.: Documentation Work in Agile Teams: The Role of Documentation Formalism in Achieving a Sustainable Practice. In: 2012 Agile Conference, pp. 31-40 (2012)

[30] Vax M., Michaud S.: Distributed Agile: Growing a Practice Together. In: 2008 Agile Conference, pp. 310-314 (2008)

[31] Paasivaara, M., Durasiewicz, S., Lassenius, C.: Using Scrum in Distributed Agile Development: A Multiple Case Study. In: 4th International Conference on Global Software Engineering, pp. 195-204 (2009)

[32] Hossain, E., Ali Babar, M., Paik H.: Using Scrum in Global Software Development: A Systematic Literature Review. In: 4th International Conference on Global Software Engineering, pp. 175--184 (2009)

[33] Jalali, S., Wohlin, C.: Agile Practices in Global Software Engineering – A Systematic Map. In: 5th International Conference on Global Software Engineering, pp. 45-54 (2010)