Exploring the Evolution of Software Practices

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

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

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

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

Allan Ebdrup
Copenhagen, Denmark
allan@878.dk

ABSTRACT

When software products and services are developed and maintained over longer time, software engineering practices tend to drift away from both structured and agile methods. Nonetheless, in many cases the evolving practices are far from ad hoc or chaotic. How are the teams involved able to coordinate their joint development? This article reports on an ethnographic study of a small team at a successful provider of software as a service. What struck us was the very explicit way in which the team adopted and adapted their practices to fit the needs of the evolving development. The discussion relates the findings to the concepts of social practices and methods in software engineering, and explores the differences between degraded behavior and the coordinated evolution of development practices. The analysis helps to better understand how software engineering practices evolve, and thus provides a starting point for rethinking software engineering methods and their relation to software engineering practice.

CCS CONCEPTS

• **Software and its engineering** → **Agile software development**; **Programming teams**; • **Human-centered computing** → **Computer supported cooperative work**.

KEYWORDS

Software Processes, Agile software development, Cooperative and human aspects of software engineering, Empirical software engineering

ACM Reference Format:
Yvonne Dittrich, Christian Bo Michelsen, Paolo Tell, Pernille Lous, and Allan Ebdrup. 2020. Exploring the Evolution of Software Practices. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3368089.3409766

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

1 INTRODUCTION

When software products and services are developed and maintained over longer time, software engineering practices tend to drift away from both structured and agile methods. Lous et al. analyzed software development practices that have evolved over time to address the challenges of continuous software engineering [16] and distributed development [17]. A survey-based investigation into industrial software development reveals the widespread use of hybrid develoipment methods [36].

In many cases, the reported software development practices are far from ad hoc or chaotic. Though there have been a number of empirical studies that document that companies use or do not use particular methods (see [4, 18, 26, 37] as examples), so far there has been no research into how these situated development practices come about, or whether an observed practice is the result of careful development or an accidental deterioration. The research questions the article addresses thus are: How can the evolution of software development practices be investigated? What distinguishes a software process that has degenerated from one that has evolved as the result of conscious deliberation?

We had the possibility to study a distributed software engineering team developing business software as a service. The development practices did not resemble any method, but both successfully delivered software and was satisfying for the team members. We decided on an ethnographic study as our research method, as we were interested in understanding the team's continuous software development practices from a members' point of view. As the method section below details, the early part of the fieldwork indicated that one important aspect of the success and high satisfaction was that the team continuously developed and groomed the way they developed software.

In the field work, we were able to follow the evolution of the teams' way to implement stand-up meetings as a concrete example. In the thematic analysis (section 5.3), we refer to a number of practices that were evolved by the team prior to our study. The focus on the evolution of the stand-up meetings allowed us to follow the evolution of a specific practice life, and to unfold a related set of *practices of changing practices*. The analysis identifies the implementation of bottom-up, situated software process improvement as established practices that is based on joint reflection and experimentation as core ingredients. This, in turn, requires the team to demonstrate responsibility and self-organization.

The concept of software development practices is at the center of our empirical research. To clarify this core concept, we reference social theory to develop the concepts of social practices [5, 28, 39], meta-work, and articulation work [10, 33] as theoretical underpinnings. Section 3.2 develops the conceptual framework we apply.

The article's contribution is a theoretical one based on empirical research (see also [32]). We show how concepts derived from social theory allow an understanding of specific observations. The observations illustrate the theoretical concepts and thereby demonstrate their value. Further, the theoretical concepts allow us to identify aspects of our observations that might be transferable to other contexts. With this contribution, we address Herbsleb's request for theoretical foundations for research of cooperative aspects of software engineering [13].

To motivate the related work and the theoretical underpinnings, the next section introduces the company we studied, and highlight some of the observations that determined this article's focus. It introduces the company's daily stand-up meetings and their evolution over time as the main example. Section 3 introduces the related work on the relationship between methods and practice in software engineering, and develops the theoretical underpinning. Then, we present the research method. Section 5 presents the analysis, which is then discussed in section 6. In the conclusion, we summarize our findings, and argue for the further development of a practice theory for software engineering, to better understand how methods and practice relate.

2 SOFTWARE DEVELOPMENT AT DB

DB is a Danish SME that develops invoicing software for web applications and native mobile applications. DB's journey started in 2012 with a reorganization of the mother company. In 2015, the current company was founded to focus solely on the current software product provided as a service. By the time we ended the study, DB had 40 employees, 12 of which were software developers. The team is internationally distributed, with two main offices in Denmark and Ukraine, and two additional developers working from home. At the time of our research, DB had developed a company-specific practice of continuous software engineering based on a stable deployment pipeline. The team was tightly integrated by means of a set of shared practices and a tailored set of communication channels.

This article focuses on how the developers evolve their practices, and uses the stand-up meeting as its example. As the team continuously experiments with, and improves its practices, any description of a practice is only a snapshot. This becomes evident when looking at the history of the stand-up meeting.

For the stand-up meeting, the entire development team joins a Slack conference call at 9.35, Mondays through Thursdays each week, and opens a shared slide set on Google Drive. The slide set includes standard information pulled by a robot: information about tasks and availability from the calendar, information about merge issues from the continuous integration server, bug reports that the Product Owner (PO) highlights as urgent, and information about production site errors. This standard set of slides is followed one slide for each team member, to share information about current tasks. Under the heading, 'Anything to add,' team members may add slides to raise topics they deem important to discuss with the team.

When we began our research, the stand-up meeting format had

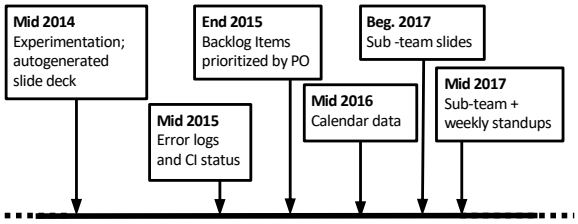

Figure 1: Timeline over changes to stand-up meeting format.

evolved from a series of experiments in 2014, when the previous format was considered boring and ineffective. (See figure 1 for a rough overview.) The developers then paired up, and each pair experimented with the stand-up format for one week. The task was to *"Find out how you would like to hold the stand-up meeting. It just must not be longer than 15 min, everybody must participate, and it must be fun."* These experiments went on for six or seven weeks. The decision was made to keep the simplest format, and to move the meeting to an audio channel with Google slides, instead of using a screen-sharing functionality that was considered time-consuming to set up. The empty slide deck was created through a custom-written scheduled service called 'Stand-up as a Service'. Slowly, other bits and pieces were added: mid-2015, the CTO extended the slide deck and the script to present the error log and the build status from the continuous integration system, to address a recurring frustration. *"Someone gets the crummy job to tell people, 'now you shall take this bug.' And that is a lousy thing to be left with. [...] For a long time, it was me and one of the others, who was concerned that the built should not be red, who pinged people. And that is really annoying."* Over time, the team adjusted the script so that only errors that appeared with a certain frequency were included. By the end of 2015, the presentation of urgent back-up items was introduced, based on the PO's initiative. The last major change before our field work started was the introduction of a slide that presented data from pulled individual calendars, regarding absences and team meetings, to provide orientation for the team members.

