Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings

Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer and Armin Heinzl

Abstract—Achieving effective inter-team coordination is one of the most pressing challenges in large-scale software development. Hybrid approaches of traditional and agile development promise combining the overview and predictability of long-term planning on inter-team level with the flexibility and adaptability of agile development on team level. It is currently unclear, however, why such hybrids often fail. Our case study within a large software development unit of 13 teams at a global enterprise software company explores how and why a combination of traditional planning on inter-team level and agile development on team level can result in ineffective coordination. Based on a variety of data, including interviews with scrum masters, product owners, architects and senior management, and using Grounded Theory data analysis procedures, we identify a lack of dependency awareness across development teams as a key explanation of ineffective coordination. Our findings show how a lack of dependency awareness emerges from misaligned planning activities of specification, prioritization, estimation and allocation between agile team and traditional inter-team levels and ultimately prevents effective coordination. Knowing about these issues, large-scale hybrid projects in similar contexts can try to better align their planning activities across levels to improve dependency awareness and in turn achieve more effective coordination.

Index Terms— Large-scale software development, agile, hybrid, inter-team coordination, dependency awareness, planning alignment, information systems development

1 Introduction

Research and practice have gained many insights into the inherent mechanisms of agile software development, its beneficial effects on how teams adapt to change [1] and its ability to help team members collaborate more effectively [2], [3]. The rapid rise of agile software development beyond the comforts of small, co-located teams [4] into the setting of large-scale software development projects, however, has opened up a Pandora's box of uncharted challenges for the software industry. Undoubtedly, the fundamental structures and methods of small-scale agile software development require adaptation in order to suit the nature of complex and large endeavors with numerous active development teams at the same time [4], [5]. It has been suggested that hybrid approaches, combining traditional plan-driven and contemporary agile development, are actually often an adequate choice given the history and requirements of large, long-running and highly integrated development projects [6]–[10]. Some reasons cited in favor of continued traditional planning in such hybrid approaches include predictability, dependability, stability and effective use of resources [11]. Although these hybrid approaches have been promoted in both academic literature [6], [9] and industrial frameworks (e.g. Disciplined Agile Delivery [12]), little is known as to why they succeed in some cases and fail in others.

Despite its promising potential, the introduction of any agile elements into traditional structured methodologies incurs major costs for reconciliation and may in fact demand for entirely new forms of development project control [13], [14]. Particularly interdependencies between many, possibly even geographically distributed, teams of software developers demand for carefully coordinated action across teams [15], [16]. In fact, *inter-team coordination* has recently been ranked the most pertinent large-scale agile challenge demanding immediate research attention [17].

Whereas traditional plan-driven approaches aim at the resolution of dependencies by comprehensive planning, detailed roles and pre-specification of software requirements and implementation activities [18], [19], agile approaches build on the idea that developers can most effectively resolve dependencies collaboratively and iteratively through self-organization when dependencies become visible on a detailed task level [20]–[23]. Prior research suggests that large and complex development projects may actually benefit from combining the flexible, organic coordination inherent to agile teamwork and the structured, plan-driven coordination of traditional project management into hybrid forms [6], [9], [14]. The questions whether

- *Saskia Bick is with SAP SE, Germany. E-mail: saskia.bick@sap.com.*
- *Kai Spohrer is with the University of Mannheim, Germany. E-mail: spohrer@uni-mannheim.de.*
- *Rashina Hoda is with the University of Auckland, New Zealand. E-mail: r.hoda@auckland.ac.nz.*
- *Alexander Scheerer is with SAP SE, Germany. E-mail: alexander.scheerer@sap.com.*
- *Armin Heinzl is with the University of Mannheim, Germany. E-mail: heinzl@uni-mannheim.de.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

such a synthesis is indeed possible and how hybrid approaches affect coordination are of utmost importance as inadequate responses to interdependencies and ineffective coordination constitute major sources of project failures [24], [25].

For this reason, we aim to better understand the critical aspects of inter-team coordination in such hybrid forms of agile development and traditional planning and coordination. Despite few studies of successful hybrid projects [13], [26], prior work in this area of coordination in hybrids is scarce and there is only limited extant research to build upon [14], [27], [28]. As one consequence, the frequently reported coordination failures and issues in hybrid settings [11], [29] have largely remained unexplained. We therefore approach our research from a problem-oriented perspective and empirically investigate a case study where a hybrid form struggled to achieve effective coordination in a large software project. In particular, we address the question:

"How and why does a combination of traditional planning on an inter-team level and agile development on a team level lead to ineffective coordination in large-scale software development?"

We leverage diverse qualitative data from a revelatory case study of a long-running hybrid software development project observed at a large global software company where inter-team coordination was known to be an ongoing challenge. In this context, 13 teams consisting of more than 140 developers worked on a single, highly integrated system of standardized enterprise software, combining agile development on a team level with traditional planning and coordination on an inter-team level. This provided a valuable context to study the causes of inter-team coordination challenges not discernible in well coordinated units.

Following a Grounded Theory approach to qualitative data analysis of interviews with scrum masters, product owners, architects and senior management, as well as project documentation, product backlogs and wikis, the key contributions of this study are twofold, 1) we identify a lack of dependency awareness as a key explanation of ineffective coordination in hybrid development settings, 2) we show how this lack of dependency awareness emerges from misaligned planning activities between team and inter-team levels, particularly during specification, prioritization, estimation and allocation.

In addition to implications for research, we highlight practical recommendations arising from our study with the aim to help software practitioners identify and avoid critical pitfalls in order to achieve better inter-team coordination via alignment of planning activities and higher dependency awareness.

2 Conceptual Foundations and Related Work

In this section, we present the relevant foundations on agile, as well as on large-scale agile software development, including the key terminology. Further, we recapitulate the current state of research on the coordination of tasks in comparable systems of multiple teams, in particular focusing on the associated challenges. For the purpose of discussing this research, we use the term *team level* to refer to actors, actions and interactions *within* a single development team of a larger software development unit. The *inter-team level*, by contrast, comprises all institutions and activities *between* development teams of a larger unit, including *central teams* (CT) that act as coordinating entities between development teams.

2.1 Agile Software Development

Traditionally, approaches to software development relied on top-down planning and heavyweight process management with a focus on detailed specifications and thorough upfront design. The most prominent is the waterfall approach [18], [19], which aims at process oversight and predictability based on intense upfront planning, documentation and sequential implementation and deployment. Attempting to better meet the dynamic nature of today's software development business, *agile software development*, a more flexible, lightweight approach, gradually replaced the more rigid traditional software development approach [30].

In 2001, the Agile Manifesto [20] postulated, amongst others, values such as interaction, collaboration, working software and flexibility in response to change. In this paper, we focus on the most prominent agile methodology, namely *Scrum* [31]. This project management framework aims at small software development projects and established a set of new roles such as *scrum master* (SM), *product owner* (PO) and the self-organizing team [32]. A product owner is responsible for a product's business value and the scrum team's remaining product backlog – an ordered list of product requirements. The cross-functional, self-organizing scrum team consists of seven (+/- 2) developers and is supported by a scrum master who is the process facilitator and coach of the team. At the heart of Scrum lies the *sprint*, a time-boxed iteration of usually two or four weeks, during which the development team implements a shippable software product increment. During several institutionalized Scrum meetings, the team collaboratively and iteratively discusses requirements, identifies dependencies, estimates efforts and reflects upon recent experiences to fine-tune its own processes.

2.2 Large-Scale Agile Software Development

Originally, agile methodologies aimed at small projects [4], [8]. Increasingly, however, large organizations seek to benefit from the advantages promised by agile development on a small scale. Especially the flexibility and adaptability of the workforce constitute major improvements to the heavyweight traditional approaches to software development [33]. Several practices and approaches to scaling agile methods have been proposed over time.

One of the earliest described practices to address the scaling issue is the *Scrum-of-Scrums* (SoS) – a scaled form of the daily stand-up meeting practice where smaller teams are represented by their "ambassadors" [34]. The format of an SoS is the same as that for a single team's daily meeting in that three areas are discussed by each team representa-

tive: work completed, next steps and impediments. Resolution of impediments often involves resolving coordination challenges via team interfacing, negotiating responsibilities, etc. [35]. The use of the SoS practice has been documented in a practitioner-based experience report [36] where individual teams met daily and team leaders were reported to meet once a week in an SoS meeting. However, there is no independent empirical evidence of the use of this individual practice as a comprehensive scaling approach and the practice has also been noted to be ineffective [37]. In some cases, the SoS was replaced by other approaches such as communities of practice.

A *community of practice* (CoP), which is a group of experts wanting to deepen their knowledge of a common shared interest or topic, has been shown to be a key success factor in supporting knowledge management and coordination in large-scale agile software development [37]–[39]. These CoPs were formed around common topics of interest, passionate leaders, concrete agendas and supporting tools to create transparency. They were introduced as a large-scale agile practice to achieve: knowledge sharing and learning, coordination, common design principles and organization development. Wider adoption of the CoP approach remains to be seen in other organizations practicing large-scale development [37].

In response to the rapid adoption of agile software development, a number of industrial frameworks have been proposed by practitioners and agile evangelists in more recent times to adapt agile development to a larger scale. Examples include *Large Scale Scrum* (LeSS) [40], the *Scaled Agile Framework* (SAFe) [41], the *Nexus* framework [42] and *Disciplined Agile Delivery* (DAD) [12]. All such commercialized frameworks seek to provide guidance for large organizations, which want to scale agile from the team to the inter-team level, by offering specific principles, roles and practices. While Nexus, LeSS and SAFe take the approach of scaling agile bottom-up from the team to the inter-team level, DAD meanwhile promotes a mixed approach that combines Scrum with best practices from multiple methodologies such as Extreme Programming, Rational Unified Process, Kanban and others.

All of the described scaling frameworks, however, have a common tendency to assume a greenfield setting to build on. Yet, especially large, established software vendors often do not face such legacy-free development targets and structures. In our study, we therefore want to focus particularly on hybrid structures of agile team level and plan-driven inter-team level approaches. Despite the plethora of approaches and frameworks, there is a dearth of research evaluating their effects in the software industry. Research has shown that companies rarely adopt software development frameworks in their entirety but usually pick and combine single elements of different ones to suit their needs [43], [44]. Critical views on this common practice label the commingling of traditional approaches and agile approaches *Water-Scrum-Fall* [45] or *ScrumBut* [46] criticizing the lack of procedural discipline while at the same time acknowledging the difficulties organizations are facing during an agile transformation. However, apart from this, almost no research has addressed customized or otherwise combined approaches to large-scale agile development.

2.3 Challenges to Scaling Agile

Applying agile methods and practices on a larger scale brings along challenges and difficulties [7] such as a higher number of stakeholders, additional complexity in coordination efforts and increasingly difficult architectural integration [11], [37]. A better understanding of these challenges is key to harnessing the benefits of agility in large scale settings [47]. Yet, research on agile on a large scale still remains scarce [17], [27]. Some empirical studies illustrate individual effects of large-scale agile development [9], [48]–[52] and report that agile on a large scale contributes to knowledge sharing through an emphasis on collaboration and at the same time increases coordination effectiveness. These valuable studies, however, do not provide more in-depth explanations of how to understand and overcome the challenges of large-scale agile development. In general, team level agile methodologies, including Scrum, do not easily transfer to larger projects because preferences and priorities between multiple teams tend to differ strongly [53]. Moreover, the organic coordination of large numbers of tasks and individuals logistically and cognitively overburdens project members [11], while numerous stakeholders pose distinct process requirements on larger and thereby more costly and critical projects [14]. Consequently, also business-relevant decisions can rarely be achieved on the team level through timely and direct customer interaction [29], [47]. Based on the heterogeneous nature of existing software development organizations, scholars have therefore suggested adapting and customizing agile development to its environment, particularly to the size of the development endeavor [43], [52], [54].

2.4 Hybrid Approaches to Scaling Agile

In response to the described challenges to scaling agile approaches to large settings, so called *hybrid approaches* combining agile and traditional plan-driven methodologies have increasingly gained attention in research and practice [6], [10], [11], [55]. By definition, the term *hybrid* characterizes something "of mixed character" or "a thing made by combining two different elements" [56]. Hence, the definition of a hybrid traditional-agile approach includes a wide spectrum of the combination of traditional, structured and agile approaches. Existing conceptualizations accept that for many different areas, organizations can opt for either agile or structured and plan-driven practices thereby creating a huge variety of hybrid approaches [14]. Similarly, Barlow et al. [6] conceptualize a hybrid methodology for large, mature organizations that combines aspects of agile and plan-driven methods. For this study, we build on Barlow et al.'s [6] research and conceptualize a hybrid approach as a combination of traditional, plan-driven methods with agile methods. Key characteristics of a hybrid approach include those displaying:

1. characteristics of two or more different approaches, e.g. hierarchical, top-down communication from a traditional approach and bottom-up adjustment [23] seen in an agile approach.
2. practices of two or more different approaches, e.g.

