AUTONOMOUS, YET INTERDEPENDENT: DESIGNING INTERFACES ACROSS ROUTINE CLUSTERS

Christian A. Mahringer

Heidelberg Academy of Sciences and Humanities &

University of Stuttgart School of Management

Keplerstr. 17, 70174 Stuttgart, Germany

Christian.mahringer@bwi.uni-stuttgart.de

Anja Danner-Schröder

RPTU Kaiserslautern

Gottlieb-Daimler-Str. 42, 67663 Kaiserslautern, Germany

anja.dannerschroeder@rptu.de

Manuscript accepted at the Academy of Management Journal

AUTONOMOUS, YET INTERDEPENDENT: DESIGNING INTERFACES ACROSS ROUTINE CLUSTERS

ABSTRACT

This paper examines the process of designing interfaces between routine clusters in a world in flux in which interdependencies are emergent. We reveal that designing interfaces is an endogenous and iterative process of creating and harmonizing interdependencies. In our ethnographic study of the reorganization of agile software development, actors implemented a second routine cluster, which they envisioned to work autonomously. Because the teams shared resources, however, they created interdependencies across the two routine clusters that challenged their autonomy. Our findings contribute to research on routine dynamics, interdependence, and organizational design, first, by showing how the dynamics of designing interfaces are driven by the emergent nature of interdependencies, which routine participants may endogenously harmonize through routine performances. Second, our findings detail the practices through which ‘interface design work’ is accomplished. Although some of these practices imply transitory variations in routine performances, others change the patterning of interfaces more sustainably. Third, resources play an important role in designing interfaces because making them readily available may require their reconfiguration. Moreover, the pooled interdependencies that flow from shared resources can escalate into more complex types of interdependence.

Keywords: Agile, architectural knowledge, autonomy, ethnography, interdependence, modularity, organizational design, organizational routines, patterning, practice theory, process, routine dynamics, Scrum, self-organization.

INTRODUCTION

In dynamic environments, organizations frequently reorganize their work into semi-autonomous work units. Prominent examples are project-based organizations (Hobday, 2000; Lundin & Söderholm, 1995; Sydow, Lindkvist, & DeFillippi, 2004) or self-organizing teams (Cohen & Ledford Jr, 1994; Langfred, 2007; Lee & Edmondson, 2017), in which each work unit is envisioned to act autonomously. When organizations implement autonomous work units, however, they are confronted with various interdependencies that are characteristic of contemporary work (Raveendran, Silvestri, & Gulati, 2020). Consider, for instance, that actors often participate in multiple projects simultaneously, which makes those projects interdependent (Kremser & Blagoev, 2021), or that complex digital artifacts are often used by multiple teams (D'Adderio & Pollock, 2020). This raises the question how actors can disentangle work into autonomous work units against the backdrop of the interdependencies that characterize work in organizations.

Shedding light on this question inevitably compels us to focus on *interfaces* (Simon, 1996), defined as the "contact point[s] between relatively autonomous organizations which are nevertheless interdependent and interacting as they seek to cooperate to achieve some larger system objective" (Wren, 1967: 71). Organizational work is usually differentiated into various routine clusters (Burns & Stalker, 1961; Dougherty, 2001; Lawrence & Lorsch, 1967), each of which consists of "multiple, complementary routines, each contributing a partial result to the accomplishment of a common task" (Kremser & Schreyögg, 2016: 698). Because those clusters must be integrated in a way that contributes to the overall task of the organization, routine clusters "are enmeshed in far-reaching, complex, tangled webs of interdependence" (Feldman & Pentland, 2003: 104; see also Hoekzema, 2020; Victor & Blackburn, 1987). These interdependencies can be addressed through interfaces, which define how those interdependencies should be managed (Kremser & Schreyögg, 2016).

Prior research has often focused on the formal, programmed characteristics of interfaces (Adler, 1995; Kremser & Schreyögg, 2016; Puranam, Raveendran, & Knudsen, 2012). This research, for instance, has examined which coordination mechanisms interfaces must include to manage different types of interdependence (Thompson, 1967; Van de Ven, Delbecq, & Koenig Jr, 1976). It is also assumed that designers play an essential role, as they possess knowledge about organizational tasks and interdependencies, enabling them to exogenously design interfaces apart from routine performances (Brusoni, Henkel, Jacobides, Karim, MacCormack, Puranam et al., 2023; Kremser & Schreyögg, 2016; Puranam & Raveendran, 2013). Although these insights align with relatively stable and well-defined environments, they are somehow at odds with the flux that is characteristic of contemporary organizations (Birnholtz, Cohen, & Hoch, 2007; Mahringer, Pentland, Renzl, Sele, & Spee, 2024). A challenge for organizing in flux is that interdependencies between routine clusters are often not obvious (Mayo, 2022)—"[i]nterdependence is like air; it surrounds everything in an organization, but it is difficult to see" (Pentland, Recker, & Wyner, 2016: 1). This is because interdependencies often emerge in action (Deken, Carlile, Berends, & Lauche, 2016), and thus they prevent "formal organization design before work is performed" (Raveendran et al., 2020: 845).

We argue that taking the emergent nature of interdependencies into account requires us to focus on the process of designing interfaces between routine clusters. A practice and process ontology (Cloutier & Langley, 2020; Feldman & Orlikowski, 2011; Langley & Tsoukas, 2012) helps us to understand the process of designing interfaces because it focuses on the becoming (Tsoukas & Chia, 2002) of interfaces. This way, we can generate a more comprehensive picture of the actual work entailed in designing interfaces. Therefore, we ask two pressing questions: First, how do interdependencies emerge during the reorganization of routine clusters? And second, how does the process of designing interfaces to manage those interdependencies unfold?

In addressing these questions, we draw on a twelve-month ethnographic study of the reorganization of agile software development teams in a medium-sized company. A team in the organization had developed a routine cluster based on the Scrum framework (Schwaber & Sutherland, 2020). When the organization wanted to create a new software product, management decided that this new software product should be developed by a new team, instead of the old team developing both software products. As is common in agile approaches (Baham & Hirschheim, 2022), both teams planned to perform their routine clusters as autonomously as possible. Due to resource constraints and expected synergies, the teams started to share resources (i.e., actors and software code). However, reorganizing the clusters in this way was more challenging than assumed a priori, because various interdependencies emerged across both clusters in the process of designing interfaces. Our study sheds light on the process through which the teams iteratively created and harmonized interdependencies until they had designed interfaces that allowed both routine clusters to work relatively autonomously.

Our insights make three contributions to research on routine dynamics, interdependence, and organizational design. First, our findings reveal that designing interfaces is an endogenous process of creating and harmonizing interdependencies. The process of designing interfaces is driven in part because (a) new concerns emerge as people reorganize work into autonomous routine clusters, and (b) because efforts to harmonize interdependencies can proliferate into new, unexpected and potentially more complex interdependencies later. Due to the emergent nature of these interdependencies, designing interfaces may happen through routine participants as endogenous designers. Second, we find nuances in the practices that actors employ in designing interfaces between routine clusters, emphasizing the relevance of what we call ‘interface design work’. We identify four practices: analyzing interdependencies, reconfiguring resources, mitigating interferences, and reconfiguring schedules. We theorize that these practices can be differentiated according to two characteristics: durability of interface design work (whether practices represent transitory or patterning work) as well as design elements (whether practices privilege artifacts or timing). Related to the durability dimension, the transitory practices are often the means through which patterning practices come to be defined, while the patterning practices sometimes require transitory practices in order for patterning to become stabilized. Third, the design elements dimension nuances how resources play a prominent role in designing interfaces, as they are not always readily available. Thus, they may either need to be reconfigured when they are divisible, or they need to be scheduled in time when they are indivisible. Pooled interdependencies that flow from sharing resources can lead to more complex types of interdependence (i.e., sequential and reciprocal), ironically complicating progress toward autonomy. In other words, disentangling routine clusters may initially result in heightened forms of entanglement.

THEORETICAL BACKGROUND

Interdependence is a core issue in management and organization studies (Puranam & Raveendran, 2013; Puranam et al., 2012; Victor & Blackburn, 1987), and more particularly in research on the dynamics of routines (Deken et al., 2016; Hoekzema, 2020; Kremser, Pentland, & Brunswicker, 2019; Sailer, Loscher, & Kaiser, 2023; Spee, Jarzabkowski, & Smets, 2016; Turner & Rindova, 2012). In routines research, "[t]wo (or more) routines can be said to be interdependent to the extent that performing one routine creates an enabling and/or constraining context for performing the other(s)" (Rosa, Kremser, & Bulgacov, 2021: 244). Multiple interdependent routines cohere to form a so-called routine cluster. Routine clusters result from the division of labor, through which each routine cluster is formed, for instance, around objects (e.g., products) or activities (e.g., accounting, marketing) (Hoekzema, 2020; Lawrence & Lorsch, 1967; Victor & Blackburn, 1987). Hence, organizations consist of multiple, differentiated routine clusters that must be integrated in a way that contributes to the overall task of the organization. As these differentiated routine clusters need to form a coherent well-functioning system, they are interdependent by definition (Adler, 1995; McCann & Ferry, 1979; Victor & Blackburn, 1987).