When we were doing our research, DB's team had started to work as several sub-teams, for example, one group focused on developing a mobile client. The teams developed additional coordination practices. One of the sub-teams decided, for example, to have additional sub-team stand-up meetings. This did not go unrecognized, and resulted in a discussion of the stand-up meeting in the retrospective that provides the core of our analysis.

3 RELATED WORK

The related work relevant to this article brings together two separate scientific discourses. Subsection 3.1 presents the literature on the relationship of methods to software development practice. Subsection 3.2 provides the theoretical underpinning necessary for a deeper understanding of our observations. It introduces a set of concepts from social theory, and relates it to empirical research on software engineering.

3.1 Methods in Software Engineering

Though methods are at the core of software engineering, the role of methods in relation to what actually goes on in day-to-day development practices has been contested from the very beginning. Whereas Osterweil's well-known article, 'Software Processes are Software Too' [24] already argued for the expression of software development methods as programs, in order to automate the software development as much as possible, in 'Programming as Theory Building' Naur argues that methods – understood as 'sets of work rules for programmers, telling what kind of things the programmers should do, in what order, which notations and languages to use, and what kinds of documents to produce in various stages' [22] – are incompatible with the theory-building view of programming, which emphasises the development of an understanding of the problem, and a piece of software to address it.

Though Osterweil's view has influenced software engineering research and practices, for example, as the widely applied CMM(I) [34] and similar frameworks, it is continuously contested: For example the Agile Manifesto [2] proposes valuing *Individuals and Interaction over Processes and Tools*. Similarly, Naur's article resonates well with software engineers; however, the same developers welcome and happily explore new methods.

Though quite a few articles show how – for good reasons – methods are adapted to a specific context, rather than going by the book (see [4, 18, 26, 37] as examples), the relationship between methods as descriptions and the practices that they are meant to inform and improve is, by and large, unresearched. Mathiassen et al. [19] observe that experienced practitioners do not refer to the method descriptions on a daily basis; they do not follow the methods very closely, but competently select and implement relevant elements. Fitzgerald argues for distinguishing *formalized methods* and *methods-in-action*, where the method-in-action describes the structured way in which development that takes place in practice. "[I]t is suggested that methodologies are never applied exactly as originally intended. Different developers will not interpret and apply the same methodology in the same way; nor will the same developer apply the same methodology in the same way in different development situations. Therefore, on any development project, the methodology-in-action is uniquely *enacted* or *instantiated* by the developers" [7]. Methods are adapted and tailored to address a specific development challenge [8]. As Floyd states, "We do not apply predefined methods, but construct them to suit the situation at hand. [...] What we are ultimately doing in the course of design is developing our own methods" [9]. Such method adaptation and design practices have also been observed. For example, Sigfridsson reports on how the PyPy community adjusts its sprints to take care of new developers joining the community [31]. Giuffrida and Dittrich show how successful distributed student teams deploy an unstructured social software channel to adjust their processes [11, 12]. This is also supported by the results of a recently published survey that indicates a loose correlation between the methods or development models that are officially used, and what is implemented in software development practices [36].

Both traditional and agile development include elements of reflection and learning, including project postmortems and retrospectives. Principle 12 of the Agile Manifesto explicitly stipulates that 'At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly'[2, principles]. However, there is surprisingly little research on what takes place during such meetings and why and how the teams practices are adapted as a result. In a grounded theory study, Andriyani et al. present a categorisation of what is discussed in retrospectives [1]. Others propose the use of these reflective meetings for software process improvement [27], or propose formats to improve their effectiveness [3]. In our study, we not only report what is discussed in retrospectives, but we show how retrospectives and related practices are used by DB's team to continuously improve their practices.

3.2 Practice Theory for Software Engineering

Whereas the notion of 'methods' is well-established in software engineering, in most cases, the term 'practice' used in its colloquial sense. However, practice is a core concept of social theory. To discuss the relationship between methods and software practices, we draw on the theoretical framework based on social theory developed in 'What does it mean to use a method? Towards a practice theory for Software engineering' [6]. In this article, Dittrich argues that the problematic relationship between methods and practices might be due to our lack of understanding of software engineering practices as social practices. She suggests adopting concepts from the philosophy of sociology that have proven successful helping to understand computer-supported cooperative work [29], and that are based on Wittgenstein's Philosophical Investigations [39]. Here, we summarize the parts of her argumentation that we use in our analysis and discussion of the data.

In the pragmatic tradition, social practices are seen as constitutive for understanding the development, tradition, and evolution of language, societal norms, and technology [23]. On the one hand, these social structures are established and re-produced through the way in which they are referenced in the everyday activity of the community-of-practice [38] in which they are rooted; on the other hand, they provide affordances for the constitutive practices, as they render certain ways of expressing oneself and certain ways of acting meaningful and understandable to the others. Wittgenstein uses chess pieces as an example. The meaning of the king or the queen in a chess game depends on the way we play chess [39]. Though the rules may be explicated, the meaning of rules and, especially, what it means to follow them, is again rooted in a practice.

Schatzki further explores the concept of social practices and defines them as a 'nexus of doings and sayings' [28] linked by (a) common goals, (b) explicit rules and instructions, and (c) a shared understanding of how different means are employed for a common end [28]. Similarly common practices render tools, props, and environments that support the practices, meaningful.

Using the foregoing definition, DB's stand-up meetings may be described as a practice: (a) The purpose of the stand-up meeting is to coordinate the joint work, and to share knowledge about current tasks to this end; (b) an explicit rule is that the first slides to be discussed address issues from the continuous integration (CI), the production site, and urgent bugs, and thereafter, each team member shares what is deemed necessary; (c) the team members

share an understanding, for example, of what is important, and why or when in a meeting to raise what kinds of concerns. As one of the developers states, *"We have never done it where you have to say something. You say something if you have something to say. People are capable of figuring that out themselves."* Some of the explicit rules – such as discussing CI, live site issues, and urgent bugs first – are enforced by the software robot that collects data from various systems to create the first slides for the meeting.

Knorr Cetina [5] argues that Schatzki's framework needs to be further developed for it to explain the dynamics of the design practices she observed when researching the development of a particle accelerator: The goal of such 'objectual practices' is not fully stated when the project starts, but unfolds throughout the design and construction processes. With the common object of design, the design and development practices must also evolve and unfold. We suggest that software development needs to be understood as such a set of design and development practices, which need to evolve along with the software being developed. How does this unfolding and evolution of the shared practices occur?

To describe processes such evolution and unfolding of practices, sociology developed the concepts of articulation work [33] and meta-work [10]. Based on empirical work in a hospital, Strauss [33] uses the term 'articulation work' to describe how different actors at a hospital coordinate a tight mesh of heterogeneous activities, and the standardization of cooperative procedures they built on. Gerson [10] distinguishes between situated articulation, that is, the coordination of tasks as part of their implementation, and meta-work, which he uses to describe the planning and standardization of cooperative procedures.

In the analysis below, we show how DB maintained and evolved their common practices. We, especially show that the team developed a practice of changing practices a *practice of changing practices*.