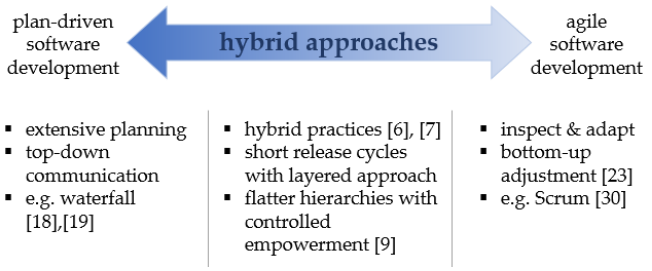


Fig. 1 Spectrum of Hybrid Approaches between Plan-Driven and Agile Software Development

extensive upfront planning in a traditional approach and regular inspections and adaptations from an agile approach.

3. roles from two or more different approaches, e.g. traditional project managers and scrum masters and product owners from an agile approach.

Hybrids are therefore not a single, discrete type of method. Much rather, they represent a huge variety of mixes within a spectrum between the extremes of purely traditional and purely agile development (see Fig. 1).

Depending on the project size, volatility and interdependencies, a hybrid approach focuses both on agile aspects such as mutual adjustment through strong social networks, as well as on traditional project planning with formalized information and knowledge transfer [6]. Hybrid approaches may therefore be particularly reasonable in large and distributed project settings with high impact that nonetheless need to cope with changes in requirements, organizations and technologies [14], [55]. For example, Scrum has been successfully embedded in structured approaches based on the PMBOK (Project Management Body of Knowledge) framework or CMMI (Capability Maturity Model Integration) in such cases [14], [55]. Batra et al. [55] conclude that elements of structured project management (such as formal management of skills and architecture together with traceable requirements and documentation) can serve as architectural foundations to maintain order and predictability throughout a project. Contrastingly, agile elements (such as time-boxed development sprints and actively prioritized backlogs) can add to a project by providing the means for sensing and responding to dynamism and uncertainty [55].

Several studies focus on the introduction of agile practices in traditional, large development projects [13], [26]. As such, Mahadevan et al. [13] find that hybrid approaches more likely show a mix of outcome control and emergence control compared to the traditional waterfall approach. Interestingly, the introduction of agile elements to traditional plan-driven approaches has been found to result in more decision power for the business side rather than the development side of large projects [13]. Moreover, studies emphasize that project uncertainty and completion urgency require a neat fit of project teams' information technology capabilities and implemented organizational controls [26].

In particular, agile teams are found to stand out in such contexts by developing either sensing capabilities for relatively foreseeable needs for change, or responding and learning capabilities for situations of unforeseeable change [26]. This resonates with findings by Ramasubbu et al. [14] who observe that project contingencies such as requirements volatility, novelty and the need for process compliance foster a mix of different frameworks within single projects and find that combinations of multiple frameworks are only beneficial for project success if project management actively enforces compliance to the defined mix of frameworks and processes, whatever that may look like in particular [14].

Lastly, studies have noted the adverse effects of hybrid contexts where teams were practicing agile methods in non-agile or more traditional environments [11], [29], such as challenges to project productivity and delayed releases. Reported challenges range from the creation of requirements, and the alignment of methods to differing opinions about processes and standards. In particular, the adaptation and alignment of software development methods and practices between the two opposing development approaches in an environment of team-of-teams structures seems to cause organizations severe difficulties [29]. However, the root causes of such issues remain largely unclear in the literature.

In sum, prior work has proposed different hybrid approaches and emphasized their numerous potential benefits. As hybrid approaches are reported to impose various challenges, scholars underline the important need for coordination across all teams involved in a development endeavor. Extant work has associated this coordination with both activities and capabilities of project management teams [13], [14], as well as of single development teams [26]. Lamentably, however, these valuable contributions to our knowledge have not yet ascertained where the frequently mentioned coordination challenges in hybrid settings stem from, why they arise and how the different actors within and across development teams are involved in not only resolving but also in causing these challenges. In fact, research attempts to unravel the detailed interactions within and between teams in large-scale agile development settings have only just begun [16]. For this reason, we engage in theory building to lay an explanatory foundation for further investigations.

2.5 Inter-Team Coordination

Applying agile methodologies in large-scale settings brings along the challenge of added complexity and a much larger number of involved stakeholders naturally arising from a larger organizational context. As such, multiple, interdependent teams that work together on a common goal represent a so-called team-of-teams structure. In the organizational literature, this structure of "two or more teams that interface directly and interdependently in response to environmental contingencies toward the accomplishment of collective goals" [57] is called a *multiteam system* (MTS). In such an MTS, the single teams have individual proximal goals and share a common distal goal with all other teams while they have input, process and outcome dependencies with at least one other team [57].

The traditional view of dependencies [58] implies the need for activity alignment in order to resolve dependen-

cies. Crowston [59] and Strode et al. [60] provide categorizations, which focus on the types of entities between which the dependencies exist, e.g. task, resource, knowledge and technical dependencies. In this paper, we refer to *task dependencies*, i.e. producer-consumer dependencies. These occur when one task has to be finished before another task can be started [60], [61], if the first task remains unfinished, the second task is blocked.

In order to manage existing dependencies within and between teams, *inter-team coordination* is of major importance. While literature holds an abundance of definitions for coordination, we make use of the definition of coordination as the management of dependencies [59], [62]. When speaking of inter-team coordination, we explicitly refer to the coordination of activities between two or more teams within an MTS. Extant literature presents several types, directions and mechanisms to categorize coordination. Espinosa et al. [63] distinguish between two coordination types: explicit coordination, which consists of mechanistic (e.g. plans, rules and routines) and organic elements (e.g. mutual adjustment and feedback), as opposed to implicit coordination, which consists of cognitive elements (e.g. shared mental models, team expertise and transactive memory systems). Based on this, Scheerer et al. [16] conceptualize *top-down planning* as a mechanistic, centralized approach with vertical coordination, e.g. between a team and a hierarchically superior person or a central team. The archetype on the other end of the spectrum is called *bottom-up adjustment* and represents a largely organic and decentralized strategy with horizontal coordination between teams of an MTS [16], [63], [64].

The performance of inter-team coordination has been measured in terms of coordination success and effectiveness [65], [66]. Both concepts are similar in their foundations and for this study, we chose coordination effectiveness as an outcome variable for our research. *Coordination effectiveness* is defined as a state of coordination, where all members of a software development MTS have a comprehensive understanding of the common goals and priorities, what is going on and when, what they as a team need to do and when, which team is doing what and how each team's work fits in with another team's work [65]. The few studies that investigated antecedents of coordination effectiveness looked at the influence of coordination strategy [67], knowledge sharing [68] and team environment complexity [69], amongst other factors. Particularly in software development organizations, however, ineffective coordinating reactions to dependencies are known to be a major cause of project failure [24], [25]. A comprehensive understanding of the necessary conditions for coordination effectiveness is still lacking, though.

3 Research Design

In our research, we aimed to understand the emergence and root causes of ineffective coordination in hybrids of traditional and agile approaches. While research and practice agree that reconciling agile and plan-driven approaches into hybrid forms appears necessary [26], [55], the frequently reported coordination failures and issues in doing so [11], [29] have largely remained unexplained. Due to this lack of existing explanations, we engaged in theory building based on empirical data [70], [71]. To do so, we focused on understanding the central mechanisms that prevent effective coordination in hybrid settings by studying a case with known coordination challenges. By understanding these detrimental mechanisms, we aimed to conceptualize a process model of effective coordination in hybrid settings. Based on the identified negative mechanisms, we describe the type and sequence of *necessary but non-sufficient* conditions that must happen as antecedents of effective coordination.

Process theorizing is well established in the literature and complements variance theories by providing a qualitative and temporal causal argument [72]–[75]. Importantly, a process model describing the relation X → Y does not read "more X therefore more Y" but rather "the occurrence of X is necessary but not sufficient for Y to occur" [74]. Our process model of effective coordination in hybrid traditional-agile development settings therefore describes necessary events that occur as prerequisites of effective coordination. By definition, this is not a comprehensive explanation of effective coordination, but a model proposing that coordination can only become effective when the antecedents exist. While such an explanation may not provide a silver bullet for successful coordination, it emphasizes and explains the roots of important challenges.

In order to build our explanatory model, we conducted a single, revelatory case study [70], [76], [77] in an ongoing, large-scale software development project at a large enterprise software company. We had the unique chance to conduct a case study that promised to be particularly informative for our research because it clearly exhibited a hybrid form of traditional and agile approaches to large-scale development that reportedly suffered from ineffective coordination. In fact, we were approached by several of the company's agile development experts in early 2013 who regarded this project as particularly interesting because it mixed agile and traditional approaches, and the experts wanted to know how this would turn out.

During a first informal talk, it became clear that heavy coordination issues existed in the project and we were invited to study those issues and their possible relation to the hybrid setting. While individual development teams of the project applied agile practices following the latest standards, there was a clearly articulated desire by management to retain traditional elements of product and project management in order to coordinate activities across the many teams involved. Despite the fact that the developed product had been successful and development could somehow keep the pace of the market, there were reportedly frequent frustrations because of ineffective coordination between the different development teams. As a consequence, this project provided a very rich and insightful case for exploring whether and how events of ineffective coordination related to the combined application of traditional and agile development approaches. Given the dearth of theory explaining coordination failures in hybrid settings and the unique promise of very rich data access to such a reportedly negative hybrid case, we deemed it appropriate to

conduct a single case study for the purpose of building a theoretical mid-range model [77], [78].

3.1 Case Description

The studied large software development unit was developing a complex and successful standard enterprise software with 13 development teams in four locations spread over Germany, India and China. With a total of around 140 employees, each development team consisted of six to 16 developers. The average company tenure of developers was 12 years, the average tenure in this development unit was five years and teams and individuals were well accustomed to working with each other. The on-premise enterprise software solution had been developed, refined and extended for more than 10 years. Despite its market success, the product had become quite complex due to a wide range of customers with diverse requirements. Moreover, the product architecture was highly integrated, exhibiting a tight coupling of product functionality.

The single development teams were, with few exceptions, co-located and usually had exclusively assigned product owners and scrum masters. Several developers, architects, UI designers and quality engineers completed the competence profile of each cross-functional development team. All development teams worked in the Scrum mode and applied agile development practices such as iterative development, daily stand-ups, sprint and release planning meetings, sprint review meetings, team retrospectives and agile techniques such as pair programming to different intensities throughout the teams.

For planning and coordination, there were synchronization cycles across the whole development unit of four weeks, which also resembled the length of the development sprints for the teams. On a strategic planning level, release planning cycles were reported to comprise up to several years.

The official process for planning and coordinating the work of more than a dozen teams involved quite a popular traditional strategy, namely establishing a central team specifically for that purpose. In this case, the central team was a non-developing team, consisting of the chief product owner (CPO), architects and product experts, a total of seven members. The central team was responsible for the high-level product planning and facilitating coordination of activities across the multiple development teams. The central team did not include any product owners or other representatives from the individual development teams that it coordinated. The central team met daily for one hour to discuss current and pressing topics as well as the plans for the upcoming sprints. Once per sprint, the central team hosted a status and hand-over meeting with each of the development teams' product owners separately. In this meeting, the central team wanted to see what had been achieved by the product owner's development team during the last sprint and communicated the backlog items and their priorities for the next sprint in a directive manner. In more detail, the central team generated high-level work items in the form of epics and allocated those to the individual development teams. An *epic* described high-level requirements and was progressively broken down into more granular *user stories* and subsequently into technical *tasks* as it passed down from the central team to the individual teams and then to individuals within teams (see Fig. 3).

