Article

Interruptions in Agile Software Development Teams

Project Management Journal
Vol. 00(0) 1–13


Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/8756972821991365
journals.sagepub.com/home/pmx


Manuel Wiesche[1]

Abstract

Agile approaches help software development project teams to better meet user needs and ensure flexibility in uncertain environments. But using agile approaches invites changes to the project and increases interactions between team members, which both cause interruptions in the workplace. While interruptions can help in task completion and increase process flexibility, they can also hinder employee productivity. We conducted an exploratory study of four agile software development teams. Our analysis identified (1) programming-related work impediments, (2) interaction-related interruptions, and (3) interruptions imposed by the external environment, which were managed by improved information retrieval and reduced team dependencies.

Keywords

agile information systems development, agile software development, interruptions, IT project management, teams

Introduction

Working in uncertain environments fundamentally changes how we organize work (Rigby et al., 2016). Handling uncertainty requires continuous adaption and coping with change. Therefore, organizations need to balance flexibility and stability (Bazigos et al., 2015). In software development projects, agile approaches to project management are used to continuously recorrect by enforcing steady interaction with external and internal stakeholders (Rigby et al., 2016; Slaughter et al., 2006). Preplanned tasks are revised in an iterative manner and agile practices such as daily standups, planning sessions, and burndown charts foster continuous feedback and refinement. Informal knowledge exchange and problem solving are encouraged, and collaborative workplaces are created as colocated or digitally connected work environments (Dery et al., 2017; Lee & Xia, 2010; Maruping et al., 2009).

Thereby, agile software development projects increase collaboration with different stakeholders in software development project teams (Kudaravalli et al., 2017; Majchrzak et al., 2005; Pflügler et al., 2018). This collaboration has many benefits for the agile team, as it ensures that customer needs are met (Recker et al., 2017; Rigby et al., 2016; Vidgen & Wang, 2009), it fosters knowledge sharing (Ghobadi & Mathiassen, 2017; Kudaravalli et al., 2017), and it increases employee motivation (Tripp et al., 2016).

However, the openness toward change and the high degree of collaboration also cause interruptions for the software development team (Conboy, 2009; Drury et al., 2012; Fægri et al., 2010; Moe et al., 2010; Tanner & Mackinnon, 2015; Tregubov et al., 2017). We understand interruptions as events that impede or delay organizational members during work tasks (Jett & George, 2003). Examples include software developers interrupting their team members when asking for help or feedback, as well as impediments, when software developers wait for input or requirements are changed during the development process (Drury et al., 2012; Moe et al., 2012; Power & Conboy, 2015; Wiklund et al., 2013). The current body of knowledge suggests that recovering from interruptions is a central problem among software development teams (LaToza et al., 2006; Stjerne et al., 2019).

Agile practices are designed to cope with the uncertainty of high customer involvement, fast responses to change, and formalized elements to support collaboration and coordination (Conboy, 2009; Recker et al., 2017; Vidgen & Wang, 2009). However, the way in which agile approaches help software development teams in coping with interruptions that are caused by this mindset is unclear (Drury et al., 2012; Tregubov et al., 2017). More specifically, literature remains silent on how interruptions are handled in agile software development contexts. Therefore, the following research question guides our study:

[1]TU Dortmund University, Dortmund, Germany

Corresponding Author:
Manuel Wiesche, TU Dortmund University, Dortmund, Germany.
Email: manuel.wiesche@tu-dortmund.de

Which interruptions occur in agile software development teams and how do teams respond to these interruptions?

We conduct an exploratory case analysis of four agile software development teams and use grounded theory methodology analytic procedures to understand interruptions in our context (Wiesche et al., 2017). Our analysis identified (1) programming-related work impediments, such as changed requirements, errors, and developers waiting for information; (2) interaction-related interruptions, such as customer requests and formalized meetings; and (3) interruptions imposed by the external environment, such as technology-induced interruptions and interruptions caused by the work environment. Our analysis suggests that software development teams use practices embedded in the agile mindset to improve information retrieval and reduce dependencies. These help to exploit the positive aspects of interruptions and find ways to reduce negative ones.

Background

The Concept of Interruptions

Software engineering research and information systems research distinguish two fundamentally different concepts of interruptions: Software engineering research conceptualizes interruptions as a form of impediment, that is, something that delays individual work processes, such as handovers, extra processes, or unnecessary motion (Power & Conboy, 2015). Information systems research, especially in the context of technology-induced interruptions, conceptualizes interruptions as nonpredictable events caused by external sources that interfere with a primary task, such as incoming emails, calls, or coworkers requesting information (Addas & Pinsonneault, 2015; Grandhi & Jones, 2010; Perlow, 1999). These diverging perspectives highlight the complex nature of interruptions in the workplace with the perspective of interruptions as a distraction in getting a task done and the perspective of interruptions as intrusions necessary to complete a task.

In order to include both perspectives, we use a broad understanding of interruptions as events that impede or delay organizational members during work tasks (Jett & George, 2003). This includes events in the work environment that interrupt individual software development team members (such as distractions by colleagues), but also impediments for tasks that hinder individual software developers to complete a task (such as missing information; Jett & George, 2003). This broad perspective allows us to understand a variety of sources and consequences of interruptions. Following this line of thought, it is important to understand the interrupted task, the interruption content, the timing and quantity, the interrupting individual as well as the consequences of an interruption (Galluch et al., 2015; Grandhi & Jones, 2010; Jett & George, 2003).

Interruptions have negative consequences, such as increased time pressure for the task that was interrupted, procrastination, and mediocre performance (Grandhi & Jones, 2010; Jett & George, 2003). However, interruptions are considered to have positive consequences as well, including being helpful for informal feedback and information sharing, individualizing work pace, enhancing performance, and mindful information processing (Bechky & Okhuysen, 2011; Chua et al., 2012; Jett & George, 2003). Interruptions are associated with increased creativity, as they prompt attention shifts toward different perspectives, increase knowledge transfer, increase team learning through interactions, and invoke a wake-up call on routine work to conscious information processing (Watson-Manheim et al., 2012; Zellmer-Bruhn, 2003). Further, unexpected breaks are associated with increased performance if they allow employees to uphold attention to the primary task (Pendem et al., 2016).