4 RESEARCH APPROACH AND METHODS

Our research focuses on the evolution of the continuous software engineering practices at DB, and the rationale behind both the evolution and the current state of the development practices, from the members' points of view. Therefore, this study uses an ethnographic approach [30]. As the development team continuously deployed its software, participation in its development was not an option. The contact with DB started when the senior researchers met the CTO at a meet-up and a practitioner conference on configuration management. Two of the authors spent a substantial amount of time on the premises, in order to observe the everyday detail of the development practices. The rich description in section 2, the observation used as an illustration in section 3.2, and the description in the analysis, are parts of the detailed and rich descriptions necessary for the trustworthiness of ethnographic research. Below, we further detail the data collection, and the analysis of the field material, and we discuss the trustworthiness of the research.

4.1 Fieldwork and Data Collection

The empirical research that is the basis for this article took place over three months. During the initial interviews and observations, it became very evident that the way software is developed at DB is not static. Instead, part of its specificity and capacity to sustain distributed, high-quality software development, with high developer satisfaction, is due to the continuous care for, and adaptation of their practices. The CTO and the developers at DB emphasized the company's pragmatic culture, and the continuous 'grooming' of their practice. This prompted an interest in understanding how DB's very specific way of developing software is continuously evolved by the team and the CTO. The initial field work suggested that the retrospectives and the three weekly one-on-one meetings between developers and the CTO were at the core of this continuous situated improvement. The focus on improving the stand-up meeting emerged as criticism of the current state of the daily stand-up, and its adaptation was the subject of the first retrospective we observed. These two foci – evolving the development practices and stand-up meetings as an example – then helped to establish the scope for the later fieldwork. The theoretical underpinning (section 3.2) was selected after the study's focus was chosen, and, once selected, informed the scoping and the further fieldwork. Table 1 presents the context according to [21, 35].

Table 1: Characterization of the empirical context.

Attribute	Value
Year	2017
Empirical focus	Empirically based
Empirical background	Industry
Industry sector	Accounting
Subject of investigation	Situated software process improvement
Study results	Theoretical conceptualisation
Empirical research method	Ethnography, interaction analysis
Source of empirical evidence	Observ., interview
Location	Distributed
Team size	14 (12 developers, 1 PO, 1 CTO)

The field work was part of a M. Sc. thesis [20]. The researcher doing the fieldwork collaborated closely with the main author of [17]. The fieldwork took place in the Danish office. The field workers took part in the virtual meetings with all developer present, and interviewed developers situated in the two main sites. The developer initiating the change in the stand-up meeting format was working from Kiev, whereas the CTO and a second developer who was interviewed worked from Denmark. Table 2 presents a list of the empirical data collected.

The observations were partly full-day observations, partly they focused on stand-ups and retrospectives. During observation, detailed notes were taken, using a computer-based observation scheme that made it possible to quickly note who and what was being observed, where, when, how and why [14]. Pictures and screen dumps were taken, which illustrated the special context and interactions, and samples of relevant documents were collected. The meetings were recorded. We triangulated non-participatory observation with semi-structured and informal interviews. Semi-structured interviews of the CTO took place before the fieldwork started, after 1 month of field work and in the end. Interview guides were developed for the semi-structured interviews, prior to undertaking them. The first interview focused on getting an overview over the company, the development organisation, and the coordination and

communication between the different sites. The second interview focused on the distributed development and on the tooling. The 3rd interview focused on the history of the continuous development practices and was designed to fill the gaps in the understanding of the continuous development process. Informal interviews were used to develop an understanding of the context of the observations: After the observation of the adaptation of stand-up meeting format, the developer initiating the discussion at the retrospective and the CTO were interviewed regarding their perspective adaptation of the stand-up meetings. An interview with a developer, who was part of the retrospectives but did not initiate the change, developed into a more broad discussion of the evolution of the development at DB and his prior experiences with a more rigor implementation of SCRUM.

Table 2: Fieldwork

Method	Date(s)	Documentation
Open non-particip. observation	30th Jan 2017 15th Feb 2017 1st Mar 2017 3rd Mar 2017 22nd Mar 2017 23rd Mar 2017 28th Mar 2017 31st Mar 2017 4th Apr 2017	Systematic field notes
Semi-struct. interview: CTO	16th Nov 2016 1st Mar 2017 21st Apr 2017	Transcript
Non-particip. observation of retrospective	3rd Mar 2017 31st Mar 2017	Field notes Transcript
Unstruct. interview: developer, formalized methods	22nd Mar 2017	Transcript
Unstruct. interview: CTO, improvement of stand-up	23rd Mar 2017	Transcript
Unstruct. interview: developer, improvement of stand-up	31st Mar 2017	Transcript
Discussion of prel. results: CTO	21st Apr 2017	Transcript

4.2 Data Analysis

The reflection connected with the beginning of the analysis influenced the subsequent field work. For example, we planned an interview with one of the developers participating in the first retrospective, which in turn informed an interview with the CTO. Owing to these characteristics of qualitative research Robson [25] also refers to qualitative research designs as 'flexible research designs'.

As the first step of our analysis, the observation schemata were developed into field notes that presented the observations in a textual form that was supplemented by personal reflections, emerging questions, and plans for future actions. Audio recordings of recorded meetings, semi-structured and informal interviews were transcribed. The interview transcripts and the field notes were coded in order to identify themes that categorized the content of the material, parallel to, or shortly after its collection. In the second round of analysis, a set of codes was developed, mirroring the study's evolving focus, the improvement of software development practices, and the focus on stand-up meetings. These codes were further influenced by the theoretical underpinning chosen at that time, and their purpose was to help to understand the rationales of the practices and their evolution. The first set of thematic codes comprised ten themes: *Experimentation as a Driver for Progress*, *Continuous Reflection*, *Practicing what makes Sense*, *Trust Drives the Capabilities to be playful and creative*, *Working Distributed Co-located*, *Using One-on-Ones as a Proactive Tool to Make Developers more Pragmatic*, *Showing Interest and Supporting rather than Controlling and Regulating*, *Decision Making is a Shared Responsibility*, *Rejecting Formalized Methods and Methodologies*,and *Fighting Status Quo*.

These themes were used in the discussion of preliminary results with the CTO. In this discussion, e.g. the importance of the one-on-one meetings between CTO and the developers was put into perspective and finally collapsed with *Continuous Reflection* into *Facilitating for Reflection*. Likewise *Working Distributed Co-located* was dropped: It was a condition to make a continuous evolution of distributed software practices work, but did not contribute to how the evolution came about. The result of the thematic analysis is presented in section 5.3

Section 5.2 uses an interaction analysis approach [15] to relevant parts of two retrospectives. In this approach a very detailed transcription of a conversation or discussion is analysed turn by turn to understand not only what the participants talk about but also how they through their interaction e.g. come to decisions, and what other resources they refer to when doing so.

For the article the first author revisited the analysis and checked and refined the transcription of the retrospective. For the timeline in figure 1 an additional interview was performed.

4.3 Trustworthiness

Threats to the validity of qualitative research are mainly addressed through the research design. To ensure the trustworthiness of our research [25], we triangulated our observations with interviews and document analysis; the fieldwork took place over an extended period, which allowed us to check both intermediary findings and the final results with the members of the team we observed; the collaboration among the field workers, and the discussions in the research team acted as debriefing. The use of theory further supported the evolving analysis and field work. Finally, detailed descriptions are provided, allowing the reader to criticize the analysis. Below, these measures are further detailed.