Managing these interdependencies across routine clusters, however, is a challenging endeavor. This is even more the case when managers envision routine clusters as functioning autonomously (Balogun & Johnson, 2004), which means that routine participants are "able to act freely, without being obstructed by external interference" (Wiedner & Mantere, 2019: 661), and they have "discretion over how to perform organizational [routines]" (Wiedner & Mantere, 2019: 662; see also Arregle, Dattée, Hitt, & Bergh, 2023). The significance of such autonomy is evident, for instance, in self-organizing teams (Cohen & Ledford Jr, 1994; Langfred, 2007; Lee & Edmondson, 2017) or project-based organizations (Hobday, 2000; Lundin & Söderholm, 1995; Sydow et al., 2004). Routine participants not only require the discretion to make autonomous decisions, but they must also be able to control the resources that enable them to carry out their work (Arregle et al., 2023; Oliver, 1991; Wiedner & Mantere, 2019). As Garnier (1982: 894) says, autonomy "allows the [... routine cluster] to mobilize its resources to solve various problems in the best possible way and thus to reach the objectives it has set for itself." Thus, the appropriate allocation of resources is an essential part of the management of interdependencies across routine clusters (McCann & Ferry, 1979; Salancik & Pfeffer, 1978).

We draw on Thompson's (1967) influential distinction between pooled, sequential, and reciprocal interdependence to reveal nuances in interdependencies across routine clusters. Pooled interdependence means that two routine clusters contribute to or share the same pool of (scarce) resources, sequential interdependence means that the outputs of one routine cluster become the inputs for the other routine cluster, and reciprocal interdependence refers to mutual effects between two routine clusters. These three types of interdependence build on each other, as sequential interdependencies also include pooled ones, and reciprocal interdependencies include the other two types (Thompson, 1967). Thus, they represent increasingly complex types of interdependence. Scholars argue that interdependencies between routine clusters should be minimized (Thompson, 1967), as this facilitates the management of interfaces.

Interfaces are the contact points between routine clusters (Wren, 1967), which make “the results of each routine expectable for the actors that perform connected routines” (Kremser & Schreyögg, 2016: 698). These expectations can be more or less formal depending on the interdependence at hand. Simpler types of interdependence can be coordinated through plans, rules, or programs, whereas more complex interdependencies require unscheduled meetings, horizontal communication channels, and improvisation (Hoekzema, 2020; Van de Ven et al., 1976; Victor & Blackburn, 1987). However, as recent research has evidenced, formal and informal expectations do not imply that routines are performed in standardized or mindless ways, but they can be performed dynamically (Geiger, Danner-Schröder, & Kremser, 2021; Spee et al., 2016; Yamauchi & Hiramoto, 2020). Thus, the performances of interfaces are effortful accomplishments (Pentland & Rueter, 1994).

The importance of (new) interfaces draws attention to their design, particularly when organizations are being reorganized. Interface design refers to “intentional efforts to change one or more aspects of a routine [interface] to create a preferred situation” (Wegener & Glaser, 2021: 301). Research suggests that interface design usually happens apart from routine performances—“the designing of interfaces [...] happens ‘outside’ the routine” (Kremser et al., 2019: 95; see also Glaser, 2017; Gupta, Hoopes, & Knott, 2015). Therefore, it is important that exogenous designers can grasp the interdependencies that they have to consider as they design interfaces. Consequently, designers require architectural “knowledge about the components of a complex system and how they are related” (Baldwin, 2010: 3; see also Baldwin & Clark, 2000; Brusoni et al., 2023). This implies that interdependencies are well-known, comprehensible, stable, and unambiguous.

These assumptions imply a stable and well-ordered organizational setting, such as Smith's (1776) classic example of pin production (Puranam et al., 2012). However, contemporary organizations are in flux (Birnholtz et al., 2007; Mahringer et al., 2024). Thus, interdependencies are not exogenously given, but more emergent than usually acknowledged (Deken et al., 2016; Kremser et al., 2019; Mayo, 2022; Sele & Grand, 2016). The emergent nature of interdependencies also has implications for designing interfaces, because "it is now virtually impossible to design an entire organization before agents actually conduct the work; indeed, novel elements [e.g., interdependencies] arise constantly while work is being performed" (Raveendran et al., 2020: 828-829). Therefore, we need to focus on the "actions specifically associated with designing routine interfaces" (Wegener & Glaser, 2021: 309). We adopt a practice and process ontology (Cloutier & Langley, 2020; Feldman & Orlikowski, 2011; Langley & Tsoukas, 2012) to understand the consequentiality of emergent interdependencies for the process of designing interfaces during the process of routine cluster reorganization.

METHOD

Context

The empirical context of this study is 'Technology Innovation Corporation' (TIC) (a pseudonym). TIC is a medium-sized, high-tech manufacturing company, which develops complex machines. These machines, for instance, cut cellphone displays or laser-weld battery packs. When TIC lost the order of a major customer, the company faced serious financial struggles. They felt that one problem that contributed to this failure was their inability to quickly address the requirements of their customers. A manager noted, "we want a dynamic product development process where we can quickly develop new products to optimally serve our very demanding customers. Our customers want us to deliver results in short intervals" (Interview, manager, 22.11.2016). Another manager said, “we need everything very quickly, preferably yesterday. Embrace the change!” (Interview, manager, 20.02.2017).

Management tackled this problem by piloting the Scrum framework, which promises agility through autonomous self-organization (Dönmez, Grote, & Brusoni, 2016; Kremser & Xiao, 2021; Ritter, Danner-Schröder, & Müller-Seitz, 2024; Sailer et al., 2023; Schwaber & Sutherland, 2020). Whereas traditional project management approaches attempt to plan the entire project from start to finish, Scrum splits the development process into iterative phases of (in our case) two weeks. Each of these phases is called a ‘Sprint’. In each Sprint, team members solve a set of so-called ‘issues’. Each issue represents a software feature that satisfies a specific customer requirement or a bug in the software that needs to be resolved. Scrum also defines three roles: the Product Owner, who is responsible for maximizing the value of the work output; the Scrum Master, who ensures that a team adheres to the Scrum rules; and the developers, who resolve issues, usually by changing software code.

Team Alpha was the first team to implement Scrum at TIC in 2013. Team Alpha was a software development team (consisting of six to seven members), which provided solutions that could be used by various business units. Team Alpha developed the software product ‘AlphaSoft’, which was used to operate the machines that TIC developed (i.e., the human-machine interface). Over the years, Team Alpha had developed a cluster of Scrum routines that helped organize software development.

In November 2016, TIC’s management decided to set up a new team, Team Beta, which consisted of four to five members. The aim of Team Beta was to develop a software tool called ‘BetaSoft’, which was used to visually assess the quality of microchips. Initially, TIC’s management discussed whether BetaSoft should be a part of AlphaSoft, but then opted for a separate software tool developed by a separate team. The overall idea was that both teams could autonomously develop their software products: “we thought about whether Beta and Alpha should be different things [...] an advantage is that both products can be released independently” (Interview, Alpha developer, 08.02.2017).

When Team Beta was implemented, it was determined that they should also use the Scrum framework. Therefore, the team replicated the routines that Team Alpha had developed. As Brandon, the Product Owner of Team Beta, noted early in the process, “we just adopted [… the routines from Team Alpha], because the Scrum Master recommended it. In our first Sprint this worked out well, we did not see something negative” (Interview, Brandon, 29.11.2016). A developer also explained that “in general we do it [Scrum] like Team Alpha. They have been using it for years and it works well, so we don’t have to reinvent the wheel” (Interview, Beta developer, 22.03.2017). Even though this reorganization of work into two autonomous routine clusters seemed beneficial at first sight, it turned out to be more challenging than assumed in the beginning. Our study aims at exploring those challenges and how the teams dealt with them.

INSERT FIGURE 1 HERE

Data Collection

The first author collected the data between November 2016 and October 2017. He used four methods of data collection (see Figure 1 for an overview): observation, interviews, collection of documents, and collection of digital trace data.