Interruptions in Agile Software Development Teams

Agile software development teams are groups of software developers who jointly work on the development of new or modifications of existing software systems (Matook & Maruping, 2014; Tripp et al., 2016). Agile software development teams use agile approaches such as Scrum, eXtreme Programming (XP), or Kanban, with Scrum being the most common agile approach applied in practice (Bazigos et al., 2015; Rigby et al., 2016). In Scrum, there are three dedicated roles: the development team, consisting of developers who implement product functionalities; the product owner, who represents the customer and is responsible for the team delivering business value; and the scrum master, who is accountable for removing work impediments to the software development team (Lee & Xia, 2010; Maruping et al., 2009; Moe et al., 2012; Rigby et al., 2016).

Two agile characteristics suggest that interruptions occur in agile software development teams. First, high involvement with customers and fast responses to change interrupt agile software development teams during planning and in iterations (Moe et al., 2010). Second, agile approaches emphasize individuals and direct interactions, which requires coordination, and involving other team members, which causes interruptions of processes by others (Conboy, 2009; Matook & Vidgen, 2014).

The high uncertainty involved in agile software development projects imposes interruptions on the team when gathering and validating requirements as well as during development decisions (Drury et al., 2012; Moe et al., 2010). Agile developers report excessive interruptions by customers who ask for updates or provide additional requirements (Fægri et al., 2010; Tanner & Mackinnon, 2015). Further, agile software development teams are interrupted by new information that causes them to fundamentally change their plans (Moe et al., 2012). Agile software development teams use a high number of meetings, including daily standups, planning sessions, and burndown charts (Przybilla et al., 2018; Stray et al., 2016; Tripp et al., 2016). These impose breaks on software development team members, and the distributed decision-making causes discrepancies and additional work (Hoda et al., 2011; Stray et al., 2016).

The close collaboration in agile software development teams causes interruptions when team members ask for help or when bottlenecks occur (Stjerne et al., 2019). While agile software development teams benefit from knowledge sharing and a shared mindset within the team, the amount of direct communication is high and causes interruptions (Ghobadi & Mathiassen, 2017; Hummel et al., 2013; Kudaravalli et al., 2017; Przybilla et al., 2018; Recker et al., 2017, Rigby et al., 2016; Vidgen & Wang, 2009). Agile software development teams rely on informal control mechanisms, such as clan control, that establish group norms and shared beliefs (Chua et al., 2012; Wiedemann & Wiesche, 2018).

Agile practices, such as daily standups and pair programming, create additional interruptions (Conboy, 2009). Daily standup meetings are used to obtain an overview of project progress and to discuss problems, but developers sometimes describe these as inefficient and focusing too much on status reporting (Stray et al., 2016). During pair programming, developers are interrupted by peers during joint activities, instantly correcting code written by the developer (Balijepally et al., 2009). In addition to these very low-level interruptions, today's work practices in software development teams involve developers simultaneously working in multiple teams (Cameron & Webster, 2013; Przybilla et al., 2020). This interrupts the developer, whose tasks fundamentally mix even throughout a workday, but also interrupts the software development team, as a missing team member can cause a delay in decision-making, quality assurance, or other path dependencies (Power & Conboy, 2015).

While extant literature illustrates a plethora of interruptions in agile software development teams, an integrated perspective remains elusive. The perspectives of information impediments (Power & Conboy, 2015) and procedural interruptions (Grandhi & Jones, 2010) are not well integrated. This research seeks to develop an integrated account of events that impede or delay software development team members during work tasks (Jett & George, 2003). This integrated view will help to develop an understanding of how teams make use of agile approaches to develop individual reactions, tactics, and responses to cope with interruptions. Whereas studies report on individual practices of managing interruptions, how software development teams deal with a broad set of parallel interruptions, both positive and negative, remains unclear (Dingsøyr et al., 2018).

Methods

We conducted an exploratory analysis of four software development teams to answer our research question. We find a qualitative approach useful to understand the phenomenon of interruptions in its natural context, including individual perceptions and experiences (Miles & Huberman, 1994). We particularly chose four agile software development cases, as our phenomenon of interruptions has not been examined in the particular context of agile software development projects. We found a case-based approach useful, as it allowed us to empirically understand the entangled unfolding of the phenomenon between the level of individuals and the team (Yin, 2009). Simultaneously, it allowed us to use procedures of grounded theory methodology to include extant literature in software engineering, management, and information systems (Urquhart, 2012). While we were interested in how agile approaches help software development teams to manage interruptions, we did not limit our analysis to the team level. We found it particularly important to understand interruptions on the individual level and the corresponding reaction by the team members. We interviewed 19 agile team members and asked about interruptions they experience in their daily work. We asked about their current project and how agile techniques were used. We asked several questions about interruptions, including causes, involved actors, process, consequences, and the relationship to the agile approaches to project management. We explored these interruptions and particularly focused on understanding how agile approaches helped team members cope with interruptions. In Table 1, we describe the four teams we studied.

Data Analysis

For our analysis, we used procedures from grounded theory methodology to analyze our cases (Glaser, 1978; Urquhart, 2012). We applied theoretical sampling, as we selected interviewees and cases based on the analysis of previously collected data (Wiesche et al., 2017). For example, we sampled team MONITOR to gain a better understanding if the mechanisms we identified in Scrum hold for other agile approaches, such as Kanban. We constantly compared our data with extant literature, as well as other data points collected before.

We followed Glaser's guidance in applying open coding and selective coding (Glaser, 1978). We used open coding to broadly identify the interruptions, their effect on the individual and the software development team, and the mechanisms of how they coped with the interruption. We applied selective coding to link interruptions and develop an understanding of how agile practices help software development teams handle interruptions. The Appendix provides an illustration of episodes of observed interruptions and our open and selective codes, which serve as the basis for our analysis. We used memoing to (1) summarize key insights of interviews, and (2) develop our conceptual understanding of how agile software development teams handle interruptions. The author followed the concept of constant comparison by conducting the coding in parallel to data collection (on average after every fourth interview) to ensure that initial insights were reflected against extant literature and deepened in the following interviews and across cases (abductive logic; Glaser, 1978; Urquhart, 2012). We further validated our results by presenting them at conferences (Wiesche, 2018) and discussing them with practitioners.