4.3.1 Triangulation. Data triangulation. Observations were triangulated with both semi-structured and informal interviews. Where suitable, photos and screen dumps were taken, and documents were collected.

Researcher triangulation. As described above, two fieldworkers closely collaborated. Interviews in several cases also included other members of the research team. The two main field workers shared their analyses and – when using the same material – reviewed each other's open coding. To increase the trustworthiness of the analysis for this article, the first author re-analyzed the field material. Deviations were discussed and addressed.

4.3.2 Member Checking. The flexible research approach allowed for adaptation of the fieldwork, which was also used to check our evolving understanding with the developers and DB's CTO. In particular, the interviews listed under 3) and 7) in Table 2 were explicitly used to (also) check the evolving understanding of the rationale behind the continuous improvement, and the change in the way stand-up meetings are held.

4.3.3 Debriefing. The research team regularly met and discussed the evolving analysis of the field material. This debriefing was used to check for possibly-evolving researcher bias due to becoming too familiar with DB's way of developing software.

4.3.4 Audit Trail and Detailed Descriptions. The fieldwork was meticulously documented; where possible interviews were recorded and transcribed. The detailed descriptions that underpin the findings in the next section further assist the reader to follow the line of argument in the field material, and criticize the findings and insights.

4.4 Limitations

As our research focuses on a specific constellation of practices in a specific company, the concrete practices cannot be abstracted and transferred to a new context with a similar result. However, relating the observations to social theory helps to understand the maintenance, adaptation, and evolution of software development practices in teams. In our discussion, we relate the analysis to the theoretical understanding developed in section 3.2 , which allows the identification of aspects of the observations that are candidates for transferability.

5 ANALYSIS: EVOLVING THE STAND-UP MEETING

Our analysis has three parts. First, we provide an overview of the activities and meetings that were involved in the evolution of the stand-up meetings. Second, we present a detailed interaction analysis of the discussion during the retrospectives, which shows how the team renegotiated its practices; here, we show how the various elements of practices presented in section 3.2 come into play. Third, we present a thematic analysis based on the interviews preceding and following the evolution of the stand-up meeting, concerning what enables a team to take care of the continuous evolution of their practices.

5.1 How the new Stand-up Meeting Developed

The change in the stand-up format was prompted by one of the developers ('initiating developer', below shortened to I-Dev) because he was dissatisfied with the knowledge-sharing in the stand-up meetings. He found he lacked information, and people contacted him to ask for information about his tasks. After first trying to change the situation by providing more information about his own work during the stand-up meetings, he took up the matter in a 'one-on-one' with the CTO. The CTO holds a one-on-one with each developer every three weeks. *"This [the one-on-ones] is more a place where [...] I listen to frustrations and maybe try to guide them."* This led to the decision to raise the question in the next retrospective.

At DB, retrospectives are conducted every fourth week. All developers, the CTO, and the product owner participate. The retrospective during which the stand-up meetings were discussed was moderated by one of the developers. A retrospective is conducted according to four agenda points: 1) results of the last retrospective; 2) sharing good and bad things, and 3) generating ideas about what to do about them; and 4) agreeing on what to change. The online Slack meeting is supported by a shared Google spreadsheet that the participants edit collaboratively during the discussion. For example, the collection of 'good and bad things' is conducted as an individual task,and then the points are elaborated on by the authors. Here, the I-Dev raised his concerns about the stand-ups. When agenda point 2 was concluded, the team members were asked which point they would like to discuss further. Here, both, the I-Dev and another team member again raised the stand-up meeting question.

At the end of the foregoing discussion, the team decided to try out a format for the stand-up meetings that emphasized knowledge-sharing among sub-teams: one member of each sub-team reported what he/she deemed relevant during the entire team's daily stand-up. Slides that share information about individual team members' tasks were still welcome. The solution was tested during the subsequent weeks. The stand-up meetings that we observed following the retrospective discussed above became longer: They now lasted an average of 14 minutes, instead of the previous average of 9 minutes.

5.2 Negotiating the Change of the Stand-up Meeting Practices

As discussed above, the retrospectives take place online via Slack, and are supported by a Google spreadsheet. The meeting is moderated by a developer (RM-Dev). Here, I-Dev raised the question of the stand-ups during the second step, 'Sharing good and bad things'. After presenting his positive points, he raised the question and pointed out how the lack of knowledge-sharing shows in everyday work, and proposed returning to an earlier good practice. By 'teams', he refers to the sub-team division.

I-Dev *Our main stand-up is getting shorter and shorter. We are not sharing a lot from different teams. I am not sure if it is a problem for everyone but at least for me. Yes, and guys needed to ask me in person what we are doing in mobile and I did the same, asking the web site guys, what is the progress on their side and so on. So, I would like to... whether we can do one stand-up like we did before. And that's it.*

The sharing continues. In the wrap-up following this initial part of the retrospective, both I-Dev and another developer start the discussion by taking up the question again. This launches the whole team's joint root-cause analysis, solution-finding, and reflection, which also explains the sometimes very tentative language. The first two developers who react agree that the knowledge-sharing in the sub-teams may be responsible for the lack of knowledge sharing in the common stand-up meetings, and discuss possible solutions.

I-Dev *Maybe could we could talk about stand-up? Me and Dev1 we have the feeling that we can improve something there? So guys, it would be nice to hear from you: Do you feel some problems with stand-ups as well as me and [Dev1]. Are they too short, too long, too separated, it's ok, leave it alone? And so on.*

Dev2 *I think that we can do it a bit better, absolutely. And the way we could do it is, take more mobile into account, present it at the common stand-up, and then, if we have something special, we should be ready to maybe have a small meeting after. Because I think sometimes we will go into details that [are] more related for the team who is doing it rather than all people. And I guess, we have exactly the same case for payments and onboarding or other projects. [...] So, the stand-up of cause should be for all people, but I still think we need somehow have something for the specific team.*

Dev3 *Thanks as for me, I think that we need to have like maybe discard our mobile stand-ups and maybe some other stand-ups [...], how it works now, and to move like mobile status, maybe some other statuses, like most important stuff in slides. And if we need something discuss, only if we need, we can agree in main stand-ups that we can make a call between the guys [...] who need to discuss this concrete thing. [...] Or maybe just to discuss it, like, personally in calls. I think it would be better.*

The CTO proposes considering whether it is the unclear responsibility for sharing the relevant information from the sub-teams, rather than that the sub-team's additional meetings that is responsible for the lack of knowledge-sharing. He proposes a different way of addressing the problem, a change of rules, which is then considered by the developers.

CTO *Do you guys think that maybe the reason it is not being shared now... right now ... is because, when you have a team of four people everybody [...] might sometimes think that the others will probably do that? It is just a suggestion and we do not have to do it, but I was thinking that maybe each team could have like a person who is the stand-up person who is like responsible for adding one slide once in a while. That's an idea.*

Dev1 *Yeah, we and we can change, so one week it is this person the next week it is another person. This person will know that he is responsible for this week and he will add the new slide. I think it will work.*

The moderator starts to discuss the guidelines of what should be shared with the whole team.

RM-Dev *I think the reason why we are not sharing that at the general stand-up is that we do not know if internal team stuff would be interesting for the whole team, so...*

Dev1 *Maybe we can share just not some very specific stuff, internal, but something that: well, 'We are working on this feature.', 'We released this feature.' I think it can be interesting for a lot of people [...].*

The CTO then offers to implement the change in the template for the stand-up meeting slide deck. Dev1's remark about the evaluation of the change indicates that this implementation is done as an experiment.