Observations. The first author spent two to three days per week in the field over a period of twelve months observing software development at TIC. The primary focus of the observations was the performances of the Scrum routines of teams Alpha and Beta as the Beta cluster was spun off from the Alpha cluster, complemented by workshops, community events, and daily interactions such as the after-lunch walk, team excursions, and Christmas parties. He frequently jotted down the doings and sayings of actors and transcribed them into detailed field notes within 24 hours (Emerson, Fretz, & Shaw, 2011). Audio tapes of many meetings complemented the observations.

Interviews. The first author conducted thirty formal interviews during the fieldwork. Moreover, five further interviews were conducted two years after the fieldwork phase to update knowledge on how Scrum at TIC evolved over time. In total, he conducted at least one interview with each member of teams Alpha and Beta, and additionally with members of other teams that considered using agile methods. Table 1 provides an overview of the interviews, including the pseudonyms used in the findings section. The table indicates that interviews with the members of teams Alpha and Beta are immediately relevant to this study, while the other interviews relate to the broader context. Appendix A provides an exemplary interview protocol.

INSERT TABLE 1 HERE

Documents. The first author also collected approximately 6300 pages of documents, such as handbooks, screenshots, and presentations, and approximately 2000 emails. These documents provided contextual information for the observations.

Digital trace data. He further collected digital trace data from a software called 'Jira'. Jira supported teams Alpha and Beta in organizing software development via functions such as creating issues and adding information to these issues (e.g., textual descriptions). When actors performed those actions, Jira tracked them in a database. These data capture chronologies of actions over long periods of time (Mahringer & Pentland, 2021). Both the documents and the trace data extend beyond the observation period (see Figure 1).

Data Analysis

The starting point for our analysis was the observation that even though the teams initially thought that reorganizing their work in two relatively autonomous routine clusters would be highly beneficial (and ‘doable’ with manageable effort), dealing with the consequences of this reorganization was more challenging than assumed. We next describe the major steps through which we developed a theoretical explanation of this empirical observation. Even though we present the data analysis as a series of steps, we frequently went back and forth.

Step 1: Identifying the routine clusters of Team Alpha and Beta. First, we developed an abstract description of the routine cluster of Team Alpha, and subsequently the routine cluster of Team Beta. These descriptions provided us with an overview of those clusters, orienting our further research. We refer to the set of routines in each team as a routine cluster because it “consists of multiple, complementary routines, each contributing a partial result to the accomplishment of a common task” (Kremser & Schreyögg, 2016: 698); in our case, the development of the software product AlphaSoft or BetaSoft. We identified the routines of both clusters by coding our data set with ‘in vivo’ codes, which means that we closely adhered to the local actors’ language. Several routines, such as the Refinement or Review, for instance, were enacted as meetings in the virtual team calendar and were also labeled as such. We further made sure that each of those routines matched the definition as a “repetitive, recognizable pattern of interdependent actions, involving multiple actors” (Feldman & Pentland, 2003: 96).

Each routine cluster consisted of the following set of routines, which mattered in the reorganization process (see Figure 3). In the *Sprint Routine,* the developers resolved issues by changing software code and testing it. Each performance of the Sprint Routine lasted two weeks. Several other routines helped to organize the Sprints. These routines involved pre-scheduled meetings that took place at a specific place and time in each two-week iteration. In performing the *Planning 1 Routine*, a team determined which issues it would address in the next Sprint. In the *Planning 2 Routine,* the developers defined how to address these issues from a technical point of view. Team Beta did not make a difference between Planning 1 and 2, but performed a common

Planning Routine. The *Refinement Routine* was a meeting used to clarify each issue, roughly determining how to address it, and estimating the complexity of resolving it. The *Daily Scrum Routine* was a brief, maximum 15-minute meeting, used for synchronization on each morning during the Sprints. The *Review Routine* was performed at the end of each Sprint, and it aimed at checking the progress during the Sprint. The *Retrospective Routine* aimed at identifying improvements in the other routines. Moreover, the *Release Routine* was used to deliver the software to the clients (three to four times per year).

We further identified the pattern of actions for each routine (see Appendix B for details). This approach builds on a sequential analysis of workplace data (Geiger et al., 2021; Pentland, Hærem, & Hillison, 2011). We made sure that the identified patterns of actions were meaningful, avoiding both excessive complexity and simplicity. For this analysis, we used fieldnotes, interviews, and digital trace data. We cross-checked these routine descriptions with actors in the field to ensure validity. Even though this step provided us with an overview of the routine clusters, it did not help us to understand the reorganization itself. Hence, we became interested in the process through which the work was reorganized into two clusters.

Step 2: Analyzing the reorganization of routine clusters over time. In the second step, we strove to understand how the teams reorganized their work in two routine clusters. Although we first focused only on how Team Beta replicated the patterns of actions of Team Alpha (which they considered 'best practices'), we soon noticed that the teams also started to share resources (i.e., actors and artifacts). This observation aligns with research on routines, which not only emphasizes patterns of actions but also the relevance of actors and artifacts for the performances of routines (e.g., Blanche & Cohendet, 2021; Boe-Lillegraven, 2019; D'Adderio, 2011). Hence, we created a timeline (Langley, 1999), indicating events when the teams replicated routines or shared resources (Figure 4). Even though this helped us to understand what the teams reorganized, we had the hunch that replicating routines and sharing resources intermingled in a way that accounted for the challenges that the teams experienced.

Step 3: Identifying emerging interdependencies across the Alpha and Beta cluster. We realized that sharing resources and replicating routines led to situations in which the performances of routines in one cluster constrained the performances of routines in the other cluster. As this observation aligns with the argument that "[t]wo (or more) routines can be said to be interdependent to the extent that performing one routine creates an enabling and/or constraining context for performing the other(s)" (Rosa et al., 2021: 244), we came to view those problems as routine interdependencies that occurred *across* the Alpha and Beta cluster.

To understand these interdependencies, we applied Thompson's (1967) distinction between pooled, sequential, and reciprocal types. Pooled interdependencies mean that the performances of different routines rely on the same scarce resources. Routine performances are sequentially interdependent if the output of one routine serves as the input for another routine. Two routine performances are reciprocally interdependent if the outputs of both become inputs for the respective other. These three types of interdependencies build on each other, and thus involve increasing degrees of complexity (Van de Ven et al., 1976). We added those interdependencies, which actors created at different points in time over the course of the reorganization, to our event timeline. This visualization helped us to see that the teams started to create pooled interdependencies, while sequential and reciprocal interdependencies proliferated later in the reorganization process. Thus, we analytically bracketed our data in three periods, each of which focuses on a different type of interdependence.

Step 4: Identifying practices of interface design work. We then strove to understand how actors dealt with the interdependencies that became salient. Hence, we analyzed how the teams harmonized each interdependence that we had mapped in our timeline in an iterative process over several rounds. The first author developed initial explanations, while the second author came up with alternative explanations and challenged provisional interpretations (for a similar approach, see Rerup & Feldman, 2011). This process enabled us to settle on a set of four distinct practices (i.e., analyzing interdependencies, reconfiguring resources, mitigating interferences, and reconfiguring schedules) that helped actors to harmonize the interdependencies, and which we added to our event timeline. Table 2 provides definitions of each practice. We realized that those practices helped the teams to develop expectations regarding actions in the other cluster. This observation echoes Kremser and Schreyögg (2016) who emphasize that interfaces contain expectations regarding other routines. Therefore, we theorize that those practices are essential for the process of designing interfaces across routine clusters.

Step 5: Analyzing the dynamics of designing interfaces. We then strove to better understand the relations between reorganizing routine clusters, as well as creating and harmonizing interdependencies. Hence, we added the connections between those elements to our timeline (Sele, Mahringer, Danner-Schröder, Grisold, & Renzl, 2024), to identify the more genral patterns in our data. This step helped us to see that interface design work often entailed creating new, and potentially more complex interdependencies, that had to be dealt with.

FINDINGS: DESIGNING INTERFACES BETWEEN ROUTINE CLUSTERS

Based on our analysis, we structure our findings in three periods. Period 1 shows how the teams started with a conception of autonomy across the two envisioned routine clusters. However, as they began to share resources (i.e., artifacts and actors) and replicated routines, they created pooled interdependencies across the two clusters. In Period 2, we show how new pooled interdependencies emerged as the teams were disentangling the routine clusters, and how actors harmonized pooled interdependencies from both periods, which created sequential interdependencies. Period 3 shows how multiple interdependencies (some newly created, and some from previous periods) became intertwined in reciprocal interdependencies. As the teams worked through those interdependencies over time, they eventually managed to stabilize the interfaces between the two routine clusters.