Results

Our analysis identified three categories of interruptions across all four teams (Table 2). Interruptions were related to the task of developing software (programming-related), the collaborative nature of agile work (interaction-imposed), and to context variables (imposed by environment). In the following, we document

Table 1. Case Overview

Team Acronym	CAR	FLEET	MONITOR	REAL ESTATE
Agile approach	Scrum	Scrum	Kanban	Scrum
Industry	Automotive	Automotive	Insurance	Banking
Software product	Customizable online store for car configuration	Website to support company fleet management	Automation, administration, and monitoring of insurance software	Information system to support real estate financing consulting
Characteristics				
Colocated	Yes	Yes	Yes	Yes
Iteration length	2 weeks	Flexible (between 2 and 5 weeks)	Flexible (between 2 and 6 weeks)	3 weeks
Additional characteristics	-	High complexity through number of users and HR system connection	Focus on operating landscape of different insurance software systems	Formal release planning
Agile Practices				
Automated unit testing	Yes	Yes	No	Yes
Continuous integration	No	Yes	No	No
Backlog	Yes	No	Yes	No
Sprint planning	Yes (biweekly)	Yes (biweekly)	Yes (weekly)	Yes (every third week)
Definition of done	Yes	No	No	Yes
Daily standups	Yes	No	Yes	No
Daily customer involvement	No	Yes	No	No
Coding standards	Yes	No	No	Yes
Burndown charts	Yes	Yes	No	Yes
Retrospective	Yes	Yes	Yes	Yes
Kanban board	No	No	Yes	No
Ticketing system	No	No	Yes	No
Team Roles (No. of team members)				
Team members	4	8	4	9
Scrum master	1	1	-	1
Product owner	1	2	-	2
Developer	2	4	3	4
Tester	-	-	-	2
UX expert	-	1	-	-
Project lead	-	-	1	-
Interviewees	3	6	4	6

Table 2. Categories of Identified Interruptions

Category	Examples	Description
Programming-related work impediments	Malfunctioning code Poorly specified requirements Backlog changes	These interruptions are caused by programming-related work impediments, i.e., when the software development task is interrupted due to missing information, errors, or changes in plans. Other programming-related interruptions are the need for information provided by others that caused software development team members to postpone work until the information is provided.
Interaction-related interruptions	New customer insights Formal meetings Training team members Quality assurance	These interruptions are caused by the collaborative nature of the work in the software development team. The nature of agile approaches welcomes changes, and its different practices frequently require changing tasks and priorities. Tools and practices that foster collaboration (such as pair programming or standups) create interruptions for software development team members.
Interruptions imposed by the external environment	Telephone calls Breaks Multiteam membership	Such interruptions are caused by the external work environment. Examples include physical environment, such as the office setup, and the daily work practices, such as breaks and modes of communication. In addition, team members working in multiple teams are interrupted by the duality of their tasks.

each category by describing the interruptions, their consequences, and how the software development project team managed them.

Interruptions Caused by Programming-Related Work Impediments

The first category of interruptions we identified was caused by programming-related work impediments such as missing information, errors, or changes in plans. Examples of programming-related work impediments are malfunctioning code, poorly specified requirements, or backlog changes. We group these interruptions in three subcategories: requirement reconfiguration, missing information, and other information impediments.

Requirement-based interruptions occur when management reprioritizes requirements, customers struggle to fully specify requirements, or when the customer changes requirements. In project FLEET, the customer changed a requirement during an iteration. The team was working on a user story and the product owner identified changed text in a user story in their issue-tracking software Jira. The change was so fundamental, that certain features of the software product had to be changed. The developer described that:

> "the customer restated the user story. This slacked our pace. Totally different goal, resulting in different task for us … For the birds … We had to do another sprint planning and needed new code." (Project FLEET, developer 2)

Interruptions around the existing code occurred frequently when the developed software was not developed as a standalone solution but integrated into an existing application environment. When describing their regular work during each iteration, developers often noted that there were additional tasks occurring during the sprint. A developer described that:

> "in the hot phase, the [back end] was down for several days. And we had to do other stuff. The [back end] cannot be simulated. We needed to stop developing. Jump to other tasks like preparing unit tests or developing a shot in the dark." (Project CAR, developer 1)

Across all teams, interviewees highlighted the importance of the interruptions in solving problems and guiding the project. Especially the customer perspective was highlighted as fundamental to understand in which direction the project should go:

> "The customer is very important to give feedback on the progress and direction of the project … Of course, it is interrupting if that comes like this, but we try to prepare for this. There are regular project review meetings where we gather feedback from our management and customers … This helps us in identifying dead ends, problems, and new developments which we were not aware of." (Project REAL ESTATE, developer 3)

Agile software development project team members responded by channeling interruptions to the relevant colleague. All teams reported that the customer often did not follow agile practices exactly and interrupted different team members at different points in time, rather than postponing issues until the next formal team meeting. The teams developed mechanisms to channel these interruptions to the responsible colleague and ensure that they would take care of this issue later during the project phase. Usually, every developer knows what the others are working on and has a brief understanding of the feature that was commented on by the customer. They used digital tools like Slack or Jira to document the issue and assigned the correct team member. Thereby, the new detail was

neither lost nor did it interrupt the colleague directly. Team REAL ESTATE's tester 1 reported that he:

> "tr[ies] to capsulate [these interruptions]. I ask the customer for details, provide a first evaluation in terms of feasibility and time horizon, and then put it in the system. If I consider it urgent, I will raise the issue in the next daily meeting; otherwise, it will sit in the system, waiting for [developer 1] to resolve."

The teams FLEET, MONITOR, and REAL ESTATE reported that they scheduled "quiet time," where developers could concentrate on their work without being interrupted. The scrum master in team FLEET explained that in his team, developers showed up as early as 7:00 a.m. to get work done until 9:00 a.m. He estimated that during this quiet time, developers got 80% of their work done. After 9:00 a.m., work was interrupted by meetings such as standups, personal breaks, and socializing.

Interruptions Related to Team Interactions

The second category of interruptions was caused by team interactions. Due to the collaborative nature of agile software development, team members and external stakeholders frequently interrupted work processes within the team. These collaboration-imposed interruptions include new customer insights, formal meetings, training, and quality assurance. We grouped these interruptions in the encapsulating interruptions in agile practices, the usage of tools to track tasks, and the interaction with the customer.

Across all four cases, we observed that the teams formalized many unplanned interruptions in agile procedures. Especially the scrum master helped the team to encapsulate problems, issues, and discussions in continuously occurring Scrum meetings. Team members collected less urgent issues that needed discussion with fellow team members for the next regular Scrum meeting. Developers learned that if the consequences would not cause a long delay, they would decide to wait for the next, for example, daily standup meeting, rather than interrupting fellow team members. The scrum master explained that he decided on this way of managing the interruption, as he thought that it was more important to have a certain result ready that could be changed rather than a sudden stop in the coding.