CTO *You know, there is the stand-up slides template. I could be responsible for – you know – the teams we have at the moment, I'll just add a slide for each team and then you can fill it out. So, it is there and it's ready to fill out.*

Dev1 *Yeah that would be nice, I think. So, we can assess this idea and... maybe implement it.*

RM-Dev *We can have a slide in the stand-up template for each team. Just to have a status for that team.*

Dev1 *Yeah.*

Dev3, who has been quiet throughout this exchange, agrees, but raises a more general concern: Does the new format for the stand-up meeting actually support the purpose of helping the team to develop and maintain a coherent understanding of the product? His concern is taken up in the following discussion, which leads to a further elaboration of what to use the stand-up for, and what to coordinate within the sub-teams. The matter is finally settled, with the CTO stating that coordination and sharing within the sub-teams is up to the sub-teams.

Dev3 *I think it is okay to add more but it is more like to mention people, to mention teams that they have to... just give a short status on what we are doing. And I think it is really important if we are going later to have to work together to have an idea about the product. In general, I think we also miss it, we miss to know what is going on.*

Dev4 *So maybe we are changing the stand-up from being what people did to what is happening on the individual project. So, we do not have personal slides, we only have project-slides. And then we put all the other miscellaneous stuff at the end. Or something like that. [...] I think, that it is a good idea in the sense, I mean, that when I am working with [Dev5] and [RM-Dev] on payments, we don't all three of us do add slides what we did on payments, that's redundant because we coordinated already between us. So, we as a team just need to tell everyone else what we did. So yes, [...] that makes sense: project-slides rather than people-slides.*

Dev1 *...or maybe teams?*

Dev3 *We should remember [...] that stand-ups have several... We should think about what is the purpose of stand-up. What we are talking about right now – and in fact also what I am really missing – is about [...] sharing knowledge about what we are doing. At the same time, I think, we [...] should also remember that stand-up is also for coordinating the day, as I see it.*

Dev4 *But I think we would do that in the individual teams on an ad hoc basis, right? I mean, RM-Dev and I wouldn't coordinate in the daily stand-up.*

Dev3 *Exactly, [...] that is my point, and that is the reason. For me it is difficult to see how we can avoid a [...] morning meeting in the team.*

Dev4 *But I don't think it necessarily is a meeting, I mean. We just talk on Slack.*

Dev1 *Yeah, we cannot make a call, we can just... So, if some task some stuff needs to be coordinated, we just talk on Slack. Just chatting or calling if we want if we need to coordinate.*

CTO *So...*

Dev1 *Sorry! Maybe I work on some stuff, and they work on different stuff, and they are not connected with each other, and we do not need to coordinate, and we just want to work on it, maybe we do not share and coordinate it.*

CTO *I think [...] as a basis [...], each team has, you know, individual needs and, I think, if someone on the team really feels that stand-up is really necessary, it's important that the team listens to that. But I agree completely that each team can [...] decide how to coordinate on the team what feels comfortable to them. It also depends on how many you are on the team. If you are two people, maybe, you do not need to scheduled stand-up but on mobile team where you are four, it might be more necessary. So... But you can decide on the teams, I guess.*

Dev4 *And it makes total sense, and that is also how we did it on Website team, where we decided to have stand-up twice a week. So, I think it is a good point, yeah.*

Towards the end of the meeting, the team summarizes the discussions and decisions.

CTO *[...] And the next one is also mine: So, it's what we talked about that each team, you know, should have somebody responsible for filling out that one slide and then you can decide who is doing it this week, as you like on the team. But I think, [...] things run a bit smoother if everybody knows who is responsible for things instead of it is just the team's responsible.*
Dev4 *But shouldn't it be the team's responsibility to figure out how they want to manage their slide?*
CTO *Hm... You can call it a suggestion. [Laughter]*
RM-Dev *Ok. What else do we have?*

In the end, the moderator sums up the action points:

RM-Dev *For the second one for each team to have slides in stand-up. [CTO], will you do it?*
CTO *Yes!*
RM-Dev *Yes. [Break] Awesome.*

The new stand-up format was then implemented. We observed an increase on time the stand-up took. Also, the content changed.

During the next retrospective, it became evident that everybody understood the implementation of the new stand-up format as an experiment. It was evaluated at the beginning of the retrospective.

RM-Dev *So, in general, how do you guys feel about those team slides?*
Dev5 *I think it is okay.*
I-Dev *Yeah, I also think it was okay.*
Dev4 *I liked it.*
Dev3 *Yeah, it was good.*
Dev2 *I think it is okay as long as you are not always supposed to fill something out.*
Dev4 *And you are still allowed to do your own slide if you want to.*

With this evaluation the change in the stand-up meeting was made permanent.

A few weeks later, the team decided to change their stand-up meetings more drastically: They decided to have daily stand-ups in the evolving sub-teams, and meet only once a week for a stand-up of the whole team.

5.3 What Enables a Team to Evolve their Practices

As described in the methods section, we triangulated our observations with interviews, both to better understand the members points of view, and to gain additional insight into how DB developed their very specific approach to developing software. This section presents a thematic analysis of observations and interviews. The focus is still on the stand-up meeting as an example; additionally, we use examples of the evolution of other practices to show that the evolution of the stand-up meeting is not an exception.

One of the most prevalent themes in connection with DB is its strong *pragmatic stance*, an emphasis on reducing waste. This is supported by *experimentation* with changes to development processes, such as the stand-up meetings that are the focus of this article, the tools, the development environment, and the implementation technology; the experimentation and the evolution of practices based on it are supported by *trust* and a sense of *responsibility*. This also requires the developers' ongoing *reflection* on, and *awareness* of their own practices.

In the analysis, a layer of supportive individual factors became evident, underpinning the concepts presented below: communication skills, courage, and a very reflective leadership style. As we did not focus on this more psychological level, and do not have enough such material, we decided to leave this to future research.

5.3.1 A Pragmatic Stance. At DB, both the developers and the CTO support a thoroughly pragmatic stance, an emphasis on reducing waste. The development processes, rules, and ceremonies are cut down to what is deemed really necessary. As one developer puts it, *"I'm actually a certified Scrum Master and I have worked as scrum master on several occasions. [...] There have been places where they introduced Scrum, and when they introduced it, they would often like to implement it as it is prescribed. [...] It's a very forced way to work, where to some extent you feel embarrassed. You just sit there and build these walls of limits, and do role play – come on, we are adults [...]; we know how to communicate."* (Translation by the authors).

The CTO confirms. *"I have been to lectures with some incredibly intelligent people who told me: 'Well, you have to do as it is in the book, otherwise you will not be able to say that you're doing Scrum' or 'You cannot say you're doing Kanban because the book says this and that' and 'I have little regard for people who bend the rules'. And I totally oppose that. I am 100% pragmatic. We just do what works and makes sense, so that is what I would call what we do."* (Translation by the authors.)