Our findings further detail the work through which the teams harmonized those interdependencies, which became apparent in four different practices (for a summary, see Table 2). First, *analyzing interdependencies* means that actors unpack and describe resource-related, pooled interdependencies across routine clusters to inform the further work of designing interfaces. This practice harmonizes pooled interdependencies, and it may create further pooled interdependencies as actors gain novel insights. Second, *reconfiguring resources* means that actors change shared, scarce resources to make them available across clusters. This practice harmonizes pooled interdependencies, but it may create sequential ones when the work of reconfiguring resources conflicts with other work performed through routines. Third, *mitigating interferences* means that actors increase or decrease the priority of work to account for emergent interdependencies that result from designing interfaces across routine clusters. This practice harmonizes sequential or reciprocal interdependencies. Fourth, *reconfiguring schedules* refers to making relatively sustainable changes to the timing of patterns of actions to enable smooth coordination of routine performances at the interfaces between routine clusters. This practice can harmonize any type of interdependence. Both, mitigating interferences and reconfiguring schedules, may create reciprocal interdependencies when temporal pressure is created. In this case, problems that could usually be resolved sequentially need to be addressed simultaneously. Moreover, the four practices can be differentiated according to two characteristics (Table 2): *durability of interface design work*—whether it represents transitory or patterning work—as well as *design elements*— whether practices privilege artifacts or timing. Taken together, we show that these practices represent the work required to create interfaces across routine clusters, and thus we label them ‘interface design work’.

INSERT TABLE 2 HERE

We next zoom into each of the three chronologically ordered periods. In each period, we first show a vignette that focuses on the process of designing interfaces between the clusters. Accordingly, we analyze the interdependencies that became salient and the practices that actors used to harmonize those interdependencies. We further examine the processual dynamics apparent in each period. The text also refers to the Sprints of Team Alpha (chronologically from 'A1' to 'A10') and Beta ('B1' to 'B10') (see Figure 2). Table 1 defines the roles and names (pseudonyms) of the actors that played a central role. The names of Team Alpha members start with the letter 'A', Team Beta members with the letter 'B', and Austin-Blake is the actor who served as a developer in Team Alpha and Scrum Master in Team Beta (therefore, the double name with 'A-B').

INSERT FIGURE 2 HERE

Vignette 1: From a Conception of Autonomy to Proliferating Pooled Interdependencies

In November 2016, Team Beta started to operate based on the Scrum-approach. As Team Alpha had gathered a lot of experience with Scrum, Team Beta wanted to replicate Alpha's routines including the time and place of the meetings. For example, Team Alpha performed the Review and Retrospective routines on Mondays and the Planning 1 and 2 routines on Tuesdays. Figure 3 provides an overview of the typical Sprint cycle, including explanations of each routine. Team Beta replicated those meetings, including the same date and time as Team Alpha.

INSERT FIGURE 3 HERE

Consequences of sharing Austin-Blake. Austin-Blake, while a developer in Team Alpha, also started to serve as the Scrum Master in Team Beta, which meant that he needed to be involved in all meeting-based routines of both teams. This 'double role' promised to support the smooth functioning of the new cluster—"Austin-Blake is the connecting link" (Interview, Alpha developer, 08.02.2017). However, sharing Austin-Blake and replicating Alpha's meeting schedule led to overlapping meetings for Austin-Blake: "Austin-Blake has a double role as Scrum Master in Team Beta and as a developer in Team Alpha, and this is a problem" (Interview, Alex, 28.11.2016). Thus, the teams scheduled the meetings with an offset of one week between the Sprints of both teams.

Despite this adaptation, Austin-Blake's double role led to further meeting conflicts because Alpha's "Refinement meetings collided all the time [with the Planning meeting in Team Beta], and Austin-Blake was unable to participate on Tuesdays because he is the Scrum Master of Team Beta" (Interview, Andrew, 13.12.2016). Thus, Team Beta postponed its Planning-B1 performance to the afternoon, even though it was scheduled for the morning. Two weeks later, Team Beta performed their Planning-B2 as usual, while Team Alpha canceled their Refinement-A2.

In the performance of the Retrospective-A2 Routine, which was aimed at discussing problems in the previous Sprint, Alex, the Scrum Master of Team Alpha, handed pens and cards to each team member. He drew three columns on the whiteboard, marking the left one with 🙂, the middle column with 😐, and the right column with 🙁. He then asked the team to jot down positive aspects and impediments that arose during the Sprint. Alex noted "Austin-Blake's side gig" on a card and placed it under the column labeled 🙁. The team talked about how to address the problem. For instance, one team member asked whether Austin-Blake had to participate in all of Beta's routines, but Austin-Blake said that this was important. Instead, he suggested “maybe I don’t have to participate in the Refinement meetings” (Fieldnotes, A2). At the end of the discussion, the team concluded that the Product Owners of both teams should find a solution to the meeting conflict. Starting in January 2017 (Sprint A4), hence, Team Alpha rescheduled the performances of their Refinement Routine to Wednesdays and Fridays.

Consequences of sharing code components and the code repository. Another problem that became salient during Period 1 was how both teams could share selected code components, which Team Alpha had developed. The Product Owners of both teams stated that “our wish … is that we can develop each product without interdependencies” (Documentation, Jira, A1/B1), because “what if I change something in BetaSoft, and this, for whatever reason, crashes AlphaSoft? This would be crappy” (Interview, Beta developer, 22.03.2017). In their Planning-B2 Routine performance, thus, Brandon, the Product Owner of Team Beta, tasked the team with an issue to unpack how to share code components and to consider possible interdependencies. The creation of an ‘issue’ was the typical way in which the team identified tasks to be completed in the development of their software products. Here, they were using the same device (creating an ‘issue’) to address what was essentially an organizing problem. Brandon further explained to the team that “first, you need to create a diagram, second you need to talk about it with the Alpha guys” (Fieldnotes, B2). The diagram was to illustrate software components in boxes and the relationships between those components with arrows. It would also indicate on which of those components AlphaSoft and BetaSoft relied.

Mapping software components in the diagram revealed that the teams required a new technical procedure to merge the code into software products of AlphaSoft and BetaSoft that could be executed on the clients’ machines. As both clusters started to share code components, this procedure had to ensure that it merged matching versions of the code components underpinning either software product. Without a common procedure, the teams might provide faulty software products to their clients. Analyzing this emergent interdependence was done by each team planning an ‘issue’ to define the new procedure and solve that issue in the Sprint. Austin-Blake noted in Team Alpha’s Planning-A4 that “we have to look at this with two or three colleagues, and [we have to] ask a colleague from Team Beta” (Fieldnotes, A4). The teams collaborated across the performances of Sprints-A4 and B3, and finally illustrated the results of their dialogue in a second diagram, which showed how code components had to be merged in the software products.

As the teams were creating this second diagram during the Sprints, it became clear that they had to reconsider how to best organize their software code in the code repository. A ‘code repository’ is like a digital folder, in which developers collectively store and manage code components, including changes (i.e., versions) of those components. How to organize the code, however, was far from trivial. Alex, for instance, noted in Alpha’s Review-A4, “it is shitty, independent of how you implement it” (Fieldnotes, A4). Pointing to the diagram that the teams had crafted, and which was projected on the wall, a developer noted that “one option is to create distinct code repositories […] the problem is that we need to take care of the interfaces.” Andrew, the Product Owner of Team Alpha, noted, “we are caught between a rock and a hard place.” Alex nodded and replied “without additional work it’s not possible. The question is, how do we have less bugs.”

Therefore, Team Alpha created a new issue for the next Sprint, in which they would further unpack the interdependence. In Sprint-A5, Team Alpha created a third diagram that represented a concept for how code components could be stored in the code repository. In Team Alpha’s Review-A5, Austin-Blake presented the diagram: “we considered how to combine multiple projects under one roof and avoid having to program everything twice and three times” (Fieldnotes, A5). He narrated that they could create three distinct areas within one code repository. Each team would have its own area, and they wanted to create an area labeled ‘common’ that included code components that both teams could use. They could use the common area to easily share code components across both clusters.

> *A specific interdependence [between the routine clusters of both teams] is that they [Team Beta] want to use code components that we [Team Alpha] developed […] We could have said, 'you can do it on your own,' and just tell them how it should look. But we rather want to create a common area that both teams can use. And this is an interdependence, because we must take care that it works on our side but also in BetaSoft.* (Interview, Alpha developer, 08.02.2017)

Analysis of Period 1

Replicating routines and sharing resources creates interdependencies. Our analysis of Period 1 reveals that the teams created *pooled interdependencies* between the two routine clusters as Team Beta replicated Alpha's routines and as the teams shared resources (i.e., an actor and code components). The vignette shows how the teams shared Austin-Blake. Moreover, Team Beta replicated the routines of Team Alpha, which also included a set of meetings. Replicating routines and sharing an actor led to a pooled interdependence, which became salient as Austin-Blake was required for the routine performances in both clusters, but he could only attend one meeting at a time.