One strength of agile practices is to put the customer at the core of the process. Many practices target updating, simulating, or integrating the customer and their needs. On the one hand, this is helpful in calibrating the solution, prioritizing, and planning the next steps, but it also interrupts the development process. Asked about how customers interrupt daily activities, one developer described:

> "They usually send emails. And they are the customer, so you have to drop all other work and respond … Stop your current task, understand the problem, solve the problem, and update [the] customer about the solution … You want to give a good impression, provide a great service and high-quality code. He is the one who pays the bill [;] that is why we do not bother about customer interruptions … they increase at the end of releases, I guess [,] because then, the customer "wakes up," but also, it is easier to criticize a working solution rather than imagining potential functionalities." (Project REAL ESTATE, developer 2)

In team MONITOR, developers reported that customers usually used an email-based ticketing system to report and track incidents. Only on particularly urgent issues were customers asked to approach the team via phone. However, developer 1 reported that some customers interpreted every issue as urgent and called immediately. If the call was not answered, they would come to the team's office, which was located in the basement of the headquarter building and interrupt the team. He explained:

> "It really is annoying. And takes you out of whatever you are doing. But we have a great work environment here, so you do not send them to heck but agree to help and ask for the matter. If it is something that takes less than 10 minutes, like rebooting a system, you just fix it immediately, just to get rid of the guy. If not, you just open another ticket and signal that you understood the urgency and importance." (Project MONITOR, developer 1)

Agile practices helped software development teams buffer interruptions in formal meetings. The sprint and the daily standups are the most important agile practices to reduce the number of unplanned interruptions related to development work within the team. Every morning, developers have the chance to raise issues where they need help. The sprint is an important vehicle to encapsulate the team from external interruptions to concentrate on development tasks. Across all cases, the scrum master tried to protect the team from unnecessary interruptions. One scrum master described that they:

> "try to keep all further interruptions from the team if possible … I consider a sprint successful when the developers can spend more than 80% of their time on tasks that we agreed on during sprint planning. So, I will do everything to protect my team from interruptions." (Team REAL ESTATE, scrum master).

Similarly, the customer is involved in sprint planning, where they can prioritize tasks to develop a focus and sprint review meetings to correct directions and demand changes to the current increment.

In teams FLEET and CAR, the project team scheduled additional meetings, which they referred to as refinement meetings. These meetings were scheduled in the middle of each sprint to formalize the continuous refinement process. One developer explained that he sees these meetings:

> "as artificial interruptions in an ongoing process. But the formalizing [in an official meeting] helps in updating the backlog and tracking progress. This opens up the view on the next

iteration that starts a week from now. You can do slight adaptions with an eye [on] what will come next week." (Project FLEET, developer 2)

Interruptions Imposed by the External Environment

The third category of interruptions relates to the work environment external to the software development team's task itself. Examples include telephone calls and work breaks. Here, the office setup, the daily schedule including breaks, and the way the teams worked together imposed positive and negative interruptions. In addition, team members who had other obligations due to multiteam membership caused interruptions as well.

One developer explained that he did not make use of the company's home-office policy very often. He described that this limited his ability to solve problems by reflection:

"I sit at home and work on a topic for four hours, so I need to set an alarm to get lunch … and continue afterwards. This is when you write a heck lot of code. But it cannot help you on thinking problems. When you need to reflect. Chances are high that you get on the wrong track with your solution. And then you are stuck. Here in the office, I can join the guys for a cigarette and I either share my current ideas or just by getting back to the screen after 5 minutes helps me think 'shoot, that can't possibly work this way.' I would call these breaks organic breaks." (Project CAR, developer 1)

Across our cases, we found instances where agile software development team members were not solely working on one single project at a time. We found that developers worked in different teams in parallel for several reasons: First, due to resource constraints, developers had to work on different client projects. Second, given their expertise with a particular topic, some team members supported other teams for particular tasks that required this particular expertise. Third, some teams did not require full-time work all the time, allowing developers to work on different projects in parallel. All these team members who worked in more than one team at a time caused interruptions when other team members required their input or when urgent issues overlapped:

"There are situations when you work in different projects in parallel. This causes interruptions as well. If you have two different strategies to follow. I had this once, kind of a firefighter job. I was working on [project A] and had to help out at [project B]. Right the next day. You don't question senior management decisions here, but of course, this is hard to plan. And to cope with." (Project FLEET, developer 1)

Across all teams, team members report interruptions related to nonwork-related incidents as well. These interruptions occurred on different channels, including telephone, email, and private surfing. The most dominant interruption was the smartphone.

"It takes time to get back to what you are doing. This is not helpful. And … you catch yourself once in a while doing stuff that is not work related. That is interrupting. Checking your phone, your newsfeed, social media. And there is a video that is more interesting or a link to something." (Project FLEET, UX designer)

In project MONITOR, the developers used a chat tool to post questions as soon as they occur. Several team members described that they used this chat tool permanently for small questions among each other. So, there were ongoing interruptions that even popped up on the developers' screens as soon as someone asked something. However, these were perceived as positive:

"[spontaneous things], I find these positive. It might not be sorted or queued, but there are these things of informal, loyal forms of collaboration that involves asking and helping, that I think [have] more advantages than disadvantages." (Project MONITOR, developer 1)

In addition, interruptions also helped team members take breaks and get their heads free. Several team members regularly went outside to smoke and used the break to think about the problem. Some team members described that the smoking break helped them to develop a different perspective or just to wait for another idea. One team member even described that she joined breaks for smoking, although she did not smoke:

"Usually, you get a coffee and discuss. But sometimes, I really need a break … And I do not smoke. And I love joining the smokers outside." (Project FLEET, UX designer)

All project teams used digital tools to reduce external interruptions related to information requirements. In every team, the developers used automation to reduce the number of interruptions related to software development tasks. Automatic deployment solutions reduced additional efforts from fellow team members during deployment and automated testing, such as unit tests, which reduced interruptions due to error detection and functional testing.

The teams regularly updated and shared their status documents, such as product backlog, sprint backlog, and burndown charts with external stakeholders to increase transparency in the development process. The team thereby reduced interruptions related to status reports, as external stakeholders, including management and customers, could access these systems to get a status overview of the project.

Within the team, developers created an informal hierarchy of tools for communicating an understanding of the urgency of the interruption. For example, in team FLEET, the developers used a chat tool for internal communication, a ticket system to

structure their work, and email to communicate with external stakeholders. Developer 2 explained that:

> "I immediately respond to requests via [chat tool]. Because we all know that the person asking benefits from a quick feedback. If not, he would have asked via [ticketing system] or sent an email. For these topics [ticketing system and email], I block some time in the afternoon and analyze and prioritize. So, this turns into a planned interruption."