The development of the stand-up meeting slides populated by the software robot is an example for a simplification as just described. The robot reduced the effort and frustration of assigning crucial bugs and live site errors individually. Another example is the communication and collaboration infrastructure. The CTO stated, *"I have been to places where many people still have a whiteboard with paper notes on it. Then there is a webcam, and then you have some screens, [...] they then try to see what's happening on the whiteboard through that webcam [...] it was completely ridiculous. It does not work in any way. [...] There is no reason to over-complicate things. [...] That's why slides are so cool - it's the perfect online version of paper and pen. You can do whatever suits you."*

The ultimate criterion for introducing new tools, methods, or new ways of collaborating is whether something helps the team to develop better software in a better way. This is where experimentation comes in: To establish whether something is better, it has to be tried out for some time.

5.3.2 Experimentation. The change in the stand-up meeting is our prime example for experimentation. The old format had already been developed through a series of experiments, where for several weeks the whole team tried out different formats. Ultimately, the team voted on different formats and elements and decided on the format used until the retrospective analyzed above.

The in-depth analysis of the stand-up meeting discussion during the retrospective reveals further aspects. The results of the discussion of the stand-up first prompt an experiment, which is then evaluated in the subsequent retrospective. The interaction analysis above shows that the developers are used to running experiments: The decision in the first retrospective was clearly understood as an experimental change in the stand-up format, and nobody was

surprised to be asked to evaluate it in the next retrospective. Only then did the new format become permanent.

Stand-up meeting formats were not the only aspect of the development method with which DB has experimented. Other examples that we learned about through the interviews were the change from a mandatory peer review before deploying code, to a voluntary one, changes in who is responsible for live site errors, and integrating testing and quality assurance into the developers' duties. As the CTO explained in an interview, *"It is normal that we evaluate what we are doing all the time, so in general [...] experimenting, instead of just saying no because it sounds like a bad idea. Often, we will just try things, and then we will see if it is actually a bad idea."*

Both the CTO and the developers emphasize that the experiments are based on the developers' situated assessment of where are the problems, of what works and what does not. In return, this requires the development team to assume responsibility for their own development processes.

5.3.3 Trust and Responsibility. The initiative to change the stand-up meeting format came from one of the developers, who recognized that information apparently relevant to his colleagues was not being shared in a relevant forum. In the retrospective, the developers conscientiously discussed whether they shared the perception of the problem, possible solutions, the implication of the proposed change, and how this would impact the knowledge-sharing mechanisms that they had developed in the various sub-teams.

This is not a unique example of developers taking responsibility for both their own work and for their team's development process. The CTO reported that after the mandatory code review was replaced by the rule that developers decide whether to ask a colleague for a code review, efficiency increased and production site errors dropped. In the foregoing discussion of the change to the stand-up meeting, the CTO was one of the last persons to contribute to and support the solution that emerged from this discussion. In the follow-up interview, it became clear that he still had some concerns regarding the lack of sharing in the stand-up meetings, but consciously held back, to leave the responsibility with the team.

Having a team take responsibility requires trust on the part of management. This was confirmed during the interviews with the CTO. Much of the reported experimentation with changes in the development process confirm that the CTO 'walks his talk' and does leave decisions regarding both changes in the development process and regarding implementation technology with the team.

5.3.4 Developer Reflection and Awareness. At DB, both trust and responsibility, and experimentation, depend on the CTO and the developers' awareness of what is going on and where problems might develop, and on continuous reflection on how to address the problems identified. When the developer became aware of the lack of knowledge-sharing in the stand-up meetings, he first took up the matter in a one-on-one meeting with the CTO. The one-on-one meetings play an important role, regarding awareness of, and reflection on common practices. In an interview, the CTO described how, during the one-on-ones, he supports the developers in their reflection on the problems they encounter, to analyze them, and to identify possible ways to address them.

The retrospectives serve a similar purpose to the one-on-one meetings, on a team level: The retrospective analyzed above is an example of a group's reflection on their practice of stand-ups, the purpose of the stand-ups, and the explicit rules of the stand-ups. This resulted in a change that was again discussed in the subsequent retrospective. At DB, the retrospective is promoted as the venue in which problems are taken up, analyzed, and where, if necessary, experiments and changes are decided on.

In particular, the continuous experimentation with their processes requires a team to be aware of both current practices and what is subject to experiment right now. In a later interview, the CTO told of a hiccup, where an experiment was not decided on in this collective forum: Part of the team was unaware of the experiment, and felt frustrated because they were left out of collective decisions.

6 DISCUSSION

The previous section presented a qualitative analysis of our empirical case. This section presents the insights we gained from comparing the analysis to the related work: We first relate our findings to the social theory introduced above. We then discuss team responsibility and self-organization as core reflective practices. In the last subsection, we discuss what other companies could learn from DB.

6.1 Establishing Improvement as a Practice

Our analysis shows that the observed adaptations of DB's development practices are not part of an unwitting deterioration of a specific method, but a conscious adaptation of a practice. In this case, the stand-up meeting needed to change, to address the implications of the introduction of sub-teams. This must be regarded as a fine-tuning of the supportive structures of a high-performing team. In this subsection, we will recur to the theoretical underpinning developed in section 3.2 to deepen our understanding of the observations.

The theoretical underpinning developed in section 3.2 allows us to analyse the observations in a way that relates purpose, rules and shared understandings underpinning the observed actions. We used the field material already in section 2 to provide an example of how the stand-up meeting may be described as a practice consistent with the social-theory underpinning. Below we further elaborate the use of Schatzki's concept of social practices. Schatzki's concepts can also be used to describe the retrospectives as *practices of changing practices* or *meta-practices*.

The interaction analysis in section 5.2 shows that the theoretical elements of a practice may be found in the way the developers discussed the stand-up meeting format. To recap practices are defined as 'nexus of doings and sayings' linked by (a) common goals, (b) explicit rules and instructions, and (c) a shared understanding of how different means are employed for a common end [28]. In the retrospective, two developers shared the concerning observation that the amount of knowledge-sharing during the stand-up meetings had decreased. They stated a problem with the 'doings and sayings' that constituted a stand-up meeting. (a) The team referred to the purpose of the stand-up meeting – to share knowledge and coordinate the day's work – in their discussion. Instead of abandoning the stand-up meeting, the team engaged in the discussion of what could be the cause. The diagnosis was that a change in team

structure led to that the shared understanding (c) of what to communicate did not fit the new situation. The team decided to change the rules (b): Each sub-team should share what they deem relevant for the whole group, though sharing of relevant issues by individual developers was still possible. To implement that, the tooling that supported the stand-up meetings was adapted. The team shared the understanding that the change in the stand-up meeting to be an experiment, and would be assessed in the following retrospective.

The retrospective itself may be described as a social practice along the definition above as well: (a) Everybody is aware of the purpose of the retrospectives that also becomes visible in the agenda points: their purpose is to reflect on the current development process, discuss how to address issues raised, and agree on what to change. Further to evaluate previous changes that were implemented as experiments. (b) There is a set of explicit rules that guides the retrospective: The retrospective has a specific structure and process. It takes place every fourth week at a specific time. All developers who are not on vacation, and the CTO are present. One of the developers leads the team through the process. All participants were familiar with the structure of the meeting, which is supported by a shared Google spreadsheet. (c) There is a shared, implicit understanding of what points should be raised, during which part of the retrospective, and how. Also, normally the discussion would result in some kind of decision. In the case of the specific retrospective analyzed above, the decision was to experiment with a different format for the stand-up meetings. To this end, necessary preparatory tasks – here, the development of the template for the stand-up meeting slide deck – were distributed. Also, everybody understood the decision as an experiment to be evaluated at the next retrospective.