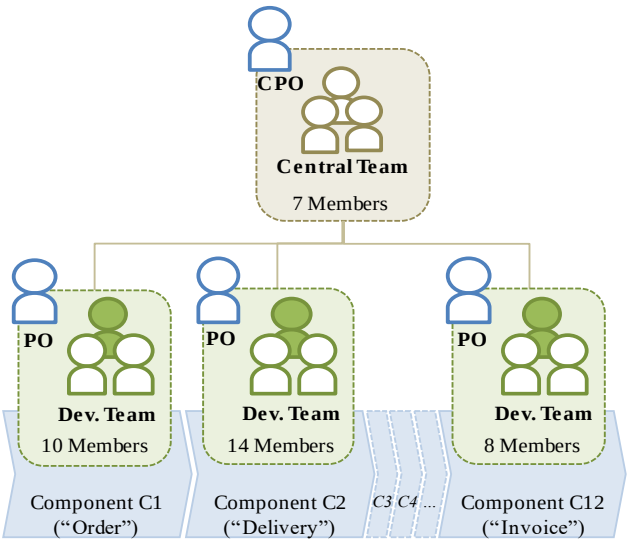


Fig. 2 Multiteam System Organizational Setup

Fig. 2 depicts the setup of the multiteam system, i.e. the large software development unit. In the depicted setting, the central team was responsible for coordinating the single development teams. It thereby presented a widely applied form of traditional higher-level planning and coordination via roles and hierarchies. At the same time, the single teams were working in sprint-based iterations, consisted of empowered team members and applied agile practices applicable to their contexts. Overall, the case featured two central elements of a hybrid approach: top-down planning via a central team and agile software development methods and practices on a team level.

In the depicted organizational setup, the individual teams frequently faced mutual dependencies due to the highly integrated and tightly coupled nature of the software codebase. In fact, the software codebase resembled a business process with sequentially interdependent components (exemplified in Fig. 2 for a procurement process). In the studied case, development team 1 was responsible for the business process component C1 "Order", i.e. developing that particular part of the enterprise software which concerns the creation and processing of a customer order

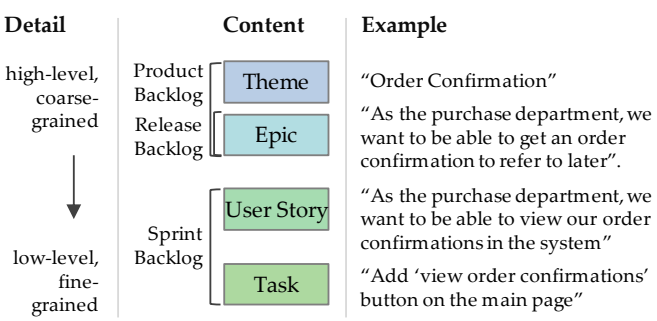


Fig. 3 Backlog Hierarchy

in a procurement process. In sum, the case at hand constituted a conducive basis for the examination of planning activities in hybrid traditional-agile approaches to software development and their consequences for inter-team coordination.

3.2 Data Collection

To gain a detailed understanding of the case, we collected qualitative data between October 2013 and November 2014. During this time, we conducted 23 semi-structured interviews with key informants of the development unit. Our interviewees included the CPO, the lead architect, 11 product owners (who usually had a strong technical background) and 10 scrum masters (who usually also performed as developers and had extensive experience of software development) from 13 development teams located in four different international locations. These informants were chosen based on their roles as they could best report on inter-team coordination issues and provide a comprehensive perspective of both the team level and the inter-team level. Instead of focusing on single developers who view a project from their team-internal perspectives, we actively chose to focus on roles that were directly engaged in the definition and management of work on an inter-team level (CPO, lead architect) or the diffusion and reflection of work packages on a team level (product owners, scrum masters).

Where possible, we conducted interviews face-to-face (10 interviews), or else via phone (13 interviews). All 23 interviews were recorded, summing up to 15 hours of audio material, then transcribed (approximately 330 pages of transcripts) and subsequently analyzed with the qualitative data analysis software NVivo 10. Additionally, we used internal documentation and project management data to triangulate our findings and increase analytical validity [76]. In particular, we collected organization charts from wiki pages to learn about responsibilities, members and distribution of development teams, i.e. the organizational structure of the rounsoftware development unit. We analyzed backlog management system data to understand the content, size and specification of backlog items. Backlog management data also allowed us to cross-check our findings from the semi-structured interviews. Product architecture maps helped us to better understand the product and the specialization of individual development teams, ultimately allowing us to check for or rule out alternative explanations for observed coordination inefficiencies.

3.3 Data Analysis

We analyzed the interview data using Grounded Theory (GT) [71] to capture the emergent themes and findings. In doing so, we employed GT's open, selective and theoretical coding procedures. These coding procedures were conducted by two of the authors individually. Emerging themes were compared and examined and were also discussed extensively amongst the other authors to ensure reliable coding. Where categorizations of events differed between the authors, the particular events were discussed intensively until consensus was reached.

To explain how we applied these procedures in this study, an example of working from raw interview data to the findings of one of the categories, *'misalignment of planning activities'*, is presented. First, we summarized portions of the interview transcripts into two to four words referred to as *codes*.

Raw data: *"What is missing today and what we need is* ***collective planning****… If you really have to deliver an end-to-end process, you* ***need to align each and every team*** *and* ***you need to plan their developments*** *completely in a collaborative fashion. That is something that we have* ***not been able to achieve*** *very well so far."*

Code: *need to align planning;* **Code:** *planning alignment not working*

It was clear that there was a need for planning alignment but at the same time it was not being achieved effectively. Comparing codes from within the same interview and with those from other interviews, we grouped the codes into a higher level of abstraction called *concepts*. In this example, the concept that emerged was *'misalignment of planning'*.

Concept: *misalignment of planning (in general)*

In addition to misalignment of planning in general, other concepts capturing misalignment of specific planning activities also emerged. These concepts included *misalignment of: specification* and *prioritization (of requirements), estimation (of effort)* and *allocation (of tasks)*. Together they captured the misalignments that were seen across four planning activities leading to the next level of abstraction called *category,* in this case *'misalignment of planning activities'.*

Category: *misalignment of planning activities*

Similarly, the second main category, *lack of dependency awareness* was identified. For example, the presence of occasional *task dependencies across teams* was strongly evident from multiple interviews. At the same time, it was obvious that there was a *lack of communication of tasks and progress* from one team that led to a *lack of awareness of tasks and progress* for the other teams and vice-versa. This led to teams making *inaccurate assumptions* about what to expect and what not to expect in terms of task dependencies. Overall,

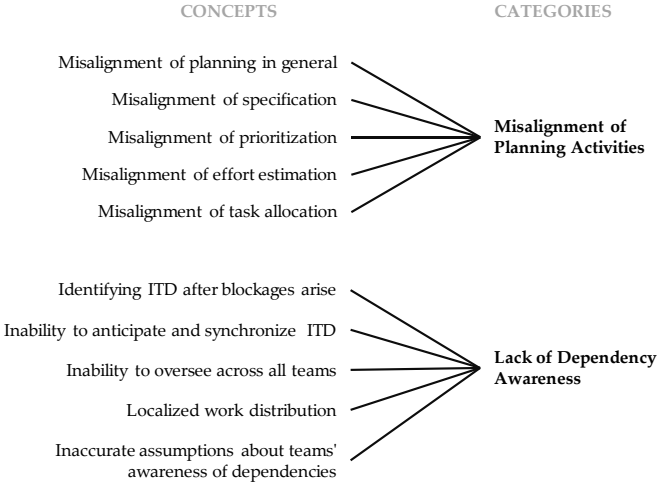


Fig. 4 Emergence of the categories 'misalignment of planning activities' and 'lack of dependency awareness' from underlying concepts through open and selective coding

these concepts highlighted a clear *lack of dependency awareness* across the teams.

Fig. 4 shows the emergence of the two categories, *misalignment of planning activities* and *lack of dependency awareness,* from their respective underlying concepts and codes. The third main category, *ineffective coordination*, was not only a known problem from the beginning of the case study but was also supported well through the data analysis in similar ways.

Further details of open, selective coding and other GT procedures can be found in GT's seminal texts [71], a recent review [79] and a dedicated paper the use of GT in software engineering [80]. The final step of GT's data analysis is theoretical coding, which involves conceptualizing the relationships between the main categories as a set of interrelated propositions [75], [80], [81]. The following section describes the main categories. The propositions, i.e. interrelationship between the categories, and the formulation of the overall theoretical model is described and discussed in section 4.3.

The categories and their inter-relationships emerged from data analysis and were not preconceived at the outset of the study. Nevertheless, they partly relate to concepts present in extant literature. When causal inter-relationships emerged, we therefore went from analyzing data to examining the literature in order to sharpen the emergent categories until they were closely tied to both the events depicted in our data and established concepts from the literature. This procedure of dialogical reasoning allowed us to properly account for the events unraveling in our case while basing arguments and conceptions on strong findings of prior work. We illustrate this for the category *lack of dependency awareness*: When we identified a dominant theme in our data that teams suffered particularly from being unable to identify and address dependencies on the work of other teams, we turned to the literature to understand already existing conceptions of this phenomenon. Prior work has shown that higher levels of work dependencies increase the failure proneness of software development projects [82] and has attributed this to individuals' inability to detect implicit workflow interdependencies and to keep track of changing coordination requirements over time [82]. Accordingly, related work has started to create tools aiming to increase individual developers' awareness of task interdependencies with the ultimate goal to facilitate more effective coordination [83]–[85]. This stream of work thereby helped us more precisely define dependency awareness with vocabulary already present in research (see TABLE 1) and reinforced our inductively gained understanding that the absence of dependency awareness inhibited effective coordination across teams. Such an approach is supported by the basic tenets of GT whereby related literature can be consulted in the later stages of theory development to support and explain the emergent findings [71].

4 Case Study Results

In the following, we present the results of our study. First, we elaborate on events of ineffective coordination in the studied case. Due to the development unit's relatively long and stable history, the fact that there was cultural diversity between the development teams did not noticeably influence the collaboration between the teams. This was consensually affirmed in our interviews. Similarly, we could not find any evidence that coordination challenges were related to the distributed or co-located settings of the individual teams. In other words, the reported issues clearly happened between the team and inter-team levels, regardless of the location or cultural background of the development teams. We first show that ineffective coordination was primarily caused by a lack of dependency awareness across teams. We then go deeper into the analysis of root causes for this lack of dependency awareness.

4.1 Dependency Awareness

Our investigation revealed that the coordination of activities on an intra-team level appeared to work well. Based on nearly text book-like Scrum, the single teams discussed requirements, opened issues, potential problems and resolved dependencies within their team on a daily basis.

Interviewee 11 (SM): "Every day, we discuss the top issues that need to be solved. […] This is good, because everybody is on the same page. Everybody knows what issues are of top priority and what everybody needs to work on. This daily sync is very effective."

TABLE 1
WORKING DEFINITIONS OF KEY CONSTRUCTS USED AND DERIVED

Construct	Working definition	Primary references
Hybrid	a combination of traditional, plan-driven methods with agile methods	[6], [55]
Planning activity alignment	The degree of coherence between specification, prioritization, estimation and allocation on team and inter-team levels	[22], [31]
Dependency awareness	the state of all of the system's relevant stakeholders (on both the team and inter-team levels) having identified, recognized and established a shared understanding of the existence of interdependencies and potentially resulting alignment issues	[82]
Coordination effectiveness	a state of coordination, where all members of a software development MTS have a comprehensive understanding of the common goals and priorities, what is going on and when, what they as a team need to do and when	[65]

In the monthly planning and review sessions, all team members including the team product owner were involved. In the daily stand-up meetings, the whole team was present. These institutionalized meetings provided room for discussion and most importantly instant feedback from team members. The scrum master facilitated these team-internal discussions and took care of impediments that potentially kept his team from doing their job. At the same time, the product owner clearly specified new backlog items and was available for questions coming from the team. Through describing open tasks and next steps in their development efforts, individuals shared insights into potential sources of dependencies with their team members.

In that same discussion space, team members were implicitly invited to speak up, share their thoughts and help resolve ad-hoc dependencies in an organic manner. During a sprint, the scrum master blocked new tasks that were not initially scheduled for the current sprint. In contrast, on an inter-team level, the development unit was facing severe coordination challenges. Several product owners reported situations where their entire development team was blocked by some unforeseen event. This was most frequently caused by an unidentified dependency with another team. In reaction to this type of situation, problems were frequently escalated to the central team. Escalation meant reporting the issue to the central team and thereby signaling a team's inability to complete its own tasks for the current sprint. This was reported to happen several times per sprint throughout the entire development unit.

Interviewee 7 (PO): "At some point it becomes clear that something is wrong and then, depending on the priority of the topic, we have internal escalations […] and can't proceed with what we were doing. […] we have meetings, reprioritize and try to solve the issue."