Similarly, in team CAR, fellow team members understood the sense of urgency by the medium of communication. One developer explained that a person rushing in the office is looking for urgent help. These colleagues are willing to pull someone out of their tasks to solve their problem. So, such problems are considered important enough to interrupt colleagues, as otherwise, these developers would be on halt with their work.

Discussion

Causes and Consequences of Interruptions in Agile Software Development Teams

Across the four cases, we identified interruptions in three categories. While some of these interruptions also occur in nonagile software development projects, our analysis revealed that the two agile characteristics—user involvement and close team collaboration—intensify the number of interruptions in agile software development teams. For example, changing customer ideas can lead to interrupting changes in the backlog or the reprioritization of requirements. Similarly, the constant interaction within the agile team increases interruptions by helping behavior, scheduled meetings, and malfunctioning code.

Across the three categories, we found interruptions where team members could not complete their tasks, because they were missing information. These interruptions are in line with the idea of interruptions as work impediments (Power & Conboy, 2015) and include missing information from fellow team members waiting for customer input and poorly specified requirements. Such impediments could be overcome by early integration and tight alignment with customers and operations departments (Maruping & Matook, 2020; Wiedemann et al., 2020). These interruptions were also caused by team configuration, such as restaffing, onboarding, or training team members, and assigning team members to multiple teams in parallel. In addition to these interventions, team members were actively interrupted by other team members and stakeholders requesting help, clarification, status updates, or progress reports. In line with the concept of interruptions as nonpredictable events caused by external sources that interfere with a primary task (Addas & Pinsonneault, 2015; Grandhi & Jones, 2010; Perlow, 1999), we found the collaborative environment causing many different interruptions for team members. Other nonpredictable events include telephone calls and voluntary breaks (e. g., for lunch or smoking).

Our analysis points to the need for a more thorough understanding of the agile idea of user involvement and close team collaboration by considering how the interplay of agile practices affects team dynamics such as dealing with interruptions (Moe et al., 2010). Our results highlight the importance of following all procedures in the agile approach to equip software development teams with tools to protect their work environment and avoid interruption overload (Tripp et al., 2016). If software development teams fail to balance the number of interruptions, they might not be able to achieve their project goal. Not being able to manage interruptions might explain the high percentage of the dysfunctionality of agile teams (Sutherland & Jacobson, 2020).

Our results further highlight the importance of the roles in agile software development teams. The scrum master needs to possess the resources and power to protect the team from too many interruptions. Further, the product owner needs to channel interruptions during the software development process to help agile software development teams cope with interruptions (Maruping & Matook, 2020). Our cases confirm the importance of meeting customer demands and therefore welcoming and even inviting customer interruptions (Grandhi & Jones, 2010). However, our analysis also suggests that team members need to educate customers in prioritizing requests and corresponding interruptions. The identified management approaches for interruptions could serve as valuable additions to nonagile teams and can be combined with existing interruption-information gathering solutions (Grandhi & Jones, 2015).

Across all these interruptions, we found positive and negative consequences on the team and individual levels. Positive individual level consequences include an increase in productivity and independence. These are accompanied by the negative individual-level consequences of discontent and stress. As studies already started examining stressors and strains in agile work environments (Huck-Fries et al., 2019), it might be worthwhile to examine which role interruptions play on sociorelational competencies in agile software development teams (Matook & Maruping, 2014). Team-level positive consequences include clarity and understanding of requirements, new ideas and perspectives on problems, efficiency, better flow of information, and ultimately better work results. Negative team-level consequences include delay, distraction, and reduced quality.

Understanding How Agile Approaches Help Teams Manage Interruptions: Improved Information Retrieval and Reduction of Dependencies

In our study, we found two mechanisms of how agile approaches help software development teams manage interruptions. First, our results suggest that agile approaches help teams develop

practices to improve information retrieval. Teams used various agile practices to structure information exchange (e.g., daily in standup meetings or when using the project backlog). Also, for each iteration, sprint planning and review meetings provide structure in communication.

In order to balance these information-retrieving requests, teams structured timeboxing, for example, through signaled focus time where developers used agreed-upon mechanisms to signal that they needed to concentrate to finish a task. Some teams used automation to reduce the number of interruptions related to software development tasks. Team members developed an informal hierarchy of tools for communicating an understanding of the urgency of the interruption. Finally, the agile team invited feedback on different stages of the project. Especially the product owner and scrum master roles balanced these feedback instances. The product owner gathered and channeled feedback during the project. External stakeholders like the customer are invited early and continuously to project meetings, allowing them to provide feedback and correct the direction of the project. The scrum master focused on protecting the iteration from additional interruptions and helped the team resolve inefficiencies in their development process.

Second, agile approaches offer software development teams various practices to reduce dependencies among team members and even external stakeholders. On the one hand, ex ante measures, such as self-organizing teams and release planning, were used to ensure a flexible plan that allowed developers degrees of freedom to make decisions. Similarly, practices such as pair or mob programming[1] and a Definition of Done were used to increase product quality and thereby reduce interruptions for error correction and other path dependencies. Finally, the backlog and issue-tracking software were used to increase transparency and thereby avoid interruptions due to reduced requests for status reports.

In summary, these two mechanisms—support information retrieval and reduce dependencies—are implemented in the strategies that both individual software developers and the project team as a group use to manage interruptions. On the team level, agile approaches help software development teams increase transparency to avoid unnecessary interruptions. Similarly, agile approaches invite interruptions from stakeholders, including the customer, yet buffer these in formal meetings. Thereby, the software development team gets regular feedback, but interruptions occur only at predefined points in the process. Finally, agile approaches help the software development team build resilience through procedural and technological support to cope with changes.

On the individual level, agile approaches help software developers gain a mindset of a helping culture, which increases openness for questions and avoids wait time for developers requesting help. Agile approaches further help individual software developers to prioritize interruptions. A symptomatic illustration is that across all cases, interruptions by the customer were seen as urgent and important events that require ad hoc reactions. Similarly, developers established routines and used tools to signal urgency. This helped in prioritizing interruptions based on sender, content, and channel. All teams implemented timeboxing to create noninterrupted time slots for software developers. Therefore, software developers withdrew from interruptions by signaling quiet time or even changed their working habits so they gained quiet time in the early morning or in home-office arrangements. This timeboxing created temporal brackets for work rather than long-term subgroups that affect team coherence and project performance (Carton & Cummings, 2012; Pflügler et al., 2018; Przybilla et al., 2018).