Further, our analysis shows that the retrospectives in our case were supported by other practices, in order to achieve their purpose. Along with the retrospective, one-on-one meetings with the CTO and experiments are practices that DB developed to assess, reflect on, and change their development practices. The purpose of this set of practices may be understood as the implementation of the team's meta-work [10]. In other words, improving and changing practices have themselves become established practices. In the triangulating interviews, it became clear that these *meta-practices* were carefully established and promoted by management.

The theoretical concepts introduced in section 3.2 help to distinguish between a deteriorating, uncoordinated development and a conscious situated improvement of the shared practices: if a specific practice is the result of its consideration and adjustment in the context of a set of meta-practices, it is a conscious situated improvement.

Thus, our analysis shows that the concept of social practices that we developed in section 3.2 may be used to analyse both software development practices and meta practices as they are established in-situ by a software team. It also shows that the concept may be used to describe the meta-practices established by the team. And, finally, the elements of the definition may be found in the analysis of the team's discussion of their own practices in their retrospective.

One could, as part of future work, further explore how the elements that, according to Schatzky, define a practice may be translated into a set of questions that guide the situated analysis and improve development practices: What are the purposes and goals of a practice? How do the tacit understandings and explicit rules contribute to these purposes? How are these practices supported by tools? Regarding the change, the questions might be: What is the reason existing practices do not work satisfactorily? Which of the elements do we need to adapt?

6.2 Reflection, Experimentation and Agile Improvement

Lous et al. [16] argue that 'Inspect and Adapt' is one of the core ingredients that allows the case company to implement radical agile and continuous development practices. In this article's in-depth analysis, we show how this is implemented as an established set of meta-practices. The one-on-one meetings with the CTO give the team members a space to reflect on the current practices. At the retrospectives the team members can raise issues for the team to discuss, and the experimentation allows the team to consciously evolve the way the team collaborates and coordinates. Practices that are determined to have become inadequate are not just abandoned, but discussed by the team. In the example above, the purpose of stand-up meetings and the possibly-changing circumstances are brought forward. New rules and guidelines are decided together, and then implemented by the whole team. Formulating such a change as an experiment allows the team to postpone possibly controversial decisions until after they have experienced the effects of a proposed change.

It is very evident in the field material that the developers are the ones who deliberate changes in rules, and tools that support collaboration and coordination in development. The case we chose for this article may not strike one as a huge change. However, it is an example of a change that we observed during the period allocated to the fieldwork. As highlighted in the section 5.3, we have learned about a number of other cases where individual practices were changed by the team.

Our findings resonate with the existing literature on retrospective. The discussion of the stand-up meeting may also be described according to the categorizations developed in [1]: Initially raising the matter may be described as 'Reporting and Responding', the root-cause analysis and development of changes as 'Relating and Reasoning', and the decision about the action plan may be seen as 'Reconstructing'.

The analysis of the revision of the stand-up meeting provides an example of how agile development consistent with the last of the 12 principles underpinning the Agile Manifesto (*'At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.'*) may take place. Our analysis demonstrates how situated software process improvement may take place bottom-up, as part of normal software development practices. This differs from a top–down implementation of software process improvement, which makes use of the retrospective as an instrument of researchers and management [27]. Nonetheless, our research makes it evident that DB's development is not a deterioration, but a consciously and carefully evolved practice. Studies such as the one presented here complement survey-oriented studies, e.g. [36]. They allow us to explore the mechanisms behind the development of the hybrid methods detailed in the article, and thus

allow us to understand whether software development practices developed based on deterioration or careful evolution.

6.3 What can other Companies Learn from DB

DB presents a very specific instance of agile development practices that have been adapted by a specific team to fit the circumstances at hand: software provided as a service by a small and distributed team with strong technical leadership, a CTO who protects the team's mandate to determine how they decide to work. Neither their development practices nor their tooling may be copied as is by another development team: they would not be able to transfer DB's success to a different context.

The insights provided by this study that may be transferred to other contexts are related to the theoretical underpinnings: The success of the application of a method is not based on developers following the rules that are part of a method, but on the methods being adapted into the practices of the development team. That means the team relates their purpose – that is, the problem the method is designed to solve – to their own needs, and adapts the implicit understandings and the explicit guidelines and rules to fit their specific circumstances. If the team, the needs, or the circumstances changes, this adaptation must evolve. To this end the often neglected meta-practices, the practices of changing practices are crucial. The analysis provides an example how such a set of related meta-practices can look like. In the article 'What does it mean to use a method?' [6] the author suggest understanding methods as practice patterns, and formulating methods to explicate their purpose, the explicit rules, and the normally not-explicated, taken-for-granted understandings. This scheme could be one way to inform the meta-practices; that way software teams would be supported in their application and tailoring of the methods.

Following the rules of a formalized method may be justified, for example, when a new team is established, or when development needs to be coordinated across a large organization. When a new team is established, an 'off the shelf' method may provide a framework for the most necessary practices, which may then be adapted to the situation at hand. With respect to large and distributed development organizations, the Rönkkö et al. [26] showed that in an international company a company-wide process model was central to facilitating cross-site collaboration and coordination. However, even there, the project members used their common sense to adapt that model to the situation at hand.

DB is still a small company, though the number of developers and tasks allows it to split into sub-teams. It is unclear how the observed meta-practices scale. In larger development projects, different teams may be able to evolve their local practices as they see fit. The evolution practices for coordinate process evolution across local teams will be subject to future research.

7 CONCLUSION

We started by asking how the evolution of software development practices can be investigated, and what distinguishes a software process that has degenerated from one that has evolved as the result of conscious deliberation. To address the questions, we studied how a team continuously developed their software development practices in a coordinated manner.

Regarding the first questions, the theoretical underpinning we developed in section 3.2 proofed useful to identify and describe both the practices that were evolved and the set of related meta-practices the team had established to this end. By combining one-on-one meetings between the CTO and the developers to foster reflection, retrospectives to address concerns about current development, and experimentation as a way to explore new ways of working, the team maintained and evolved a set of common practices meet to their evolving needs. We analyzed an example of such a controlled change – the evolution of the stand-up meeting format – so it could continue to fulfill its purpose, even when the team started to split into sub-teams.

The observation of such meta-practices provides the answer to the second question: if a team implements meta-practices like the ones described above, their software development practices are result of conscious deliberation rather than of degeneration.

In the discussion we argued that social practice theory may help to explain the observed retrospective and experimentation as practices to change practices, and how the practices that formalize the meta-work that this team developed support each other. This conclusion provides a building block for a practice theory for software engineering.

We suggest that companies consider implementing software engineering methods as a creative process and establish practices of changing practices or meta-practices as part of their everyday development.

We indicated several lines of future research: Can we apply practice theory to support teams' evaluation of their practices in their retrospectives? Is it possible to scale such bottom-up improvement through meta-practices to large teams, and, if so, how? Such research would also improve the understanding of how various practices interact with each other and with their organizational contexts when implemented by different organizations.

ACKNOWLEDGMENTS

Thanks to the DB team for their openness and welcoming attitude.