Our analysis further reveals that sharing selected code components of Team Alpha created pooled interdependencies, as the teams envisioned that using the same code could lead to problems. First, if both teams changed the same code components in their Sprints, this could lead to unintended bugs. Second, it was important that the correct code components were merged in the software products of AlphaSoft and BetaSoft, which the clients then executed on their machines. Otherwise, there could be malfunctions. Third, merging code components in both clusters required the teams to consider how they organized their code.

Harmonizing interdependencies through interface design work. Our analysis shows that the teams employed several practices through which they harmonized interdependencies and, thus, were designing interfaces. Period 1 shows two instances of a practice that we label *reconfiguring schedules*. First, Team Beta created a temporal offset of one week compared to Alpha's routine performances when they started to perform their routines. This enabled Austin-Blake to participate in both team's meetings. Second, Team Alpha changed its Refinement schedule when the Refinement meetings in Team Alpha clashed with the Planning meetings in Team Beta. The reconfiguring schedules practice defined a relatively durable expectation related to the timing of actions at the interfaces, enabling the coordinated involvement of Austin-Blake in both clusters.

Furthermore, the vignette shows three instances of the *analyzing interdependencies* practice. These instances supported harmonizing the pooled interdependencies that emerged from sharing code components. First, the teams assumed that if they changed the same code components in their Sprints, this could lead to bugs. However, it was not clear when and how this could happen. Hence, they analyzed this pooled interdependence, and visualized the results in a diagram. This analysis created more certainty about the interdependence across both clusters. Second, using the same code components meant that the correct versions of those components had to be merged into software products provided to the clients. The teams analyzed this interdependence (i.e., creating a second diagram) as they were not sure how to harmonize it. Third, if both teams wanted to change the same code components, they had to consider how to organize the code in their repository. Therefore, they analyzed this interdependence when they created a concept indicating how code could be organized in the code repository. This helped to understand the alternatives for this reorganization in designing interfaces. In all three instances, the pooled interdependence became clearer, informing further actions in designing interfaces in a meaningful way.

Overall process dynamics of designing interfaces. Figure 4 summarizes our insights from Period 1. The figure reveals how the teams created pooled interdependencies through replicating routines and sharing resources, which the teams harmonized through interface design work. Additionally, the figure shows a dynamic interplay between harmonizing pooled interdependencies and creating new ones. For instance, when addressing the pooled interdependence created by Austin-Blake's inability to attend meetings in both clusters, the teams used the reconfiguring schedules practice (i.e., creating the one-week offset between Sprint cycles). This, however, again rendered a pooled interdependence salient as Austin-Blake could not attend Alpha's Refinements and Beta's Plannings. Hence, they reiterated the reconfiguring schedules practice when Team Alpha changed their Refinement dates. Similarly, when the teams analyzed the pooled interdependence that resulted from sharing code components, a further pooled interdependence emerged (i.e., the Release performances in both teams would merge some of the same code when creating the software products), requiring a new instance of the analyzing interdependencies practice. As a consequence, a further pooled interdependence became salient (i.e., the organization of the code), which again had to be analyzed to inform designing the interface.

Vignette 2: From Harmonizing Pooled Interdependencies to Proliferating Sequential Interdependencies

Consequences of sharing code components. In Period 1 the teams had decided to share Alpha's code components and further analyzed how they could share them. Hence, Team Beta had illustrated the relationship of code components and how to merge them in a diagram in Sprint-B2 in Period 1. This diagram indicated that AlphaSoft's code was based on GroundSoft, an underlying third-party tool. Team Beta, however, decided not to develop their software code based on GroundSoft to save licensing fees and to be more flexible. This decision, however, implied that the teams could not simply share Alpha's existing code components, but these code components first had to be 'freed' from their linkages to GroundSoft.

In Period 2, thus, Team Alpha created an issue to free selected code components, and planned the issue for Sprint-A5. Meanwhile, Team Beta scheduled the implementation of a code component but realized that they had to wait until Team Alpha was done with freeing code. In the Planning-B4 Routine performance, a developer asked "did [... Team Alpha] already [free this component …], Austin-Blake?" (Fieldnotes, B4) and Austin-Blake responded that a developer of Team Alpha was "still fighting with this." Therefore, a Beta developer commented in Jira that 'this issue is blocked' by Team Alpha, and Brandon, glancing at Austin-Blake, concluded that "so we need to wait until [Team Alpha has] finished [this issue in] your current Sprint."

Consequences of sharing the code repository. Based on their analysis of how to organize their code in Period 1, the teams decided to share the code repository of Team Alpha, which required major modifications of the code repository. Team Alpha planned these reconfigurations, but problems emerged. A first problem related to creating release notes. In their Planning-A6, a developer asked, "by the way, how do we create release notes when Beta uses the same code repository?" (Fieldnotes, A6). Alex, without much confidence, mumbled, "this is a good question." Creating release notes was a crucial step through which the teams informed the clients about recent changes in AlphaSoft and BetaSoft. Those release notes were created by using a technical functionality offered by the code repository. As both clusters wanted to share the same code repository, however, release notes included changes in AlphaSoft and BetaSoft, even though clients might only be interested in the changes of one of those software products. Each team, thus, created an issue to analyze how release notes could be created. When Team Alpha planned this issue for their Sprint-A7, a developer noted that "for my taste, there are too many question marks [in the issue description]." Austin-Blake responded, "that's the point. We should figure out what to do. If we knew that already, we wouldn't have to do it." As the teams worked on these issues in their Sprints-A7, they identified alternatives to create release notes. Brandon, for instance, noted, "we could create the release notes with Git or Jira" (Fieldnotes, B5). Finally, they implemented a new software program that could tailor the release notes to either AlphaSoft or BetaSoft.

A second problem related to how both teams could bring their new code organization to the clients' machines once the code repository was reconfigured. AlphaSoft and BetaSoft both provided a part of the overall code. Therefore, installing only one of those software products on the client's machines would lead to a faulty code organization. For example, if a client had an old version of AlphaSoft and then installed the latest version of BetaSoft, the two programs might not work together. This could cause problems like parts of the programs getting mixed up or certain functions not working because they can't find the right connections. To avoid these problems, the teams decided to release both AlphaSoft and BetaSoft at the day.

It became also salient to the teams that the work of changing the code repository could interfere with regular work in the Sprints. In the Planning-A6 performance, thus, Andrew created a Sprint board for the next Sprint-A6 (as usual), but he also created an additional board entitled "Welcome BetaSoft" for the subsequent Sprint-A7, in which they planned to change the code repository. Alex noted, "our aim is to complete everything [that has to be completed before the repository change] in the next Sprint. Then we test it overnight and merge it. This is important for the Beta integration" (Fieldnotes, A6). Preparing the next two Sprints in advance was not typical but should ensure that they prioritized all the relevant work planned before the modification of the code repository, and that both teams could smoothly proceed with their work in the shared code repository afterwards.

Having addressed those problems, Team Alpha modified the code repository in Sprint-A7, while Team Beta deprioritized the development of a new BetaSoft function that could only be implemented after the new code repository was in place. In their Planning-B6, Brandon noted that "we need the new [repository] structure [to complete this functionality]" (Fieldnotes, B6), and the team prioritized other issues.

Analysis of Period 2

Sharing resources creates interdependencies. Our analysis of Period 2 reveals how sharing resources created new pooled interdependencies, but also proliferated sequential interdependencies. For instance, the vignette shows how a *pooled interdependence* became salient based on the analysis in Period 1, because Alpha's code components, which the teams wanted to share, were linked to GroundSoft. Team Beta, however, decided not to build their code on GroundSoft. Hence the teams had to reconfigure resources (i.e., code components) so that both teams could use them. This reconfiguration created a *sequential interdependence*, as Team Alpha had to free code components in their Sprint, but this work of freeing code components in Alpha's Sprint performance had to be done before Team Beta could plan and use those freed code components in their Sprints.

Moreover, based on the analysis of how to organize code components in Period 1, the decision to share the same code repository created a further *pooled interdependence*. This pooled interdependence became salient because release notes provided to the clients would incorporate information about AlphaSoft and BetaSoft, even though clients may only use one of those software products. Additionally, reorganizing the code repository could imply that incompatible code components were used on the clients' machines. To share the code repository the teams had to reconfigure it first, which rendered a *sequential interdependence* salient when Team Alpha scheduled changing the code repository in the Sprint. Here, the regular work of changing code in the Sprints of both teams had to be done either before or after changing the repository in Alpha's Sprint performance, because it was difficult to make changes in the code while the code repository was being reconfigured.