Limitations and Contributions

Our study is subject to limitations. First, we considered teams that actively used agile approaches and thus, we might have missed mechanisms that reduce interruptions within the process. However, we examined four different cases and asked interviewees about how agile procedures were applied and what the consequences were. Second, our analysis was mainly from an internal perspective, reducing external views on how gatekeeping was perceived from the outside. We interviewed one customer and asked respondents about how their actions were perceived by customers. Third, it is inherent to exploratory qualitative work that generalizing the results is challenging. For example, we derived our results from agile teams in large and professional organizational contexts; thus, they cannot be taken for granted for smaller, informal organizations like startups or open-source projects. In addition, there might be unique characteristics of the IT workforce (Riemenschneider & Armstrong, forthcoming; Wiesche et al., 2019) that limit generalization to other knowledge workers.

This study has implications for theory and practice. Practitioners may use the list of interruptions to identify and manage potential sources of interruptions in agile software development and other projects. For designing work environments, it is important to colocate or establish other mechanisms to exchange informal communication (e.g., through digital collaboration tools; Dery et al., 2017); however, managers are advised to leave space for quiet time and meetings. We identify information retrieving and reduction of dependencies as two project goals that require collaboration and thus interruptions. The illustrated agile practices help achieve these goals. We suggest that practitioners focus on project team members' individual preferences toward interruptions. Scrum masters are advised to observe team dynamics and how uninterrupted time slots develop. They can further encourage emerging routines for handling interruptions and train newcomers and virtual team members in these routines. We found that all teams used agile approaches to bundle interruptions and, therefore, suggest that scrum masters observe routines, analyze their purposes, and steer these into bundles of practices that structure interruptions.

Our analysis revealed several positive consequences of interruptions in agile software development teams that go

beyond individual-level research (Jett & George, 2003). The dominance of real-time tools in collaboration within the team highlights the importance of fast feedback during software development work. We found that this personal help reduces slack and duplicate work. Similarly, all cases highlight the importance of feedback from software development project stakeholders. While such feedback might interrupt a team's daily routines, it also reduces double work by ensuring the early recognition of the need for change and action (Bechky & Okhuysen, 2011). We further highlight that structured interruptions can solve problems by increasing communication or just explicating the problem. Finally, using the example of training fellow developers via pair programming, we show that short-term consequences of interruptions might be negative, but in the end will increase overall project performance.

We find that software development teams filter interruptions based on the context and the current work situation. While agile approaches provide practices that invite interruptions in processes, work setups, and decision-making, teams filter and channel useful interruptions and cope with hindering interruptions (Stray et al., 2016). These mechanisms confirm the importance of implementing separate roles in agile software development projects and underline the importance of the scrum master in protecting the team and removing impediments (Lee & Xia, 2010; Maruping et al., 2009; Rigby et al., 2016). While extant research discusses the importance of timely and constant customer feedback (Recker et al., 2017; Vidgen & Wang, 2009), we extend this view in suggesting a more nuanced perspective on time. Customers can help scrum masters protect the team by using prespecified agile practices like sprint review meetings or daily standups to provide feedback and course corrections.

Finally, our results highlight how different implementations of agile, namely Scrum and Kanban, affect agile software development projects. We found that interruptions differed between team MONITOR, which used Kanban, and the other teams that followed Scrum. Tasks in team MONITOR were smaller, reducing the impact of interruptions for the team. To cope with lengthy interruptions, teams CAR, FLEET, and REAL ESTATE used sprints as safety zones in iterations to protect the team from additional interruptions. Finally, our results highlight the importance of the role of scrum master in agile software development projects. When following Kanban, team MONITOR had a strong manager who planned the process and imposed the approach on the team, as well as external stakeholders.

Future research could extend our theoretical sampling to other approaches, such as Scrumban or DevOps, as this might provide additional evidence of mechanisms to handle interruptions, as well as distributed teams, as their way of collaboration differs from colocated teams (Nikitina et al., 2012; Wiedemann et al., 2019). Our finding that software development teams prioritize interruptions based on sender, content, and channel sheds light on the mechanisms of agile approaches to cope with and build resilience against constant change. Testing the boundaries of our model for operations-savvy approaches like DevOps, where developers are confronted with other contextual conditions such as criticality of maintenance would further help to generalize our results. Finally, future research could expand our findings to feedback mechanisms or other inhibitors of project productivity, such as multiple-team membership or team fluidity.

Appendix

Illustration of Coding Procedure

Summary of episodes of interruption	Open coding	Selective coding
In team REAL ESTATE, the scrum master explained that team members shared their status in daily standup meetings. In these meetings, developers could seek or give feedback instead of interrupting colleagues on other tasks. Thus, they postponed problems that were not that urgent until the next day's standup meeting. In addition, developers grasped an overview of issues, problems, and solutions that other developers experienced, which might become relevant for their work in the future.	*Formal meetings for structured information exchange*	
All teams experienced situations when the customer changed backlog items during the project. The less interrupting point in time for interruptions was when the customer changed the backlog during sprint planning or sprint review meetings. However, all teams reported that customers regularly changed items during the sprint.		*Information retrieving*
In team CAR, developers spend a lot of time clarifying requirements. While they strive to get as many details during planning sessions, during the sprint, things pop up or are changed by the customer, causing delay or double work. Team CAR invited the customer to daily standup sessions and implemented short reporting cycles to cope with these changes in requirements.	*Adaptive planning*	
Team FLEET's scrum master described the positive open-door atmosphere within the team, where all team members implicitly agreed on establishing a culture of reciprocal helping behavior within the team. Team members actively asked for help or advice and let other team members interrupt their work.	*Encourage helping culture to avoid wait times*	
Team MONITOR reported on an implemented phone ring, where the team's phone calls were automatically handed over to fellow team members if a call was not answered. This ongoing ringing was perceived as interrupting and they approached management to deactivate the phone ring.	*(re)structure information exchange*	

(Continued)

(Continued)