REFERENCES

[1] Yanti Andriyani, Rashina Hoda, and Robert Amor. 2017. Understanding knowledge management in agile software development practice. In *International Conference on Knowledge Science, Engineering and Management*. Springer, 195–207.
[2] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. [n.d.]. Manifesto for agile software development. https://agilemanifesto.org/
[3] Elizabeth Bjarnason and Björn Regnell. 2012. Evidence-based timelines for agile project Retrospectives–A method proposal. In *International Conference on Agile Software Development*. Springer, 177–184.
[4] Graham Button and Wes Sharrock. 1994. Occasioned practices in the work of software engineers. In *Requirements engineering*. Academic Press Professional, Inc., 217–240.
[5] Karin Knorr Cetina. 2005. Objectual practice. In *The practice turn in contemporary theory*. Routledge.
[6] Yvonne Dittrich. 2016. What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology* 70 (2016), 220–231.
[7] Brian Fitzgerald. 1996. Formalized systems development methodologies: a critical perspective. *Information Systems Journal* 6, 1 (1996), 3–23.
[8] Brian Fitzgerald, Nancy L Russo, and Erik Stolterman. 2002. *Information systems development: Methods in action*. McGraw-Hill Education.
[9] Christiane Floyd. 1992. Software development as reality construction. In *Software development and reality construction*. Springer, 86–100.
[10] Elihu M. Gerson. 2008. *Reach, Bracket, and the Limits of Rationalized Coordination: Some Challenges for CSCW*. Springer London, London, 193–220. https://doi.org/

10.1007/978-1-84628-901-9_8
[11] Rosalba Giuffrida and Yvonne Dittrich. 2014. How social software supports cooperative practices in a globally distributed software project. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 24–31.
[12] Rosalba Giuffrida and Yvonne Dittrich. 2015. A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams. *Information and Software Technology* 63 (2015), 11–30.
[13] James Herbsleb. 2016. Building a socio-technical theory of coordination: why and how (outstanding research award). In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2–10.
[14] Brian A Hoey. 2014. A simple introduction to the practice of ethnography and guide to ethnographic fieldnotes. *Marshall University Digital Scholar* 2014 (2014), 1–10.
[15] Brigitte Jordan and Austin Henderson. 1995. Interaction analysis: Foundations and practice. *The journal of the learning sciences* 4, 1 (1995), 39–103.
[16] Pernille Lous, Paolo Tell, Christian Bo Michelsen, Yvonne Dittrich, and Allan Ebdrup. 2018. From Scrum to Agile: a journey to tackle the challenges of distributed development in an Agile team. In *Proceedings of the 2018 International Conference on Software and System Process*. ACM, 11–20.
[17] Pernille Lous, Paolo Tell, Christian Bo Michelsen, Yvonne Dittrich, Marco Kuhrmann, and Allan Ebdrup. 2018. Virtual by design: how a work environment can support agile distributed software development. In *2018 IEEE/ACM 13th International Conference on Global Software Engineering (ICGSE)*. IEEE, 97–106.
[18] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville. 2007. 'Good' Organisational Reasons for 'Bad' Software Testing: An Ethnographic Study of Testing in a Small Software Company. In *29th International Conference on Software Engineering (ICSE'07)*. 602–611. https://doi.org/10.1109/ICSE.2007.1
[19] L. Mathiassen, A. Munk-Madsen, P. A. Nielsen, and J. Stage. 1996. *Method Engineering: Who's the Customer?* Springer US, Boston, MA, 232–245. https://doi.org/10.1007/978-0-387-35080-6_15
[20] Christian Bo Michelsen. 2017. *The Practice of ChangingPractices*. IT University of Copenhagen.
[21] M.B. Miles, A.M. Huberman, and J. Saldaña. 2013. Qualitative Data Analysis. SAGE Publications.
[22] Peter Naur. 1985. Programming as theory building. *Microprocessing and microprogramming* 15, 5 (1985), 253–261.
[23] Davide Nicolini. 2012. *Practice theory, work, and organization: An introduction*. OUP Oxford.
[24] Leon Osterweil. 2011. Software processes are software too. In *Engineering of Software*. Springer, 323–344.
[25] Colin Robson. 2011. *Real world research*. Vol. 3. Wiley Chichester.
[26] Kari Rönkkö, Yvonne Dittrich, and Dave Randall. 2005. When plans do not work out: How plans are used in software development projects. *Computer Supported Cooperative Work (CSCW)* 14, 5 (2005), 433–468.
[27] Outi Salo and Pekka Abrahamsson. 2007. An iterative improvement process for agile software development. *Software Process: Improvement and Practice* 12, 1 (2007), 81–100.
[28] Theodore R Schatzki and Theodore R Schatzki. 1996. *Social practices: A Wittgensteinian approach to human activity and the social*. Cambridge University Press.
[29] Kjeld Schmidt. 2014. The Concept of 'Practice': What's the Point?. In *COOP 2014 - Proceedings of the 11th International Conference on the Design of Cooperative Systems, 27-30 May 2014, Nice (France)*, Chiara Rossitto, Luigina Ciolfi, David Martin, and Bernard Conein (Eds.). Springer International Publishing, Cham, 427–444.
[30] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. 2016. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering* 42, 8 (2016), 786–804.
[31] Anders Sigfridsson, Gabriela Avram, Anne Sheehan, and Daniel K Sullivan. 2007. Sprint-driven development: working, learning and the process of enculturation in the PyPy community. In *IFIP International Conference on Open Source Systems*. Springer, 133–146.
[32] Margaret-Anne Storey, Neil A Ernst, Courtney Williams, and Eirini Kalliamvakou. 2020. The who, what, how of software engineering research: a socio-technical framework. *Empirical Software Engineering* (2020), 1–33.
[33] Anselm Strauss. 1985. Work and the Division of Labor. *The Sociological Quarterly* 26, 1 (1985), 1–19. https://doi.org/10.1111/j.1533-8525.1985.tb00212.x arXiv:https://doi.org/10.1111/j.1533-8525.1985.tb00212.x
[34] CMMI Product Team. 2006. CMMI for Development, version 1.2. (2006).
[35] Antônio R. D. R. Techio, Rafael Prikladnicki, and Sabrina Marczak. 2015. Reporting Empirical Evidence in Distributed Software Development: An Extended Taxonomy. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, Washington, DC, USA, 71–80.
[36] Paolo Tell, Jil Klünder, Steffen Küpper, David Raffo, Stephen G MacDonell, Jürgen Münch, Dietmar Pfahl, Oliver Linssen, and Marco Kuhrmann. 2019. What are hybrid development methods made of?: an evidence-based characterization. In *Proceedings of the International Conference on Software and System Processes*. IEEE Press, 105–114.
[37] Hataichanok Unphon and Yvonne Dittrich. 2010. Software architecture awareness in long-term software product evolution. *Journal of Systems and Software* 83, 11 (2010), 2211 – 2226. https://doi.org/10.1016/j.jss.2010.06.043 Interplay between Usability Evaluation and Software Development.
[38] Etienne Wenger. 1998. Communities of practice: Learning as a social system. *Systems thinker* 9, 5 (1998), 2–3.
[39] Ludwig Wittgenstein. 2009. *Philosophical investigations*. John Wiley & Sons.