That is, through escalating an issue to the inter-team level via the central team, a team usually passed on the responsibility for the situation at hand, instead of taking action and attempting to resolve the source of the conflict directly with the other involved team(s). This happened as a result of the hierarchical setup of the central team as the main coordinating authority on the inter-team level. At the inter-team level, there were hardly any higher-level epics one team alone could work on. Cross-team requirements were very common. Therefore, not only the developed product, which represented a tightly integrated business process solution, but also the development teams were highly interconnected. Interconnections seemed to cause problems if the points of contact in terms of handovers, deliverables and dependencies in general were not clearly defined upfront. The problems did not appear in those situations, however, where the central team could identify and communicate the majority of the potentially critical dependencies between epics they assigned to the development teams. Even if problems arose during the sprint, resolution could then usually happen in a controlled fashion.

Interviewee 7 (PO): "Then, you either get [the deliverable] when you need it, or […] it is at least defined in a way that we can re-estimate the effort. And if it is too much so we have to say that we cannot do it this sprint, we have to postpone it to the next. But then at least the situation is clear and not like you rely on their input which doesn't come eventually."

A much more problematic picture presented itself in situations where dependencies between teams had not been identified and anticipated upfront. This led to teams blocking each other, escalations in the middle of a sprint, delays and a high level of general discontent in the development unit.

Interviewee 4 (PO): "[…] team 1 builds something and somehow it is forgotten to talk with team 2 and 3 and in the end something doesn't work because somebody expected that the requirement which was built by team 1 is also taken into consideration in the other teams."

Often, teams did not know of other teams' activities and thereby were not aware of requirements towards them. Many of these task dependencies appeared to be unknown to at least one of the involved parties. Once a development team was at the point where it needed the input of another team and realized that for whatever reason the required input could not be delivered in time, the observed reaction was often an escalation to the central team.

Interviewee 2 (PO): "Something was built […] and other teams didn't know about what was going on, and then the (development) processes just stopped all of a sudden; or you knew what was needed, but the capacity just wasn't sufficient […] then you're facing escalations."

Clearly, the central team's approach to ensuring good coordination across the teams was to prevent dependencies from arising. However, this was largely seen as ineffective given the size of the central team in comparison to the size of the large software development unit and due the fact that no team representatives took part in the discussions.

These insights more and more raised the question as to what might have been causing the coordination issues between the teams. Statements by product owners and scrum masters indicated that teams did not seem to know about the requirements and dependencies of other teams. Especially those undetected dependencies that existed between one's own team and another fellow team were reported to cause severe problems. We noted a severe lack of *dependency awareness* in the studied case. That is, teams lacked "an understanding of the activities of others, which provides a context for [their] own activity" [86, p. 1].

Interviewee 4 (PO): "[…] This is what doesn't work at the moment – that the central team launches team 1 with topic 1, team 2 with topic 2 and team 3 with topic 3, although topics 1, 2 and 3 would also require the capacity of team 1, 2 and 3. So it's distributed locally and months down the road, you notice that the other teams should deliver something, too."

Having illustrated the conflicts and resulting problems that we saw, we can say that in a majority of cases where coordination issues became apparent, the reason was a lack of dependency awareness. Unidentified dependencies and covered potential conflicts led to ineffective coordination between the development teams which in turn had an impact on the teams' productivity and ability to deliver.

Without being aware of dependencies to other teams, a development team will neither be able to assess potential consequences, nor will it even think about ways to resolve these dependencies. Regardless of the size of the multiteam system, it is therefore necessary for all its members to recognize and come to a common understanding of potential or existing dependencies. We define *dependency awareness* as the state of all of the system's relevant stakeholders (on both the team and inter-team levels) having identified, recognized and established a shared understanding of the existence of interdependencies and potentially resulting alignment issues. In single, co-located agile teams, dependency awareness has not been identified as an issue [87]. Yet, to the best of our knowledge, research on dependency awareness in the context of hybrid approaches to large-scale software development is scarce [88]. Our case data suggests that dependency awareness is a necessary, though not sufficient, antecedent to effective coordination.

4.2 Misalignment of Planning Activities

Having identified a lack of dependency awareness as the critical factor inhibiting effective inter-team coordination, we proceeded with an in-depth analysis of the root causes that impaired dependency awareness in the first place. Data suggest that the source of the conflict lies in the planning activities of the development process.

> *Interviewee 14 (PO): "We are not able to pre-plan. […] The central team […] needs to have a significant amount of process clarity and then needs to reach the teams. Then every team needs to align their sprint according to that. […] If you really have to deliver an end-to-end process, you need to align each and every team and you need to plan their developments completely in a collaborative fashion. That is something that we have not been able to achieve very well so far."*

According to our data, we see that even before the actual implementation of requirements, several planning activities, namely specification, prioritization, estimation and allocation, were misaligned between the team and the inter-team levels of the development unit, which caused severe problems later on during the implementation. The misalignment of the four planning activities were concepts derived from the GT data analysis as described in section 3.3.

Particular aspects of a concept can be captured as *properties* in GT. 'Focus' was captured as a property of planning concepts. In other words, it reminds us that one of the main dimensions of misalignment was how the team and inter-team levels varied in their focus, where the former was more content-oriented while the latter focused on time. That is, actors either concentrated on a defined scope of functionality that should be developed, or on a defined timeframe or time-box. For example, for the specification activity, it was found that in case of the inter-team level, the central team's focus was on getting a certain defined set of features developed, preferrably in the shortest time possible. On the team-level, in contrast, the focus was on a time-boxed cycle of four weeks during which as many user stories as possible would be completed. A time-oriented focus could range from daily to sprint-, or even release-based time frame. In our case, a release usually spanned 12 sprints, i.e. almost one year.

Similarly, two other cross-cutting dimensions of misalignment emerged: *degree of detail* and *competence*. The dimension *degree of detail* captures the extent of details present in the requirements expressed at each level. At the inter-team level, for example, requirements were more abstract, coarse-grained concepts, such as themes or epics. At the team-level, on the other hand, requirements were broken down further and the degree of detail was fine-grained on the task or user-story level. This content hierarchy is depicted in Fig. 3.

The third dimension *competence* refers to the specific technical knowledge, expertise and project experience that enables a certain role or entity responsible for the planning to perform their job. For example, the central team's competence to perform coarse-grained specification at the inter-team level was based on long-standing market knowledge, customer insight and a high-level view of the release plan; which the individual development teams usually lacked. On the other hand, the product owners' and the teams' competence to perform a detailed and fine-grained specification was based on detailed technical knowledge of their specific product components and collective team expertise; which the central team did not possess. Incorporating the above dimensions of misalignment, we define *planning misalignment* as "the incompatible arrangement of focus, degree of detail and competence in planning activities on the team level in relation to the inter-team level". The findings are summarized in Fig. 9 and described in detail in the following sections.

Specification

Before starting to implement a user story, the desired functionality first had to be clearly discussed and elaborated. As depicted in Fig. 3, there was a hierarchy regarding the degree of detail of the specification. By specifying a user story or task, the relevant and necessary information regarding functionality, design and performance needs was stated in a concise way. On the inter-team level, the central team specified coarse-grained requirements with a clearly content-oriented focus (Fig. 5). Their market knowledge, customer insight and a high-level view of the current release plan enabled the central team to provide such a high-level specification. Through the up-front specification of high-level epics, the central team generated a coherent picture of the functionality that had to be developed.

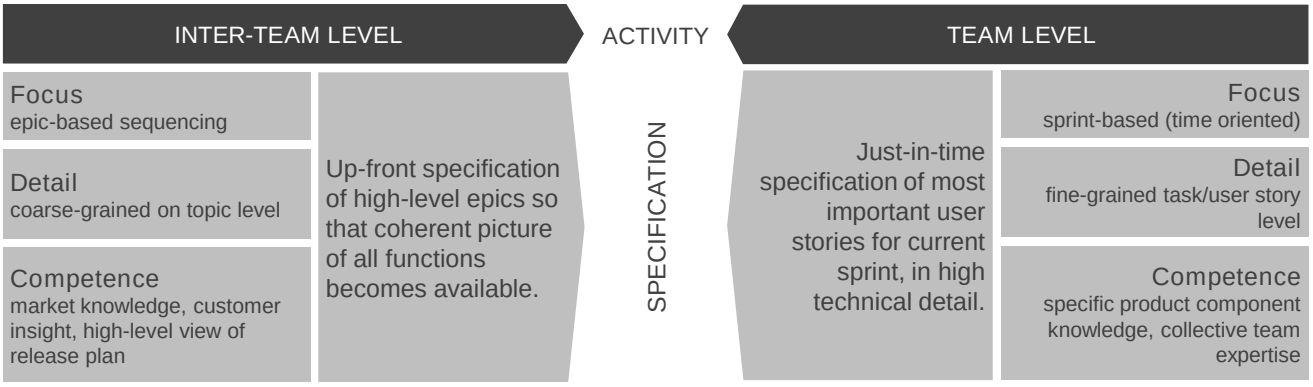


Fig. 5 Specification Misalignment Dimensions

Interviewee 8 (PO): "[…] there's a central team which we have, who toss a coarse-grained backlog item, an epic, into the team. […]Once you get to the implementation it's a whole new story. […] when it's about tossing the stuff into the scrum teams, what they need is not what they get, cause the teams have work left to finish [from previous sprints]."

On the team level, we observed a much more time-oriented focus. That is, the product owner together with his team detailed out fine-grained user stories or tasks in a sprint-based manner right before the sprint in which the respective user stories should be implemented. The product owner's competence to do the fine-grained task specification was based on specific product component knowledge and the collective expertise of his team that he represented. Due to the time-oriented focus, only the few user stories were detailed out that were to be implemented in the according sprint. Those, however, were elaborated with a high degree of technical detail during refinement sessions that were facilitated by the scrum master. If dependencies with other teams were discovered during these sessions, this consequently happened at the time when the respective other teams were also detailing out their own user stories already and had closed their backlog for the respective sprint. A resolution of such newly discovered dependencies thus interfered inherently with the idea to specify user stories collaboratively and just before implementation.

Viewed separately, both the central team as well as the product owners and scrum masters acted in a reasonable way considering their goals and their areas of competence. However, for an active alignment of specifications across the two levels, we found two main impeding factors. First, the central team did not have the detailed technical view necessary to do the kind of specification that aims to prevent all sorts of potential dependencies up front, by identifying them early on

Interviewee 22 (CT member): "Usually we [the central team] cannot anticipate all dependencies between the teams. […] normally when we have problems with coordination, the fact that we don't manage to synchronize the teams in such a way that they can work in parallel without blocking each other comes up."

Thus, for a traditional top-down specification of modular tasks, which would be based on detailed documentation and a high level of insight, the provided degree of detail was simply too low. Since the central team did not gather and incorporate any feedback about detailed specifications coming from the development teams, they did not compensate for this shortcoming and the teams themselves were tasked with finding other teams affected by newly discovered interdependencies.

Interviewee 4 (PO): "They [central team] work on an item level with 2-3 bullet points per [epic]. […] that's what they dump on a team and this team then has to find the other teams that are affected by this [epic]."

Second, the teams did have the technical expertise to discover inter-team dependencies for their specified user stories right before their next sprint, but they lacked lead time and the high-level picture to resolve them with other teams. What is more, active sharing of their specification details with one another or with the central team was neither encouraged nor institutionalized. Therefore, discovered inter-team level dependencies were known to single development teams only and affected teams could not take them into account. Together, the specification misalignment, in the form of a lack of detail for a top-down task specification and a lack of cross-team collaborativeness and iterativeness for a bottom-up task elaboration led to a lack of dependency awareness.

Prioritization

Once specified, the work packages for the upcoming development iteration needed to be prioritized so as to explicitly state an order of importance in which they were to be implemented. In the studied multiteam system, the central team conducted an absolute, high-level prioritization of the epics they had specified based on their overall view of the release plan and market needs (Fig. 6). The epics they identified, however, mostly related to indispensable features of the product to be released. Therefore, they assigned mostly very high priorities to those epics and often had only little difference between the priorities of multiple epics.