Summary of episodes of interruption	Open coding	Selective coding
Developer 2 from team FLEET described their open office space, which allowed them to easily collaborate on joint tasks and interact with others. However, he also described that sometimes—often in the afternoon—team members interacted with nearshore colleagues via Skype, which created a continuously high level of noise in the office, interrupting all coworkers.		
In team REAL ESTATE, the product owner was colocated with the team, sitting in the same room. Developers valued instant feedback and ongoing discussions, which avoided slack, delay, and double work.	*Inspiration from information exchange*	
Several developers described the work environment as helpful and inspiring, as there were occasional interruptions. Rather than solely focusing on a task, they found that discussing ideas, which they referred to as "thinking through a task," is helpful for problem solving. There were many situations where developers would struggle with finding a solution on their own and benefit from explicating their problem to colleagues and getting feedback and ideas.		
In team FLEET, one developer learned a new development framework, which was a planned task during their iteration. Afterward, she spent a couple of days training other team members using pair programming rather than working on new tasks by herself more efficiently. In this way, the team gained expertise and was more efficient on future tasks. As a senior developer, she reflected that her own contribution was in distributing knowledge within the team rather than solving backlog items. Similarly, developer 1 from team MONITOR explained the importance of training new team members not only in the basic understanding of agile approaches, but even more important, in the team's developed routines and ceremonies when using agile approaches in their particular work environment.	*Avoid rework in later phases of the project* *Ensuring high-quality standards*	*Reduce dependencies*
Both teams FLEET and CAR implemented internal and external status reports. Internal meetings document development status on a regular basis and identify impediments. External meetings give customers the chance to provide feedback on increments. Several teams used tools like Jira or Trello for providing transparency to internal and external stakeholders.	*Increase transparency*	

Acknowledgments

The author would like to thank the editor and the three reviewers for their constructive and developmental feedback throughout the review process. He would further thank the interviewees for participating in this study; both their time and their enthusiasm were greatly appreciated.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Note

1. Mob programming is an agile practice that follows the idea of pair programming. However, instead of only two developers working at one work station, the whole development team jointly works at one work station.

References

Addas, S., & Pinsonneault, A. (2015). The many faces of information technology interruptions: A taxonomy and preliminary investigation of their performance effects. *Information Systems Journal*, *25*(3), 231–273.

Balijepally, V., Mahapatra, R., Nerur, S., & Price, K. H. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *MIS Quarterly*, *33*(1), 91–118.

Bazigos, M., Smet, A. D., & Gagnon, C. (2015). *Why agility pays. McKinsey Quarterly* (12). https://www.mckinsey.com/business-functions/organization/our-insights/why-agility-pays#

Bechky, B. A., & Okhuysen, G. A. (2011). Expecting the unexpected? How SWAT officers and film crews handle surprises. *Academy of Management Journal*, *54*(2), 239–261.

Cameron, A. -F., & Webster, J. (2013). Multicommunicating: Juggling multiple conversations in the workplace. *Information Systems Research*, *24*(2), 352–371.

Carton, A. M., & Cummings, J. N. (2012). A theory of subgroups in work teams. *Academy of Management Review*, *37*(3), 441–470.

Chua, C. E. H., Lim, W.-K., Soh, C., & Sia, S. K. (2012). Enacting clan control in complex it projects: A social capital perspective. *MIS Quarterly*, *36*(2), 577–600.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, *20*(3), 329–354.

Dery, K., Sebastian, I. M., & van der Meulen, N. (2017). The digital workplace is key to digital innovation. *MIS Quarterly Executive*, *16*(2), 135–152.

Dingsøyr, T., Moe, N. B., & Seim, E. A. (2018). Coordinating knowledge work in multiteam programs: Findings from a large-scale agile development program. *Project Management Journal*, *49*(6), 64–77.

Drury, M., Conboy, K., & Power, K. (2012). Obstacles to decision making in Agile software development teams. *Journal of Systems and Software*, *85*(6), 1239–1254.

Fægri, T. E., Dybå, T., & Dingsøyr, T. (2010). Introducing knowledge redundancy practice in software development: Experiences with job rotation in support work. *Information and Software Technology*, *52*(10), 1118–1132.

Galluch, P., Grover, V., Thatcher, J., & Clemson University. (2015). Interrupting the workplace: Examining stressors in an information technology context. *Journal of the Association for Information Systems*, *16*(1), 1–47.

Ghobadi, S., & Mathiassen, L. (2017). Risks to effective knowledge sharing in agile software teams: A model for assessing and mitigating risks. *Information Systems Journal*, *27*(6), 699–731.

Glaser, B. G. (1978). *Theoretical sensitivity: Advances in the methodology of grounded theory*. Sociology Press.

Grandhi, S., & Jones, Q. (2010). Technology-mediated interruption management. *International Journal of Human-Computer Studies*, *68*(5), 288–306.

Grandhi, S. A., & Jones, Q. (2015). Knock, knock! Who's there? Putting the user in control of managing interruptions. *International Journal of Human-Computer Studies*, *79*, 35–50.

Hoda, R., Noble, J., & Marshall, S. (2011). The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology*, *53*(5), 521–534.

Huck-Fries, V., Prommegger, B., Wiesche, M., & Krcmar, H. (2019). *The role of work engagement in agile software development: Investigating job demands and job resources* [Conference session]. Proceedings of the 52nd Hawaii International Conference on System Sciences, Maui, Hawaii.

Hummel, M., Rosenkranz, C., & Holten, R. (2013). The role of communication in agile systems development. *Business & Information Systems Engineering*, *5*(5), 343–355.

Jett, Q. R., & George, J. M. (2003). Work interrupted: A closer look at the role of interruptions in organizational life. *Academy of Management Review*, *28*(3), 494–507.

Kudaravalli, S., Faraj, S., Johnson, S. L., & University of Virginia. (2017). A configural approach to coordinating expertise in software development teams. *MIS Quarterly*, *41*(1), 43–64.

LaToza, T. D., Venolia, G., & DeLine, R. (2006). *Maintaining mental models: A study of developer work habits* [Conference session]. Proceedings of the 28th International Conference on Software Engineering, Shanghai, China.

Lee, G., & Xia, W. (2010). Toward agile: An integrated analysis of quantitative and qualitative field data on software development agility. *MIS Quarterly*, *34*(1), 87–114.

Majchrzak, B. A., Beath, C. M., Lim, R. A., & Chin, W. W. (2005). Managing client dialogues during information systems design to facilitate client learning. *MIS Quarterly*, *29*(4), 653–672.

Maruping, L., & Matook, S. (2020). The multiplex nature of the customer representative role in agile information systems development. *MIS Quarterly*, *44*(3), 1411–1437.

Maruping, L. M., Venkatesh, V., & Agarwal, R. (2009). A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research*, *20*(3), 377–399.

Matook, S., & Maruping, L. M. (2014). A competency model for customer representatives in agile software development projects. *MIS Quarterly Executive*, *13*(2), 77–95.