Harmonizing interdependencies through interface design work. Our findings show that the teams employed several practices through which they were able to harmonize interdependencies in the process of designing interfaces. Period 2 reveals two instances of the practice of *reconfiguring resources,* which was enacted to harmonize pooled interdependencies. First, Team Alpha had to reconfigure the code components by freeing them from their links to GroundSoft. Second, the teams wanted to share Alpha's code repository so that both clusters could use it, but this required major reconfigurations of the code repository. In Period 2, they thus enacted the concept that they had developed in Period 1, creating a common area for shared code components and one individual area for each routine cluster. This way, they changed the resource (i.e., the code repository) on which the routine performances in both clusters relied. Reconfiguring the code repository also entailed changes in the way it was used, and thus defined the interfaces across both clusters.

Moreover, Period 2 shows three instances of the practice of *mitigating interferences,* which was enacted to harmonize sequential interdependencies. First, Team Beta could not proceed with selected issues in their Sprint while Team Alpha was freeing code components. Hence, they postponed work that related to these code components to a later point in time. Second, Team Alpha created two Sprint boards in advance. This way, they emphasized the priority of completing the regular work in both teams' Sprints before Team Alpha changed the code repository in the following one. Planning two Sprints in advance should ensure a smooth transition, and was intended to prevent temporal problems resulting from the sequential interdependence that they had created. Third, Team Beta postponed work until the changed code repository was in place to manage the sequential interdependence that the teams had created. Thus, they did not have to change code that was stored in the code repository, and which was not available during the modification of the code repository. The practice of mitigating interferences supported the process of designing interfaces across clusters, because it enabled both teams to continue with their regular work while designing interfaces.

The teams also used the practice of *analyzing interdependencies* to harmonize a pooled interdependence. Each team created an issue to analyze the pooled interdependence that both clusters should use the same code repository, which had consequences for creating release notes. Initially, the issues were loaded with ambiguity, which one developer described as “here are too many question marks” (Fieldnotes, A7). Throughout the Sprints, analyzing made the interdependence clearer and clarified possible ways forward, for instance, by assessing different options to create release notes. Analyzing interdependencies finally enabled the teams to implement a program that differentiated between AlphaSoft’s and BetaSoft’s release notes. This practice supported the designing of interfaces by informing further courses of action.

Finally, Period 2 shows one instance of the *reconfiguring schedules* practice, which was used to harmonize the pooled interdependence related to sharing the same code repository. The teams decided that both routine clusters should release their software products on the same day. Synchronizing release dates ensured that the clients had the most recent version of both software products on their machines, preventing malfunctions that could emerge from sharing the code repository.

Overall process dynamics of designing interfaces. Figure 4 summarizes the analysis of Period 2, including connections to Period 1. Similar to Period 1, the vignette shows how new pooled interdependencies emerged as a consequence of efforts to harmonize interdependencies that became salient in Period 1. Moreover, our analysis reveals two instances in which actors created sequential interdependencies because of their attempts to harmonize pooled ones, which proliferated into a more complex type of interdependence. These sequential interdependencies emerged because the work of reconfiguring resources in specific Sprints interfered with other software development work. When Team Alpha freed code components to make them available for both clusters, Team Beta had to wait until Team Alpha's work was completed. Similarly, reconfiguring the code repository in Team Alpha's Sprint collided with other work in the Sprints of both clusters, creating another sequential interdependence. This scheduled reconfiguration of the code repository led to further interface design work, such as mitigating interferences through creating two Sprint boards in advance in Team Alpha and postponing regular work in Team Beta. Both teams were aware that they could not access the code repository during the time it was reconfigured. Hence, they scheduled their regular work either before or after this change. Our analysis thus shows that pooled interdependencies can proliferate to sequential ones when the resources that underlie those interdependencies (e.g., artifacts) need to be reconfigured. This is a notable difference to Period 1, where the resource (i.e., Austin-Blake) was not divisible, and thus the teams could only reconfigure schedules.

Vignette 3: Becoming Intertwined in Reciprocal Interdependencies and Stabilizing Interfaces

Consequences of sharing the flyout menu and synchronizing release dates. In Beta's Planning-B6 meeting before the joint release date with Team Alpha, a new concern emerged. Brandon noted "we have two Sprints left until our first release […] but we did not implement the flyout, yet." The 'flyout menu' was a button that, when clicked, expanded additional options (i.e., a code component). Brandon had created an issue in Jira to implement the flyout and he wanted to include it in the first release of BetaSoft. He had also commented in the Jira issue that the "design [of the flyout menu] should be similar to [the] attached flyout menu from AlphaSoft" (Documentation, B3), tasking the team to use the flyout that already existed in Team Alpha. Brandon, however, mentioned that Team Alpha had to make changes to their flyout before Beta could use it, and that Team Alpha had created an issue to make these changes. This was because Team Alpha wanted to extend the existing flyout for another purpose. As Team Beta scrolled through Alpha's backlog in Jira, however, they saw that the respective issue was "pretty far down in the backlog" (Fieldnotes, B6), indicating a low priority. Brandon conferred with Andrew, who, then, not only prioritized the respective Alpha issue to change the flyout menu for the next Sprint-A8, but he also tasked both teams to collaborate to successfully implement the flyout to meet the joint release date: "[Beta issue] and [Alpha issue] are interdependent! […] Please clarify which output Team Beta delivers at the end of their Sprint next week, which Team Alpha can use for [Alpha issue], and how this affects the workload and content of [Alpha issue]" (E-mail, A7). In Alpha's Planning-A8 performance, he further noted that "it is important that Beta can implement the flyout until the end of their next Sprint" to ensure that it could be included in the release. He also mentioned that a Beta developer "asked for their help, and so it would be good to cultivate cooperation" (Fieldnotes, A8) with Team Beta.

Consequences of replicating the stabilizing Sprint and synchronizing release dates. Before each release, Team Alpha avoided new features and instead "focus[ed] on [...] bugfixes" (Fieldnotes, Andrew, A8), which reduced the likelihood of releasing bugs to the clients. They referred to this as a 'Stabilizing Sprint'. Coming closer to the joint release date, which also was the first release of BetaSoft, Team Beta wanted to replicate the Stabilizing Sprint pattern from Team Alpha. In the Refinement-B4 performance before the envisioned release date, Brandon said that "the last Sprint will be for bugfixes, so technically, there are only two Sprints [focused on new features] left [until the release]" (Fieldnotes, B4).

It soon turned out, however, that "Brandon has doubts whether this [schedule] is feasible for Team Beta because they would [not have] enough time for stabilization" (Fieldnotes, Andrew, A8). Because the teams had created an offset between their Sprint cycles in Period 1, and because they wanted to release on the same day, which they had decided on in Period 2, Team Beta only had one week rather than two for stabilizing their code. This was not sufficient to perform a full stabilizing Sprint. Hence, the problem that the teams encountered resulted from the entanglement of a new concern (replicating the stabilizing Sprint) with consequences of harmonizing interdependencies in previous periods (creating the offset between Sprints; synchronizing release dates across the clusters). Therefore, Andrew reported to Team Alpha in their Planning-A8 that “Brandon asked, gnashing his teeth, whether [we] can postpone the release to the 13th of April” (Fieldnotes, A8). Thus, both teams postponed their joint release to this new date.

Some days later, in Alpha’s Review-A8 meeting, however, “Brandon indicated that the Beta guys cannot meet the new release date. I will talk to him because we can’t wait that long” (Fieldnotes, Andrew, A8). Team Beta had realized that they would not be able to implement all necessary code until the joint release date, and additionally stabilize it. Team Alpha, however, had already finished their work, and the clients required the AlphaSoft release for their machines. After Brandon and Andrew talked about this problem, Andrew decided that Team Alpha should help Team Beta in implementing three features that had to be included in the first release of BetaSoft. Thus, they had to postpone their other work for subsequent Sprints. Brandon joined Alpha’s Planning-A9 performance to hand-off the issues, and a developer of Team Beta worked with Team Alpha in Sprint-A9 to successfully resolve these issues.

As Team Alpha tried to resolve these issues for Team Beta, however, they were “overwhelmed that we just said we do it and then we don’t have a clue [what has to be done]” (Fieldnotes, A9). In the performance of the Retrospective-A9 Routine, Team Alpha acknowledged that their Sprint was “picked to pieces” (Fieldnotes, A9). Andrew, thus, noted that “I need to talk to Brandon. That’s not how it is supposed to be. I think that such interdependencies will also emerge in the future. It must go like clockwork.”