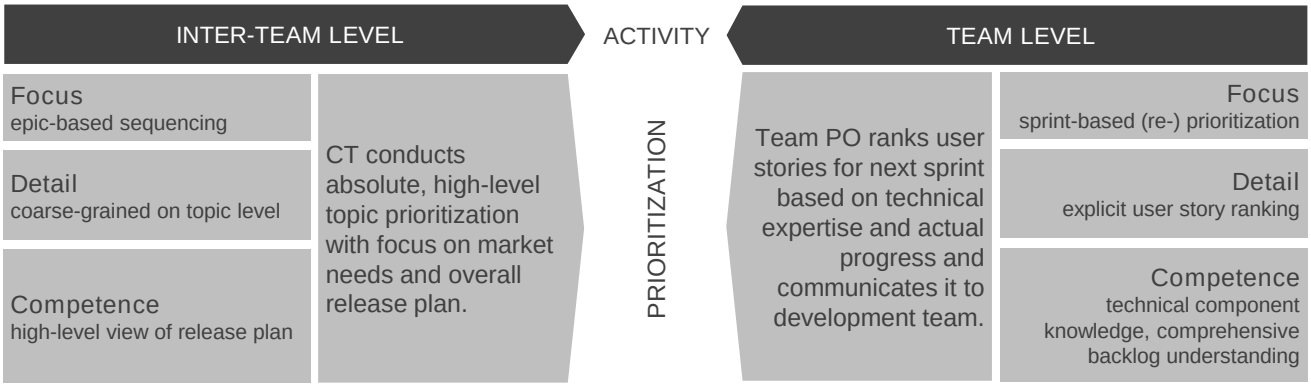


Fig. 6 Prioritization Misalignment Dimensions

Interviewee 2 (PO): "If you're already late [in the current sprint] and everything has been defined as prio 1, it doesn't make sense to put another prio 1 on top [for the next sprint]."

Rather than clearly differentiating priorities, the central team provided development teams with a sequence in which epics should be implemented. The effect could be described as an epic-based sequencing. On the team level, each product owner did a sprint-based prioritization and reprioritization of the current backlog items based on his technical component knowledge and a comprehensive backlog understanding. The product owner thereby created an explicit order by which the items were to be implemented. The results of the prioritization activity were then communicated to the development team.

Again, our data suggest two problems with the unaligned approaches between the inter-team and the team level. First, the central team set many, equally high absolute priorities for indispensable features from a business perspective and only infrequently reprioritized those. By consequence, teams had a hard time prioritizing left-over work from previous sprints compared to new epics. Second, through their local team level priorities, the teams resolved team-internal backlog dependencies. However, as development teams did not know about one another's reprioritizations and sequence of work steps, this frequently intensified the effects of existing inter-team dependencies.

Interviewee 22 (CT member): "An issue which we face […] was the mutual blocking of teams. Because they are strongly interconnected, we previously often had the situation that priority 1 from one team was priority 3 of the next team and these two teams actually had to work together."

Therefore, we can state that the rough prioritization on the inter-team level together with the nontransparent, potentially conflicting team level reprioritizations prevented dependency awareness.

Estimation

Each product owner and development team jointly estimated the exact efforts for open tasks of the following sprint through internal discussions (Fig. 7). Whenever new information caused the initial estimation to be unrealistic, tasks were re-estimated. In their collaborative discussion sessions, the whole team came up with carefully refined effort estimations, which were very close to reality. On the inter-team level, a coarse-grained effort estimation happened in a more content-oriented manner, based on epics. The central team roughly approximated the epic size relying on their long experience with this multiteam system and their collective expertise.

Since the central team did not have such a strong time-oriented focus as the development teams did, we observed a conflict with the team level time-boxing. Despite applying a best effort approach, the central team's estimations were often very imprecise, yet still constituted the basis for

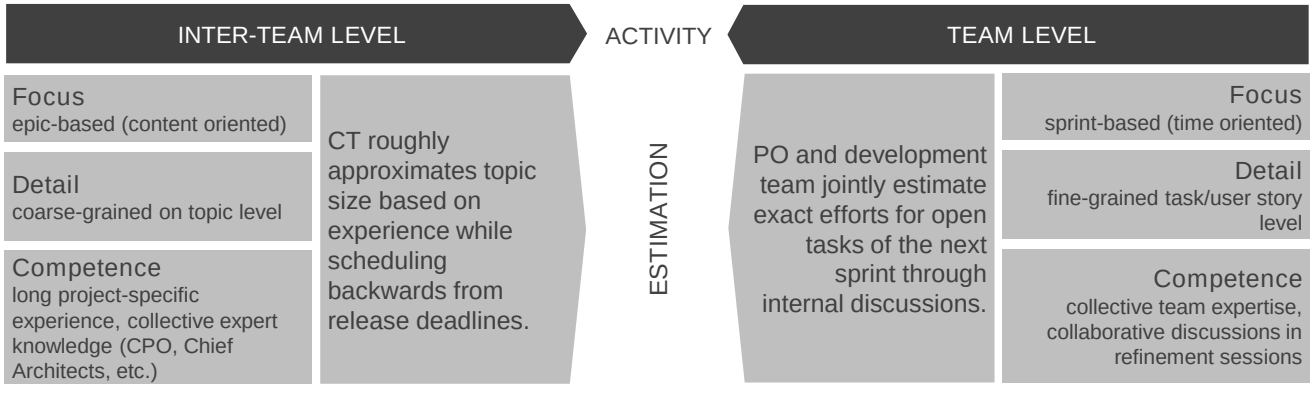


Fig. 7 Estimation Misalignment Dimensions

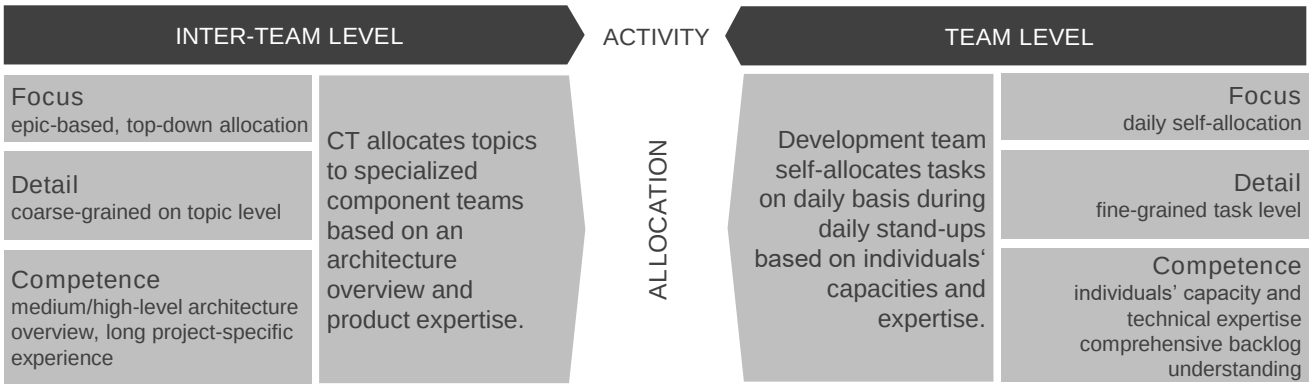


Fig. 8 Allocation Misalignment Dimensions

the distribution of epics to the development teams. Estimations on inter-team and team levels also diverged as the much more accurate estimations made by single teams were kept internally and could not be taken into account by other teams. As there was no standardized process during which the teams could communicate their collaboratively refined effort estimations, no alignment of estimation was accomplished.

Interviewee 22 (CT member): "[…] we give [epics] to the teams, the teams do the sprint planning […] and report to us what they achieved. Let's say, in 9 out of 10 times, they [the teams] achieve less than what we told them to do. So for the sprint thereafter, a good percentage of their capacity is already blocked."

The nontransparent team level estimations also prevented development teams from knowing exactly when to expect handovers from other teams. This could in turn intensify inter-team dependencies when teams were relying on others to deliver certain functionality in order for them to complete their own tasks. The estimation misalignment can be summarized as an imprecise top-down effort estimation and a lack of feedback mechanisms for bottom-up estimation adjustment. Together, this hindered dependency awareness.

Interviewee 2 (PO): "You have a monthly hand-over meeting [from the central team to each development team] and let me say, this is not effective at all, because it doesn't make sense if you have backlog items lasting for five more sprints, to get even more on top."

Allocation

After specifying, prioritizing and estimating work packages, the last step in the planning process was the allocation of the latter to a team for implementation. On the inter-team level, the central team allocated coarse-grained epics to specialized component teams based on an architecture overview and enabled through their long development unit-specific experience (Fig. 8). The epic-based and sequenced top-down allocation was meant to fit the teams' expertise and prevent inter-team dependencies but usually happened irrespective of detailed team level priorities and estimations.

The epic allocation seemed to be partially predetermined by a high degree of specialization of the development teams in parts of the business process (see also Fig. 2). Within the self-organized scrum teams, the allocation of tasks happened on a voluntary and daily basis during the daily stand-up meetings, based on the individuals' capacities and expertise. This process of task self-allocation was facilitated through the detailed specification and the explicitly ordered sprint backlog. On the inter-team level, however, there was very little communication or exchange about epics between the individual teams, just like in previously illustrated planning activities. The development teams were aware of other teams' general responsibilities, but knew only little about their currently assigned epics. This was also true for the teams'product owners who were the only ones to receive their teams' epics for the next sprint from the central team. These handovers happend during one-on-one sessions between a single product owner and the central team and thereby did not allow for gaining insights into other teams' epics, either.

Clearly, the upfront allocation of epics without a detailed specification, prioritization and estimation prevented the central team from identifying and communicating dependencies. This resulted in a lack of dependency awareness. What is more, many inter-team dependencies were only discovered by the development teams during their sprint planning. However, lacking lead time and a high-level overview, there was rarely any chance for them to adjust or reallocate epics across teams ad hoc.

Interviewee 22 (CT member): "Often we think that [epics] are independent and give them to the teams only to have the experts come back to us and tell us, if I should implement this, then another team has to implement something first and the other team doesn't have time right know."

4.3 Development of a Process Model

Theoretical coding (as explained in section 3.3) finally allowed us to understand the inter-relationships of the main categories: We found that the misalignment of planning activities had constantly caused a lack of dependency awareness, which had generally resulted in ineffective coordination. Based on this analysis, we are therefore led to believe that effective coordination was not possible without dependency awareness which, in turn, was not possible without aligned planning activities between team and inter-team levels. We elaborate on this in the following.

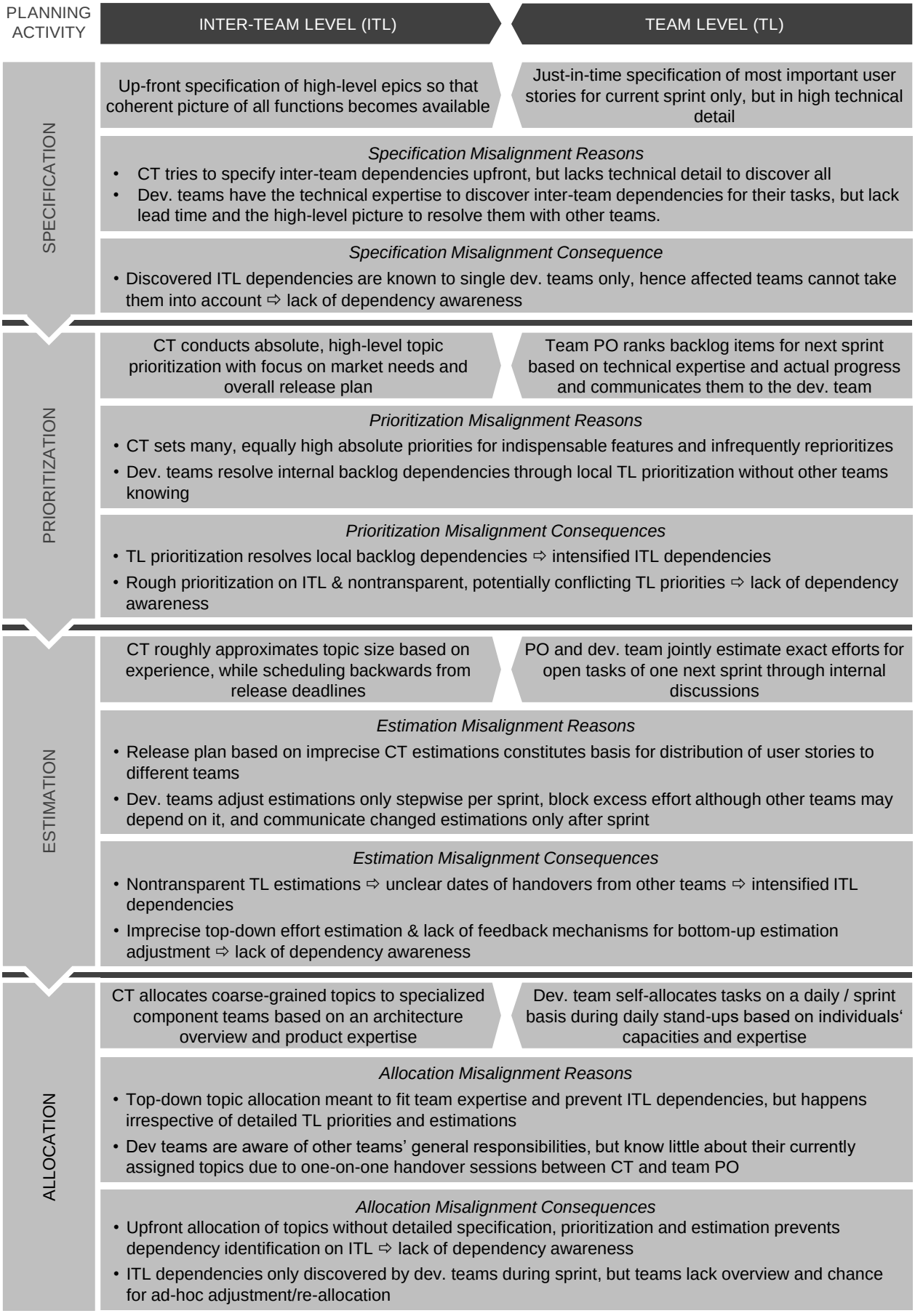


Fig. 9 Effect of Planning Misalignment between inter-team and team level on Dependency Awareness

As suggested in section 4.1, all teams including the central team were frequently unaware of dependencies spanning
the boundaries of single teams. Quite reliably this resulted in a) unsuccessful endeavors by teams to adapt mutually once they became aware during implementation or integration, b) in escalations to the central team who could not easily resolve the dependencies and c) ultimately in process breakdowns and the need to postpone delivery.