Matook, S., & Vidgen, R. (2014). *Harmonizing critical success factors in agile ISD projects* [Conference session]. Proceedings of the 20th Americas Conference on Information Systems, Savannah, Georgia.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. SAGE.

Moe, N. B., Aurum, A., & Dybå, T. (2012). Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology*, *54*(8), 853–865.

Moe, N. B., Dingsøyr, T., & Dybå, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology*, *52*(5), 480–491.

Nikitina, N., Kajko-Mattsson, M., & Stråle, M. (2012). *From scrum to scrumban: A case study of a process transition* [Conference session]. Proceedings of the 2012 International Conference on Software and System Process (ICSSP), Zurich, Switzerland.

Pendem, P., Green, P., Staats, B. R., & Gino, F. (2016). *The microstructure of work: How unexpected breaks let you rest, but not lose focus*. Harvard Business School Working Paper Series. Harvard Business School.

Perlow, L. A. (1999). The time famine: Toward a sociology of work time. *Administrative Science Quarterly*, *44*(1), 57–81.

Pflügler, C., Wiesche, M., & Krcmar, H. (2018). *Subgroups in agile and traditional IT project teams* [Conference session]. Proceedings of the 51st Hawaii International Conference on System Sciences, Waikoloa Village, HI.

Power, K., & Conboy, K. (2015). *A metric-based approach to managing architecture-related impediments in product development flow: An industry case study from Cisco* [Workshop]. Proceedings of the Second International Workshop on Software Architecture and Metrics, Firenze, Italy.

Przybilla, L., Wiesche, M., & Krcmar, H. (2018). *The influence of agile practices on performance in software engineering teams: A subgroup perspective* [Conference session]. Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research, Buffalo-Niagara Falls, NY.

Przybilla, L., Wiesche, M., & Thatcher, J. B. (2020). *Conceptualizing fluid team membership and its effects in IT Projects: A preliminary model* [Conference session]. Proceedings of the European Conference on Information Systems, Virtual Conference.

Recker, J., Holten, R., Hummel, M., & Rosenkranz, C. (2017). How agile practices impact customer responsiveness and development success: A field study. *Project Management Journal*, *48*(2), 99–121.

Riemenschneider, C. K., & Armstrong, D. J. (forthcoming). The development of the perceived distinctiveness antecedent of information systems professional identity. *MIS Quarterly*.

Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). Embracing agile. *Harvard Business Review*, *94*(5), 40–50.

Slaughter, S. A., Levine, L., Ramesh, B., Pries-Heje, J., & Baskerville, R. (2006). Aligning software processes with strategy. *MIS Quarterly*, *30*(4), 891–918.

Stjerne, I. S., Söderlund, J., & Minbaeva, D. (2019). Crossing times: Temporal boundary-spanning practices in interorganizational projects. *International Journal of Project Management*, *37*(2), 347–365.

Stray, V., Sjøberg, D. I. K., & Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software*, *114*, 101–124.

Sutherland, J., & Jacobson, I. (Producer). (2020). *A better scrum with essence* [Webinar]. ACM SIGSOFT Webinars.

Tanner, M., & Mackinnon, A. (2015). Sources of interruptions experienced during a scrum sprint. *Electronic Journal of Information Systems Evaluation*, *18*(1), 3–18.

Tregubov, A., Boehm, B., Rodchenko, N., & Lane, J. A. (2017). *Impact of task switching and work interruptions on software development processes* [Conference session]. Proceedings of the 2017 International Conference on Software and System Process, Paris, France.

Tripp, J., Rienemschneider, C., Thatcher, J., & Clemson University. (2016). Job satisfaction in agile development teams: Agile development as work redesign. *Journal of the Association for Information Systems*, *17*(4), 267–307.

Urquhart, C. (2012). *Grounded theory for qualitative research: A practical guide*. SAGE.

Vidgen, R., & Wang, X. (2009). Coevolving systems and the organization of agile software development. *Information Systems Research*, *20*(3), 355–376.

Watson-Manheim, M. B., Chudoba, K. M., & Crowston, K. (2012). Perceived discontinuities and constructed continuities in virtual work. *Information Systems Journal*, *22*(1), 29–52.

Wiedemann, A., Forsgren, N., Wiesche, M., Gewald, H., & Krcmar, H. (2019). The DevOps phenomenon. *Communications of the ACM*, *62*(8), 44–49.

Wiedemann, A., & Wiesche, M. (2018). *How to implement clan control in DevOps teams* [Conference session]. Proceedings of the 24th Americas Conference on Information Systems, New Orleans, LA.

Wiedemann, A., Wiesche, M., Gewald, H., & Krcmar, H. (2020). Understanding how DevOps aligns development and operations: A tripartite model of intra-IT alignment. *European Journal of Information Systems*, *29*(5), 458–473.

Wiesche, M. (2018). *Toward a model of managing interruptions in agile IT projects* [Workshop]. Proceedings of the International Research Workshop on IT Project Management 2018, San Francisco, CA.

Wiesche, M., Joseph, D., Thatcher, J., Gu, B., & Krcmar, H. (2019). IT workforce. *MIS Quarterly Research Curation*.

Wiesche, M., Jurisch, M. C., Yetton, P. W., & Krcmar, H. (2017). Grounded theory methodology in information systems research. *MIS Quarterly*, *41*(3), 685–701.

Wiklund, K., Sundmark, D., Eldh, S., & Lundqvist, K. (2013). *Impediments in agile software development: An empirical investigation* [Conference session]. Proceedings of the International Conference on Product Focused Software Process Improvement, Paphos, Cyprus.

Yin, R. (2009). *Case study research: Design and methods* (4th ed.). SAGE.

Zellmer-Bruhn, M. E. (2003). Interruptive events and team knowledge acquisition. *Management Science*, *49*(4), 514–528.

Author Biography

Manuel Wiesche is Full Professor and Chair of Digital Transformation at TU Dortmund University, Dortmund, Germany. He graduated in Information Systems from Westfälische Wilhelms-Universität, Münster, Germany and holds a doctoral degree and a habilitation degree from TUM School of Management, Technische Universität München, Munich, Germany. His current research interests include IT workforce, IT project management, digital platform ecosystems, and IT service innovation. His research has been published in *Management Information Systems Quarterly, European Journal of Information Systems, Journal of Management Accounting Research, Communications of the ACM, Information and Management*, *Electronic Markets*, *IEEE Transactions on Engineering Management* and *MIS Quarterly Executive*. He can be contacted at manuel.wiesche@tu-dortmund.de