Stabilizing interfaces. Consequently, Andrew invited both teams to a meeting (being held during A10/B9) aimed at working out these problems. He wrote in the invitation e-mail that "the goal [of the meeting] is to find a way that minimizes tensions and helps us do our work in an efficient way" (E-mail, A10). A key insight from this meeting was that both teams should detect possible interdependencies across both clusters as early as possible when they refined issues. The developer who detected the interdependence should then assign the Product Owner of his or her team to the issue, who would then interact with the Product Owner of the other team. These adjustments to the routines in both teams should ensure "a reliable and fast handling of such tasks [… and to] make sure that all work is done, and no tasks get lost" (Documentation, A10/B9).

While the teams had invested substantial work over those months to design interfaces that functioned for them, there were substantially fewer problems after this meeting. It seemed that both teams had successfully developed expectations about how to coordinate their clusters. Thus, they rarely brought up problems related to the interfaces in the following months, and instead focused on developing their respective software product. For instance, Team Beta recognized in their Retrospective-B10 Routine performance that the Sprint was "super," as there were "no external disturbances" (Fieldnotes, B10). In their Planning-B15 meeting before the next joint release (in late July 2017), a Beta developer asked about the release. After a brief discussion, the team concluded that the Product Owners should talk to ensure that possible interdependencies that could threaten a successful release were considered. So, even though the teams still interacted, how to perform their interfaces became more mundane. A developer confirmed the impression that the interfaces had been stabilized for now. He noted: "I think [… the interaction across the clusters] works very well now. The process is clear, and when there is an issue with interdependencies it's not a big deal. There are no problems anymore" (Interview, Beta developer, 27.06.2017). Andrew also noted:

Once in a while, something goes wrong [because interdependencies emerge]. We are all humans, so you can't avoid it. […] Yes, it annoys one or the other team when you know: 'I did not change something here, why does it not work anymore?' But in the meantime [we know that you can …] ask [the other team]. You are sitting in the same room, and you ask: 'hey, did you change something here'? (Interview, Andrew, 03.07.2017).

Analysis of Period 3

Replicating routines and sharing resources creates interdependencies. The vignette shows another instance of how a *pooled interdependence* became salient, as both teams wanted to share the flyout menu (an artifact). Moreover, our analysis of Period 3 reveals how multiple interdependencies became intertwined in *reciprocal interdependencies*. First, both teams had to make changes that affected the other routine cluster to adapt the flyout so that it would fit both teams' purposes. Hence, the work of Team Alpha affected the work of Team Beta, and vice versa. This interdependence became salient because Team Beta could not include the flyout function in their release if Team Alpha did not make the respective changes, and Team Alpha also had to manage its capacity to successfully plan their Sprint. The teams, thus, faced time pressure to complete those changes due to the joint release date (Period 2).

Second, Period 3 shows another reciprocal interdependence, which resulted from three interrelated aspects: a) Team Beta replicated the 'Stabilizing Sprint' as an aspect of Alpha's routines (Period 3), b) both teams had decided to release their software products on the same day (Period 2), and c) the teams had created a one-week offset between both clusters (Period 1). In conjunction, replicating the Stabilizing Sprint pattern, creating the offset between Sprints, and the time pressure created through the joint release date clashed with consequences for both teams: The Beta cluster could not perform the full Stabilizing Sprint (i.e., two weeks) without postponing the release date, but the Alpha cluster faced pressure from clients to release the software. Hence, both clusters encountered challenging temporal tensions.

Harmonizing interdependencies through interface design work. The teams enacted different practices to harmonize these interdependencies. The vignette shows the practice of *reconfiguring resources*, which harmonized a pooled interdependence. Team Alpha had to make changes in the flyout menu so that Team Beta could also use it. At the same time, Team Beta extended the flyout menu with new functions that benefited both clusters. Hence, in designing the interface, both teams invested work to change the resource in a way that it would fit the needs of each cluster.

Our analysis further shows three instances of the *mitigating interferences* practice, which was used to harmonize reciprocal interdependencies. First, Team Alpha prioritized work when they assigned their respective issue (i.e., changing the flyout menu) a higher priority than they had planned. Usually, issues at the top of the backlog enter the next Sprint, but in this case, Team Alpha moved the issue from the bottom of their backlog to the top, because Team Beta asked them for this favor. This harmonized the interdependence because it contributed to Team Beta releasing the flyout to their clients on time. Second, the teams postponed their joint release date, mitigating insufficient time to stabilize the software and preventing bugs on clients' machines. Third, Team Alpha, by deprioritizing some of their own work, supported Team Beta's resolution of issues so that both teams could meet their joint release date. Mitigating these interferences enabled Team Beta to release all the features that they had planned for their first release, but it also enabled Team Alpha to provide the clients with the most recent version of AlphaSoft in due time.

The vignette also shows the *reconfiguring schedules* practice, which harmonized a reciprocal interdependence. The teams decided that they should detect future interdependencies across both routine clusters early on and inform the Product Owner accordingly. This promised to harmonize consequences that may result from interdependencies, particularly in the Planning and Sprint performances of both teams. This way, they defined a temporal expectation (i.e., when and how to handle interdependencies) at the interfaces of both clusters.

Overall process dynamics of designing interfaces. Figure 4 summarizes the analysis of Period 3, including connections to the other periods. The analysis shows another instance of a newly created pooled interdependence. It also shows how multiple attempts of harmonizing interdependencies create two reciprocal interdependencies. Interdependencies became reciprocal because of time pressure that emerged from the combination of the joint release date (Period 2) with other practices of designing interfaces. In the first instance, both teams scheduled the reconfiguration of code related to the flyout menu. However, as Team Beta faced time pressure related to the upcoming joint release, Team Alpha prioritized their respective issue, and then both teams had to work in parallel on reconfiguring the code components. The second instance also results from several consequential endeavors: In Period 1, the teams had shared Austin-Blake (i.e., an actor), which required them to create an offset between their Sprint cycles. In Period 2, they had shared the code repository (i.e., an artifact), which led them to synchronize their release dates. In Period 3, they replicated the 'Stabilizing Sprint'. Those elements in conjunction, however, created a reciprocal interdependence across both routine clusters. This way, problems that might have emerged in sequential form had to be solved simultaneously, which is characteristic of reciprocal interdependencies.

INSERT FIGURE 4 HERE

DISCUSSION

This paper focuses on a challenge that many contemporary organizations face when they reorganize: How can they disentangle work into autonomous routine clusters when emergent interdependencies are omnipresent? We argue that this challenge requires turning our attention to the process of designing interfaces between routine clusters, and the ‘interface design work’ entailed. Thus, interfaces must harmonize interdependencies to make autonomous work possible.

Designing Interfaces as an Endogenous Process of Creating and Harmonizing Interdependencies

In a stable world, it is assumed that exogenous designers can comprehend the organization’s interdependencies, enabling them to design interfaces (Lawrence & Lorsch, 1967; Puranam et al., 2012; Van de Ven et al., 1976). This assumption, however, becomes problematic in a world in flux (Birnholtz et al., 2007; Mahringer et al., 2024), in which emergent interdependencies are omnipresent (Deken et al., 2016; Mayo, 2022; Pentland et al., 2016; Raveendran et al., 2020). Our study shows that designing interfaces is an endogenous and iterative process of creating and harmonizing interdependencies when interdependencies are emergent.

Figure 5 summarizes our insights in a process model. Overall, the model shows how actors reorganize their work in two routine clusters (i.e., the grey shapes at the top), which includes the creation of stabilized interfaces between those clusters (i.e., the thick arrows). The bottom part of the figure zooms into the process of designing interfaces between routine clusters. It shows how replicating routines and sharing resources create interdependencies between emergent routine clusters. Actors harmonize these interdependencies through interface design work, and specifically the four practices of analyzing interdependencies, reconfiguring resources, mitigating interferences, and reconfiguring schedules. These practices, however, may create further, and potentially more complex (i.e., sequential or reciprocal), interdependencies, initially resulting in heightened forms of entanglement. Through this iterative process of creating and harmonizing interdependencies, actors eventually stabilize the interfaces between routine clusters, finally disentangling work into two routine clusters. This way, our research stresses the emerging consequences (Deken et al., 2016; Dittrich & Seidl, 2018) and the serendipitous nature of interdependencies (Sele & Grand, 2016) in the becoming (Tsoukas & Chia, 2002) of interfaces. Shifting from an entitative to a processual perspective enabled us to show the consequences of emergent interdependencies for the process of designing interfaces.