Although a number of other reasons can also lead to ineffective coordination, lack of dependency awareness was clearly identified as the dominant one for the reported events of ineffective coordination in this study. Nonetheless, we investigated also for other potential explanations of ineffective coordination that were ultimately discarded as they did not fit what we observed.

As one example, ineffective coordination between teams could also be rooted in teams' inability to adequately react on observed dependencies rather than not being aware of them. We used the rare situations in which there was high dependency awareness to probe into this possibility. We saw that the teams were generally able to resolve dependencies they were aware of. For such known dependencies, the central team tried to resolve them already during planning. If this was not possible, the central team informed single teams about the expected interdependencies who sorted them out easily enough. For example, product owners and scrum masters of involved teams had phone calls several times a week to communicate local progress and facilitate mutual adaptation.

Interviewee 7 (PO): "Usually it works, if they [the CT] prepared it properly […] If you check all aspects again collaboratively and document it, then usually it works."

In sum, we found a consistent co-occurrence of ineffective coordination with, and only with, lacking dependency awareness. Data from our case therefore suggest that dependency awareness is necessary for effective coordination of all teams. However, its presence does by no means guarantee for effective coordination. Where only single development teams are aware of interdependencies, this precludes a resolution of dependencies through roles and hierarchy [89] as would be typical for traditional development approaches. If all information were available to the entire MTS, hierarchically higher institutions (such as a central team) could redefine roles of single teams and remodularize tasks to resolve interdependencies [58], [90]. Without knowing about all the actors involved and without knowing about the nature of the dependency, however, hierarchical mechanisms cannot function properly [89]. Mutual adaptation as would be typical for agile development [33], on the other hand, cannot work if the parties who would need to adapt mutually do not clearly share the same understanding of a dependency to resolve [91]. This leads us to propose:

P1: *Dependency awareness across both team and inter-team levels is necessary though not sufficient for effective coordination to occur.*

Moreover, misaligned planning activities on team and inter-team levels frequently caused our case teams, including the central team, to overlook dependencies. In our case, the central team was not specifying in as much detail as a traditional specification approach would require, neither were the teams able to collaboratively define and specify all tasks as would be common within a single agile team. Consequently, each team on its own was not able to identify its dependencies with other teams effectively during specification. As a result, there often was a lack of dependency awareness on an inter-team level, which inhibited effective coordination. This was particularly true because the central team, in its aim to prevent dependencies between teams, directly allocated epics top-down to the single development teams, mostly without consulting other teams. Breaking epics down into single tasks to work on, individual teams refined the planning which resulted in priority-based task sequences and detailed effort estimations that partly deviated from the central team's plan. When teams worked on isolated epics, this did not constitute a problem and was actually desirable. Frequently, however, there were dependencies that had not been spotted and local decisions inherently affected other teams. Consequently, dependencies across different teams that were not identified by the central team via pre-specification were usually only found far too late in a sprint when all other teams that would have been needed to resolve such dependencies via mutual adaptation had already committed to other highly prioritized work.

In our case, particularly misalignments between the planning activities conducted on a team level and those conducted on an inter-team level resulted in impaired dependency awareness. In other words, misalignment of planning activities was the dominant cause leading to a lack of dependency awareness. And we could, in fact, not observe any situations in which the teams managed to achieve project-wide awareness of any dependency without aligning their planning activities.

From a theoretical perspective, this appears quite reasonable as each misaligned planning activity between team and inter-team levels presents an individual chance to obfuscate dependencies and aggravate their impact. As such, purely local specification activities that detail out requirements on short notice clearly increase the chance of purely local discovery of interdependencies. Teams not involved in the specification cannot easily know about such a discovered dependency and much less share the same understanding of it. Although teams discovering dependencies may identify a technical need for adapting a specific component in order to proceed, they may lack the high level overview of which other team could address this need, i.e. who they actually depend on. Given the short time frame from specification to implementation typical for agile development, also the possibilities of ongoing intensive communication across teams are severly restricted, thereby undermining chances for more time-consuming communication-based coordination [91]. Single teams' planning activities further affect the sequence and timing

of their contributions to the overall product when they locally refine and change priorities and effort estimations in an agile way. Where other teams depend on these contributions, such changes impact those teams' freedom to act and execute their own plans. Without aligning these activities across teams, there can be no shared view that allows for mutual adaptation or hierarchical decision making to enforce a specific order. Lastly, the traditional top-down allocation of development tasks, even of coarse-grained ones, to single teams is bound to create unresolved dependencies in all complex projects [25]. When a team discovers such a dependency and does not know about other teams' task allocations, it becomes difficult to identify who they actually depend on. Although developers and teams can establish bottom-up routines to partition or re-allocate such tasks to appropriate others, this takes time and resources usually not spent in organizational settings where more efficient hierarchical coordination constitutes a viable alternative [92]. To make either hierarchical or bottom-up coordination possible, however, task allocations need to be aligned across single teams (and possibly central institutions) to provide all parties with the same understanding of dependencies. Based on this reasoning and the empirical support found in our case study, we therefore propose P2:

P2 *Planning alignment of all planning phases (specification, prioritization, estimation, allocation) is necessary though not sufficient for dependency awareness to occur.*

With propositions P1 and P2, we propose a process model of effective coordination in hybrid settings (see Fig. 10). Importantly, this does *not* mean "the more aligned the planning activities, the higher the dependency awareness and the higher the coordination effectiveness". Instead, it defines the sequence of necessary conditions that need to happen before effective coordination becomes possible. The model thereby holds that when planning activities are not aligned, this inhibits dependency awareness of all entities; and when there is no project-wide dependency awareness, coordination becomes ineffective [17], [54].

5 Discussion

Current literature on large-scale software development suggests that hybrid approaches of traditional plan-driven and agile software development may provide high value in very large, complex and long-running development projects [6]. While prior work has lined out the general conceptual differences between traditional and agile approaches [7], there is little knowledge about the details of their compatibility for hybrid settings. In particular, assertions that a combination appears to be promising [6], [55] but problematic [11], [29] have yet to culminate in explanations why hybrids succeed or fail.

Our study addressed this research gap and explored coordination challenges in a hybrid setting. In summary, the investigated case study lent a number of insights into inter-team coordination in a hybrid setting of traditional high-level planning and agile development. We found that coordination issues were primarily caused when there was no common awareness of dependencies across teams. This lack of dependency awareness in turn resulted from the misalignment of separate planning activities that were conducted on a team level and on an inter-team level respectively. In particular, we found that misalignments in specification and allocation inhibited dependency awareness, and misaligned prioritization and effort estimation activities further aggravated these issues. With this, we make the following contributions to research and practice.

5.1 Implications for Research

First and foremost, our study adds to prior work on hybrid development settings [6], [13], [14], [55] showing that dependency awareness constitutes a significant issue where top-down traditional planning needs to be brought together with bottom-up agile development. Our results thereby back and extend prior research on the crucial role of dependencies in software projects [25], [53], [82], [93]. While this stream of research has emphasized that coordination breakdowns frequently occur when developers are not capable to include the right colleagues in dependency resolution [25], we show that organizational structures and processes of hybrid approaches can even reinforce this incapacity. Through misaligned planning activities on team and inter-team levels, dependencies can remain concealed and consequently exert their known negative effects.

Prior work moreover suggests that creating hybrid development projects is possible but necessitates appropriate governance and control activities [13], [14]. The provided understanding of planning activity alignment and dependency awareness as a central antecedent of effective coordination hopefully allows future research as well as industry to tailor better practices and technologies that help increase dependency awareness also on an inter-team level. In fact, research has long and successfully engaged in providing tools that help identify and highlight dependencies between requirements, existing source code and newly developed items [83]–[85], [93]. Our study suggests that, particularly in large, hybrid traditional-agile development units, such visibility is necessary and planning activities need to be aligned even before fine-grained specifications are entirely detailed out. From a practical perspective, contemporary concepts such as 'delivery stories' described in

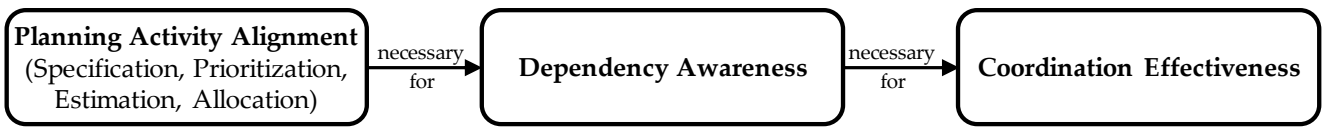


Fig. 10 Process Model Showing Relationships Between Planning Activity Alignment, Dependency Awareness & Coordination Effectiveness

software development outsourcing research [15] may potentially provide a helpful starting point to achieve this visibility. Future work on dependency visibility tools may take this into account and provide planning activity support that helps align inter-team level and team level planning already before and during specification.

Our findings not only re-emphasize the need for institutionalized coordination practices across single development teams, which agile methods do not incorporate today [53], [54], they also lend detailed insight into what these coordination practices must be able to accomplish: such coordination practices (i) must allow for involving higher level institutions such as central product teams, (ii) they must commence prior to single teams planning their next iterations in detail and (iii) they must bridge the differences in focus, detail and competence that distinguish team and inter-team level actors during activities of specification, prioritization, estimation and allocation. We thereby add to prior work that emphasized the importance of matching coordination reactions to evolving dependencies [24], [25] and line out temporal as well as organizational requirements of such coordination mechanisms.

While the collaborative and iterative work inherent to agile practices allows development teams to identify and resolve dependencies between their single tasks on the fly [23], traditional methods for planning and coordination applied on an inter-team level aim at preventing dependencies by nearly complete pre-specifications [94]. Although planning activities on both inter-team and team levels may be appropriate and well understood when viewed separately, our study suggests their alignment is crucial to discover and manage inter-team dependencies.

Furthermore, we identified the main points of differences and misalignment between the traditional and agile planning activities of our case. As described in section 4.2, these differences were seen along three main dimensions we define as: *focus, degree of detail* and *competence.* Since our case study was an exemplar of largely ineffective coordination, we were able to identify these specific dimensions of misalignment. As such, we do not claim these dimensions to be absolute or final, rather they can hopefully be extended and adapted through further research on the topic. Future studies can investigate how these dimensions can be aligned in practice by studying successful cases with a focus on planning alignment. Centralized units and processes that structurally enforce alignment [14] may constitute equally possible candidate mechanisms as informal and process-oriented ones [13].

5.2 Recommendations for Practice

Improving dependency awareness:

A direct practical implication of our findings is that regular inter-team level meetings may be required to improve dependency awareness and in turn facilitate inter-team coordination. For example, in addition to the Scrum-of-Scrums approach described in section 2.2, other collaborative team-level meetings such as sprint planning, reviews and retrospectives that capture agile principles of iterative delivery and collaboration can be mirrored at the inter-team level to facilitate planning alignment. The inter-team counterparts of these standard team-level agile meetings should include informed representatives from all teams concerned in addition to the members of the central team. This should establish a two-way feedback channel between the individual teams on the one hand and the central team on the other, as well as a platform for all teams to become aware of the existing and potential dependencies between them. What we are recommending differs from the SoS approach in that it is not limited to a single status-reporting meeting, but rather encompasses joint planning, reviews, and retrospective meetings. In general, the frequency and logistic needs of such meetings will likely be contingent on development unit size as well as on an organization's preferred approach to large-scale agile. As such, smaller development units may be able to conduct meetings with all team members present. For larger software development units and team sizes, representatives of each team may be required. Other, more traditional, meeting practices such as those conducted by senior management and market experts to create high-level product visions and roadmaps will likely still be relevant for companies operating in large-scale settings.

Aligning planning activities:

Our findings show that awareness issues can result from conflicting orientations toward content and time frames between inter-team and team levels. It may be possible to resolve such conflicts partially by preponing iterative planning activities on an inter-team level, propagating results to the team level and allowing for an iterative adjustment process between inter-team and team levels. As such, specification and prioritization on an inter-team level may take place with sufficient lead time before the start of the next sprint of the development teams. Development teams may then be able to specify high-priority tasks in detail and feed results regarding emergent inter-team dependencies back to the inter-team level for resolution before actually committing to the particular set of tasks for the upcoming sprint.

Planning tools development and evaluation:

Development of new tools that can be deployed in the planning stages to capture dependencies during activities such as specification, estimation, prioritization and allocation should be considered. Popular project management tools such as Rally's "ready > sync > go" [95] and VersionOne's enterprise agile platform [30] seem to acknowledge the varied approaches to scaling and aim to support different methodologies such as SAFe, LeSS, DAD and hybrid approaches. While their effectiveness remains to be studied, we suggest that companies that evaluate such tools for their purposes should particularly consider to which degree these tools are useful in achieving dependency awareness across development teams and help align specification, prioritization, estimation and allocation.

5.3 Limitations

This single case study served the purpose of building theory on a little understood phenomenon based on a specific revelatory case [70], [76]. We used empirical data to inductively create a process model explaining the sequence of events that if interrupted inhibits effective coordination in hybrid settings of traditional planning on an inter-team level and agile development on a team level. Importantly, this research design did not aim for generalization into all other settings of hybrid approaches which would rather follow a statistical replication logic [77], [96]. Instead, it aimed for creating a detailed and in-depth understanding of the events and mechanisms within this case to propose a powerful and fitted explanation through rigorous data collection and analysis [71], [76], [97]. In particular, while GT typically does not claim generalization to all contexts, the resulting theory or model should be adaptable to other contexts. What this means is we do not claim our model to be absolute or final. We want to encourage fellow researchers to pick up on this model and particularly the misalignment of planning activities to verify or challenge their explanatory value in related contexts. We welcome future studies to provide extensions to the model based on unseen aspects and refinements of the present categories and dimensions.

Although this specific case was revelatory regarding the misalignments of traditional high-level planning and agile development, we also acknowledge these elements as boundaries to our research. Instead of claiming generalizability to all hybrids of traditional and agile development, we hope to provide rich, valuable and detailed insights into settings where multiple agile teams need to collaboratively create a single software product based on longer release cycles and supported by more intense and structured high-level planning. Additionally, we only examined one single product, namely an established, highly integrated, ongoing and thus complex enterprise software system. It is very reasonable to assume that the architecture of this product influenced the technical interdependencies of single tasks and teams. Large software development units developing highly modular software may possibly face less negative effects from misaligned planning activities, simply because they also face less interdependencies between technical components altogether. Future research should therefore pay close attention to software architecture and its relation to dependency awareness.

6 Conclusion

In this study, we aimed to investigate how and why the combination of traditional planning on an inter-team level and agile development on a team level leads to ineffective coordination in large-scale software development. We found that a lack of dependency awareness is a major cause for ineffective inter-team coordination and that dependency awareness itself is inhibited by misaligned planning activities across team and inter-team levels, namely specification, prioritization, estimation and allocation. One avenue of future research could be the investigation into how to effectively prevent the described planning misalignments. Along these lines, an in-depth examination of cases seems promising where inter-team coordination works very well so as to delineate measures how to prevent or resolve dependencies.

References

[1] C. T. Schmidt, T. Kude, J. Tripp, A. Heinzl, and K. Spohrer, "Team Adaptability in Agile Information Systems Development," in *34th International Conference on Information Systems (ICIS)*, 2013, pp. 1–11.

[2] H. Sharp and H. Robinson, "Collaboration and co-ordination in mature eXtreme programming teams," *Int. J. Hum. Comput. Stud.*, vol. 66, no. 7, pp. 506–518, 2008.

[3] L. A. Williams and R. R. Kessler, "All I Really Need to Know About Pair Programming I Learned in Kindergarten," *Commun. ACM*, vol. 43, no. 5, pp. 108–114, 2000.

[4] R. Hoda, P. Kruchten, J. Noble, and S. Marshall, "Agility in Context," in *ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 2010, pp. 74–88.

[5] C. Larman and B. Vodde, *Practices for Scaling Lean & Agile Development Large, Multisite, and Offshore Product Development with Large-Scale Scrum*, 1st ed. Upper Saddle River, NJ: Addison Wesley, 2010.

[6] J. B. Barlow, J. S. Giboney, M. J. Keith, D. W. Wilson, and R. M. Schuetzler, "Overview and Guidance on Agile Development in Large Organizations," *Commun. Assoc. Inf. Syst.*, vol. 29, no. 2, pp. 25–44, 2011.

[7] B. Boehm and R. Turner, "Management Challenges to Implementing Agile Processes in Traditional Development Organizations," *IEEE Softw.*, vol. 22, no. 5, pp. 30–39, 2005.

[8] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *Computer (Long. Beach. Calif).*, vol. 36, no. 6, pp. 57–66, 2003.

[9] L. Cao, K. Mohan, P. Xu, and B. Ramesh, "How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects," in *37th Hawaii International Conference on System Sciences (HICSS)*, 2004.

[10] L. Cao, K. Mohan, P. Xu, and B. Ramesh, "A framework for adapting agile development methodologies," *Eur. J. Inf. Syst.*, vol. 18, no. 4, pp. 332–343, 2009.

[11] D. Badampudi, B. Fricker, and A. M. Moreno, "Perspectives on Productivity and Delays in Large-Scale Agile Projects," in *14th International Conference on XP, Agile Processes in Software Engineering and Extreme Programming*, 2013, pp. 180–194.

[12] S. W. Ambler and M. Lines, *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. Pearson Education, 2012.

[13] L. Mahadevan, W. J. Ketinger, and T. O. Meservy, "Running on Hybrid: Control Changes when Introducing an Agile Methodology in a Traditional 'Waterfall' System Development Environment," *Commun. Assoc. Inf. Syst.*, vol. 36, no. 1, pp. 77–103, 2015.

[14] N. Ramasubbu, A. Bharadwaj, and G. K. Tayi, "Software process diversity: conceptualization, measurement, and

analysis of impact on project performance," *MIS Q.*, vol. 39, no. 4, pp. 787–807, 2015.

[15] M. Daneva, E. van der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, and R. Wieringa, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1333–1353, 2013.

[16] A. Scheerer and T. Kude, "Exploring Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective," in *35th International Conference on Information Systems (ICIS)*, 2014, pp. 1–11.

[17] T. Dingsøyr and N. B. Moe, "Research Challenges in Large-Scale Agile Software Development," in *ACM SIGSOFT Software Engineering Notes*, 2013, vol. 38, no. 5, pp. 38–39.

[18] H. D. Benington, "Production of large computer programs," *Ann. Hist. Comput.*, vol. 5, no. 4, pp. 350–361, 1983.

[19] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," in *9th International Conference on Software Engineering (ICSE)*, 1987, pp. 328–338.

[20] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas, and A. Development, "Manifesto for Agile Software Development," *Agile Alliance*, 2001. [Online]. Available: http://www.agilemanifesto.org/. [Accessed: 24-Aug-2014].

[21] N. B. Moe, T. Dingsøyr, and T. Dybå, "A teamwork model for understanding an agile team: A case study of a Scrum project," *Inf. Softw. Technol.*, vol. 52, no. 5, pp. 480–491, 2010.

[22] R. Hoda and L. K. Murugesan, "Multi-Level Agile Project Management Challenges: A Self-Organizing Team Perspective," *J. Syst. Softw.*, vol. 117, no. April, pp. 245–257, 2016.

[23] K. Conboy, "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Inf. Syst. Res.*, vol. 20, no. 3, pp. 329–354, 2009.

[24] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity," in *2nd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2008, pp. 2–11.

[25] M. Cataldo and J. D. Herbsleb, "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures," *IEEE Trans. Softw. Eng.*, vol. 39, no. 3, pp. 343–360, Mar. 2013.

[26] J. C.-L. Goh, S. L. Pan, and M. Zuo, "Developing the Agile IS Development Practices in Large-Scale IT Projects: The Trust-Mediated Organizational Controls and IT Project Team Capabilities Perspectives," *J. Assoc. Inf. Syst.*, vol. 14, no. 12, pp. 722–756, 2012.

[27] T. Dingsøyr and N. B. Moe, "Towards Principles of Large-Scale Agile Development," in *XP 2014 International Workshop on Agile Methods, Large-Scale Development, Refactoring, Testing, and Estimation*, 2014, pp. 1–8.

[28] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation," *Empir. Softw. Eng.*, pp. 1–31, 2017.

[29] D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE Softw.*, vol. 20, no. 4, pp. 12–15, 2003.

[30] VersionOne, "VersionOne Enterprise Agile Platform," *State of Agile Survey - Version One*, 2015. [Online]. Available: http://info.versionone.com/state-of-agile-development-survey-ninth.html.

[31] P. Deemer, G. Benefield, C. Larman, and B. Vodde, "The Scrum Primer Version 2.0," 2012. [Online]. Available: http://www.scrumprimer.com/. [Accessed: 18-Aug-2015].

[32] K. Schwaber and J. Sutherland, "The Scrum Guide." pp. 1–16, 2011.

[33] T. Kude, S. Bick, C. T. Schmidt, and A. Heinzl, "Adaptation Patterns in Agile Information Systems Development Teams," in *22nd European Conference on Information Systems (ECIS)*, 2014, pp. 1–15.

[34] "Definition of Scrum of Scrums," *Agile Alliance Scrum Guide*, 2016. [Online]. Available: http://guide.agilealliance.org/guide/scrumofscrums.html. [Accessed: 14-Mar-2016].

[35] C. Larman and B. Vodde, *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*, 1st ed. Addison-Wesley Professional, 2008.

[36] J. Sutherland, "Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies," *Cut. IT J.*, vol. 14, no. 12, pp. 5–11, 2001.

[37] M. Paasivaara and C. Lassenius, "Communities of practice in a large distributed agile software development organization - Case Ericsson," *Inf. Softw. Technol.*, vol. 56, no. 12, pp. 1556–1577, 2014.

[38] G. C. Kane and M. Alavi, "Information Technology and Organizational Learning: An Investigation of Exploration and Exploitation Processes," *Organ. Sci.*, vol. 18, no. 5, pp. 796–812, 2007.

[39] M. Alavi and D. E. Leidner, "Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues," *MIS Q.*, vol. 25, no. 1, pp. 107–136, 2001.

[40] B. Vodde and C. Larman, "LeSS Framework," 2016. [Online]. Available: http://less.works/. [Accessed: 24-Aug-2015].

[41] D. Leffingwell, "Scaled Agile Framework (SAFe)," 2014. [Online]. Available: http://www.scaledagileframework.com/. [Accessed: 03-Jun-2014].

[42] K. Schwaber, "Scaled Professional Scrum | Nexus Framework," 2016. [Online]. Available: https://www.scrum.org/Resources/What-is-Scaled-Scrum? [Accessed: 24-Aug-2015].

[43] K. Conboy and B. Fitzgerald, "Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 1, pp. 1–30, 2010.

[44] B. Fitzgerald, G. Hartnett, and K. Conboy, "Customising agile methods to software practices at Intel Shannon," *Eur. J. Inf. Syst.*, vol. 15, no. 2, pp. 197–210, 2006.

[45] G. Theocharis, M. Kuhrmann, J. Münch, and P. Diebold, "Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices," in *16th International Conference on Product-Focused Software Process Improvement (PROFES 2015)*, 2015, pp. 149–166.

[46] V. T. Heikkila, M. Paasivaara, and C. Lassenius, "ScrumBut,

But Does it Matter? A Mixed-Method Study of the Planning Process of a Multi-team Scrum Organization," *2013 ACM / IEEE Int. Symp. Empir. Softw. Eng. Meas.*, pp. 85–94, 2013.

[47] P. Kettunen and M. Laanti, "Combining Agile Software Projects and Large-scale Organizational Agility," *Softw. Process Improv. Pract.*, vol. 13, no. 2, pp. 183–193, 2008.

[48] C. Fry and S. Greene, "Large Scale Agile Transformation in an On-Demand World.," in *Agile 2007 Conference*, 2007, pp. 136–142.

[49] V. Heikkilä, K. Rautiainen, and S. Jansen, "A revelatory case study on scaling agile release planning," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2010, pp. 289–296.

[50] L. Lagerberg, T. Skude, P. Emanuelsson, K. Sandahl, and D. Ståhl, "The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at Ericsson," in *IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 348–356.

[51] A. Begel and N. Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," in *1st International Symposium on Empirical Software Engineering and Metrics*, 2007, no. 9, p. 10.

[52] B. Fitzgerald, K.-J. Stol, R. O'Sullivan, and D. O'Brien, "Scaling Agile Methods to Regulated Environments: An Industry Case Study," in *35th International Conference on Software Engineering (ICSE)*, 2013, pp. 863–872.

[53] J. Vlietland and H. van Vliet, "Towards a governance framework for chains of Scrum teams," *Inf. Softw. Technol.*, vol. 57, pp. 52–65, 2015.

[54] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Empir. Softw. Eng.*, vol. 13, no. 3, pp. 303–337, 2008.

[55] D. Batra, W. Xia, D. VanderMeer, and K. Dutta, "Balancing Agile and Structured Development Approaches to Successfully Manage Large Distributed Software Projects: A Case Study from the Cruise Line Industry," *Commun. Assoc. Inf. Syst.*, vol. 27, no. 1, pp. 379–394, 2010.

[56] oxforddictionaries.com, "hybrid," 2017. [Online]. Available: https://en.oxforddictionaries.com/definition/hybrid.

[57] J. E. Mathieu, M. A. Marks, and S. J. Zaccaro, "Multiteam Systems," in *Handbook of Industrial, Work and Organizational Psychology*, vol. 2, N. Anderson, D. S. Ones, H. K. Sinangil, and C. Viswesvaran, Eds. London, United Kingdom: Routledge, 2001, pp. 289–313.

[58] J. D. Thompson, *Organizations in Action: Social Science Bases of Administrative Theory*, 48th ed. New York, NY: McGraw-Hill, 1967.

[59] K. Crowston, "A Taxonomy of Organizational Dependencies and Coordination Mechanisms," in *Organizing Business Knowledge: The MIT Process Handbook*, T. W. Malone, K. Crowston, and G. Herman, Eds. Cambridge, MA, USA: MIT Press, 2003, pp. 85–108.

[60] D. E. Strode and S. L. Huff, "A taxonomy of dependencies in agile software development," in *23rd Australasian Conference on Information Systems (ACIS)*, 2012, pp. 1–10.

[61] J. Howison and K. Crowston, "Collaboration Through Open Superposition," *MIS Q.*, vol. 38, no. 1, pp. 29–50, 2014.

[62] K. Crowston, J. Rubleske, and J. Howison, "Coordination theory and its application in HCI," Paper 485, 2006.

[63] J. A. Espinosa, F. Armour, and W. F. Boh, "Coordination in Enterprise Architecting: An Interview Study," in *43rd Hawaii International Conference on System Sciences (HICSS)*, 2010, pp. 1–10.

[64] H. Mintzberg, "Structure in 5's: A Synthesis of the Research on Organization Design," *Manage. Sci.*, vol. 26, no. 3, pp. 322–341, 1980.

[65] D. E. Strode, B. Hope, S. L. Huff, and S. Link, "Coordination Effectiveness In An Agile Software Development Context.," in *Pacific Asia Conference on Information Systems (PACIS)*, 2011.

[66] Y. Li and A. Mädche, "Formulating Effective Coordination Strategies in Agile Global Software Development Teams," in *33rd International Conference on Information Systems (ICIS)*, 2012, pp. 1–12.

[67] D. E. Strode, S. L. Huff, B. Hope, and S. Link, "Coordination in co-located agile software development projects," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1222–1238, 2012.

[68] M. Yuan, X. Zhang, Z. Chen, D. R. Vogel, X. Chu, Y. Minghui, Z. Xi, C. Zhenjiao, D. R. Vogel, and C. Xuelin, "Antecedents of Coordination Effectiveness of Software Developer Dyads From Interacting Teams: An Empirical Investigation.," *IEEE Trans. Eng. Manag.*, vol. 56, no. 3, pp. 494–507, 2009.

[69] G. Lee, J. A. Espinosa, and W. H. DeLone, "Task Environment Complexity, Global Team Dispersion, Process Capabilities, and Coordination in Software Development," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1753–1771, 2013.

[70] K. M. Eisenhardt, "Building Theories from Case Study Research," *Acad. Manag. Rev.*, vol. 14, no. 4, pp. 532–550, Oct. 1989.

[71] B. G. Glaser, *Emergence vs. Forcing: Basics of Grounded Theory Analysis*. Mill Valley: Sociology Press, 1992.

[72] C. Soh and M. L. Markus, "How IT Creates Business Value: A Process Theory Synthesis," in *16th International Conference on Information Systems (ICIS)*, 1995, pp. 29–41.

[73] D. Robey, M. L. Markus, and D. Robey, "Information Technology and Organizational Change: Causal Structure in Theory and Research," *Manage. Sci.*, vol. 34, no. 5, pp. 583–598, 1988.

[74] P. Ralph, "Software engineering process theory: A multi-method comparison of Sensemaking-Coevolution-Implementation Theory and function-behavior-structure theory," *Inf. Softw. Technol.*, vol. 70, pp. 232–250, 2016.

[75] R. Hoda and J. Noble, "Becoming Agile: A Grounded Theory of Agile Transitions in Practice," in *International Conference on Software Engineering (ICSE)*, 2017.

[76] R. K. Yin, *Case Study Research: Design and Methods*, 4th ed. SAGE Publications Inc., 2009.

[77] A. S. Lee, "A Scientific Methodology for MIS Case Studies," *MIS Q.*, vol. 13, no. 1, pp. 33–50, 1989.

[78] K. M. Eisenhardt and M. E. Graebner, "Theory Building from Cases: Opportunities and Challenges," *Acad. Manag. J.*, vol. 50, no. 1, pp. 25–32, 2007.

[79] K.-J. Stol, P. Avgeriou, M. A. Babar, Y. Lucas, and B. Fitzgerald, "Key Factors for Adopting Inner Source," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 2, pp. 1–35, 2014.

[80] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing Agile teams," *Empir. Softw. Eng.*, vol. 17, no. 6, pp. 609–639, 2012.

[81] B. G. Glaser, *Theoretical sensitivity: Advances in the methodology of grounded theory*. Sociology Press, 1978.

[82] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," in *ACM Computer Supported Cooperative Work (CSCW)*, 2006.

[83] A. Borici, K. Blincoe, A. Schröter, G. Valetto, and D. Damian, "ProxiScientia: Toward Real-Time Visualization of Task and Developer Dependencies in Collaborating Software Development Teams," in *ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2012, pp. 5–11.

[84] C. R. B. de Souza, T. Hildenbrand, and D. Redmiles, "Toward Visualization and Analysis of Traceability Relationships in Distributed and Offshore Software Development Projects," in *1st International Conference Software Engineering Approaches for Offshore and Outsourced Development (SEAFOOD)*, 2007, pp. 182–199.

[85] T. Hildenbrand, *Improving Traceability in Distributed Collaborative Software Development: A Design Science Approach*, 1st ed. Frankfurt, Germany: Peter Lang Frankfurt, 2008.

[86] P. Dourish and V. Bellotti, "Awareness and Coordination in Shared Workspaces," in *ACM Conference on Computer-Supported Cooperative Work (CSCW)*, 1992, no. November, pp. 107–114.

[87] E. Whitworth and R. Biddle, "The Social Nature of Agile Teams," in *Agile 2007 Conference*, 2007, pp. 26–36.

[88] E. Litwak and L. F. Hylton, "Interorganizational Analysis: A Hypothesis on Co-ordinating Agencies," *Adm. Sci. Q.*, vol. 6, no. 4, pp. 395–420, 1962.

[89] A. H. Van de Ven, A. L. Delbecq, and R. Koenig, "Determinants of Coordination Modes within Organizations," *Am. Sociol. Rev.*, vol. 41, no. 2, pp. 322–338, 1976.

[90] J. R. Galbraith, "Organization Design: An Information Processing View," *Interfaces (Providence).*, vol. 4, no. 3, pp. 28–36, 1974.

[91] K. Srikanth and P. Puranam, "Integrating distributed work: comparing task design, communication, and tacit coordination mechanisms," *Strateg. Manag. J.*, vol. 32, no. 8, pp. 849–875, 2011.

[92] A. Lindberg, N. Berente, J. Gaskin, and K. Lyytinen, "Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project," *Inf. Syst. Res.*, vol. 27, no. 4, pp. 751–772, 2016.

[93] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software Dependencies, Work Dependencies, and Their Impact on Failures," *IEEE Trans. Softw. Eng.*, vol. 35, no. November, pp. 864–878, 2009.

[94] R. W. Zmud, "Management of Large Software Development Efforts," *MIS Q.*, vol. 4, no. 2, pp. 45–55, 1980.

[95] "Rally Ready > Sync > Go." [Online]. Available: https://www.rallydev.com/. [Accessed: 24-Aug-2015].

[96] A. S. Lee and R. L. Baskerville, "Generalizing Generalizability in Information Systems Research," *Inf. Syst. Res.*, vol. 14, no. 3, pp. 221–243, 2003.

[97] J. P. Davis, K. M. Eisenhardt, and C. B. Bingham, "Developing Theory Through Simulation Methods.," *Acad. Manag. Rev.*, vol. 32, no. 2, pp. 480–499, Apr. 2007.



Saskia Bick received the B.A. degree in International Management from the University of Applied Sciences FH Joanneum in Graz, Austria, in 2010, the M.Sc. degree in Management from the University of Mannheim, Germany, in 2013, where she also received her PhD degree in Information Systems. She is currently working as a Program Lead and Agile Coach at SAP SE and is a certified Scrum Master. Her research interests include agile software development, inter-team coordination and dependency management.



Kai Spohrer is an Assistant Professor in Information Systems at the University of Mannheim, Germany. He received his PhD in Information Systems from the University of Mannheim in 2015 for his dissertation on the effects of collaborative quality assurance on team cognition in software development teams. His research has appeared at major international conferences and he received a best paper award from the International Conference on Intelligent Environments. His research interests include coordination and team cognition in software development and IT healthcare. He is an active member of the AIS.



Rashina Hoda is a Senior Lecturer in Software Engineering in the Department of Electrical and Computer Engineering, The University of Auckland, New Zealand where she leads the SEPTA research group. She received a bachelor's degree in Computer Science from LSU (USA), her PhD from Victoria University of Wellington, New Zealand, and is a certified Scrum Master. Her papers appear in the IEEE Transactions on Software Engineering, Information and Software Technology, Empirical Software Engineering, Journal of Systems and Software, IEEE International Conference on Software Engineering, and others. Her research interests include human aspects of software engineering, in particular, agile software development, and HCI. She received a distinguished paper award at ICSE2017, and has served as the Research Chair of Agile India 2012 and as Associate Editor for ICIS (2015-2016). She is an active member of the IEEE.



Alexander Scheerer received the diploma (M.Sc. equivalent) in Information Systems from the University of Mannheim, Germany, in 2011 where he also received his PhD degree in Information Systems. His research has appeared at major international conferences including International Conference on Information Systems. He is working as an Agile Coach and Scrum Master at SAP SE. His research interests include large-scale agile software development and inter-team coordination.



Armin Heinzl is a Professor and Chair in General Management and Information Systems at the University of Mannheim, Germany. He received his doctoral degree in 1990 and his habilitation in 1995 from the Koblenz Corporate School of Management (WHU). He was a full professor at the University of Bayreuth and has held visiting positions at Harvard, Berkeley, Irvine, ESSEC and LSE. His papers have appeared in MIS Quarterly, the Journal of the Association of Information Systems, IEEE Transactions on Engineering Management, Journal of Business Economics, Journal of Organizational Computing and E-Commerce, Evolutionary Computation and others. His research and teaching interests include IT outsourcing and governance, software development processes as well as human information behavior. He is Vice Editor in Chief of the journal Business & Information Systems Engineering and an active member of the ACM and the AIS.