INSERT FIGURE 5 HERE

This shift has important implications. Though scholars tend to assume that interfaces can be designed *apart* from the performances of routines (Gupta et al., 2015; Kremser & Schreyögg, 2016), we find that designing interfaces can also happen while actors are performing routines. In this vein, we address Glaser (2017: 2148), who has called for research on routine design when "design performances are more tightly connected to the ongoing performance of routines." Our research reveals that those 'design performances' may not be distinct from routine performances. In contrast, interfaces can be designed through routine performances.

Moreover, scholars often assume that interfaces are designed by exogenous designers possessing architectural knowledge, whereas routine participants are recipients tasked to enact those designs (Baldwin, 2010; Baldwin & Clark, 2000; Brusoni et al., 2023; Carroll, Gormley, Bilardo, Burton, & Woodman, 2006; Puranam et al., 2012). By contrast, we show that routine participants can be designers themselves. This insight goes beyond the idea of 'co-designers' (Raveendran et al., 2020), because it locates agency primarily in the actors performing routines. Put differently, routine participants can also be designers who are endogenous to the performances of routines.

Uncovering Interface Design Work

Our findings uncover the importance of ‘interface design work’, which refers to the effortful “actions specifically associated with designing routine interfaces” (Wegener & Glaser, 2021: 309). The effortful nature of interface design work becomes apparent in the practices (i.e., analyzing interdependencies, mitigating interferences, reconfiguring resources, and reconfiguring schedules) that we identified. As shown in Table 2, we argue that these interface design practices can be differentiated according to two orthogonal dimensions. Specifically, the first dimension of *durability of interface design work* refers to whether a practice involves *transitory work* aimed at solving situated problems that emerge during interface design, or whether it involves longer-term *patterning work* that has more sustainable consequences for interfaces. In particular, the practices of analyzing interdependencies and mitigating interferences involve transitory work while the practices of reconfiguring resources and reconfiguring schedules imply longer-term patterning (see Table 2). The second dimension concerns the *interface design elements* privileged for a practice, specifically, whether it involves designing interfaces through *artifacts* or through *timing*. The practices of analyzing interdependencies and reconfiguring resources privilege artifacts, while mitigating interferences and reconfiguring schedules involve designing interfaces based on timing.

As explained above, durability of interface design work means that practices either involve patterning or transitory work. In the context of routines research, Feldman and Pentland (2022: 850) define patterning as referring “to the impact of patterns on actions and the ongoing creation of patterns through actions.” Designing interfaces involves patterning in the latter sense, when actors create patterns of interaction (i.e., interfaces) across routine clusters, notably by reconfiguring resources and schedules. We extend research on patterning (Danner-Schröder & Geiger, 2016; Goh & Pentland, 2019; LeBaron, Christianson, Garrett, & Ilan, 2016; Pentland, Mahringer, Dittrich, Feldman, & Ryan Wolf, 2020) by introducing the mutually constitutive dynamics between resources and interfaces as a new patterning mechanism. First, our findings reveal that the resources used in the existing routine cluster were not readily available in the new routine cluster. Accordingly, actors engaged in interface design work by reconfiguring those resources performed through their regular routines. This finding is in line with previous research which has shown that resourcing (i.e., changing and reinterpreting resources) may be enacted through routine performances (Feldman, 2004). Our research also shows, however, how reconfiguring resources changes the patterning of interactions at the interfaces between routine clusters. Hence, we add to previous research by showing how resourcing can constrain and enable the patterning of actions at the interfaces between routine clusters. Performing, patterning, and resourcing, thus form a mutually constitutive relationship. This argument echoes research that has shown how artifacts (as specific types of resources) enable and constrain patterns of actions (D'Adderio, 2011; Glaser, 2017; Omidvar, Safavi, & Glaser, 2023). Second, when actors are unable to harmonize interdependencies by dividing resources (e.g., an actor that participates in both clusters), they may reconfigure schedules to adjust the timing of resource use.

Thus, these two practices (reconfiguring resources and reconfiguring schedules) jointly make sustainable changes in interfaces. Stabilized interfaces include expectations about a) how and when each cluster uses scarce, shared resources, and b) how and when to interact with the other cluster. Although this does not mean that interfaces are carved in stone, patterning work is important because actors develop expectations related to interactions with other routine clusters (Kremser et al., 2019; Kremser & Schreyögg, 2016; Spee et al., 2016). Those expectations may become temporarily stabilized, but they can still evolve over time.

On the other hand, other interface design practices entail *transitory work*. In other words, interface design work not only includes actions directly aimed at patterning interfaces but also practices that inform the further process of designing interfaces by analyzing interdependencies and mitigating interferences through ongoing routine performances. We thus advance research on effortful routine performances (Danner-Schröder & Geiger, 2016; Pentland & Rueter, 1994; Sailer et al., 2023; Spee et al., 2016), by stressing the supportive role of transitory work practices in designing interfaces. First, the practice of analyzing interdependencies captures the efforts required to inform further actions. This practice shows that actors may not design interfaces through a trial-and-error learning process (Rerup & Feldman, 2011), but they may also perform meaningful action (Kramer, 2007). Meaningful action refers to working through the problems at hand and creating a preliminary understanding, which needs to be constantly updated (Christianson, 2019). This practice echoes Wegener and Glaser (2021: 308), who draw our attention to the "unfolding of the design process through actions that generate a deeper understanding of the situation, and in turn lead to new actions." Because only action can generate new information (Dittrich & Seidl, 2018; Rudolph, Morrison, & Carroll, 2009), analyzing interdependencies is an important prerequisite for the patterning work practices that design interfaces more sustainably. Second, the practice of mitigating interferences shows how actors deal with the possible mismatches between the accomplishment of regular routine goals (e.g., developing a software product) and the designing of interfaces through the same routine performances. As these goals are not always complementary (Balogun & Johnson, 2004; Salvato & Rerup, 2018; Turner & Rindova, 2012), and designing interfaces through artifacts takes elapsed time, the practice of mitigating interferences is needed to dynamically prioritize the work of designing interfaces and regular work within the same performances.

Unpacking the Role of Resources in Designing Interfaces

Although most research on routine interdependence has focused on patterns of actions (Kremser et al., 2019; Spee et al., 2016; Turner & Rindova, 2012; Yamauchi & Hiramoto, 2020), it has been implicitly taken for granted that the resources required to perform those patterns are readily available. Against this backdrop, Sailer et al. (2023: 1886) recently hinted at the importance of orchestrating pooled interdependencies, where "routines share the same material, spatial, or temporal resources." The authors further demonstrate how actors coordinate resources, for example, by using simple printed schedules. This finding is in line with the idea that resources, and their associated pooled interdependencies, are simple to coordinate (Thompson, 1967; Van de Ven et al., 1976): "with pooled interdependence, action can proceed without regard to action in other positions [i.e., clusters]" (Hoekzema, 2020: 565). However, our findings show that creating a configuration of resources that can be used autonomously across routine clusters can be challenging when actors reorganize work into two autonomous routine clusters. A key reason is that resources are not always readily available or simple to share or replicate, but they can become rival resources (Cornes & Sandler, 1996).

We identified two different kinds of *design elements* that actors may draw on to create interfaces (see design element dimension in Table 2). These design elements depend on whether resources are divisible or indivisible. On the one hand, interface design work may influence divisible resources, such as artifacts. In those cases, actors are able to analyze interdependencies and reconfigure resources. Resourcing, thus, not only entails the reinterpretation or reconnection of existing resources (Deken, Berends, Gemser, & Lauche, 2018; Feldman, 2004; Wiedner, Barrett, & Oborn, 2017), but it might also entail analysis and reconfiguration of resources. On the other hand, designing interfaces can entail situations which cannot be addressed by designing artifacts because resources are indivisible (e.g., Austin-Blake could not participate in both clusters performances at the same time). In those cases, we find that actors manage interfaces through *timing*, either by mitigating interferences or reconfiguring schedules.

Moreover, our findings show that resource sharing can be challenging because pooled interdependencies may often escalate into more complex types—as we saw in the proliferation from pooled to sequential to reciprocal interdependences across the three periods in our study. As commonly known, pooled interdependencies result from sharing scarce resources (Salancik & Pfeffer, 1978; Thompson, 1967). Harmonizing those pooled interdependencies may require actors to reconfigure resources in a way that they become usable across clusters. Reconfiguring those resources, however, requires work, and this work may conflict with other work performed through routines. Classic interdependence theory suggests that routine clusters are interdependent if one cluster must fulfill requirements before the other cluster can proceed with its work (Victor & Blackburn, 1987). In the process of designing interfaces, actors may need to reconfigure resources before regular work can continue. Pooled interdependencies then become sequential because the output of routine performances in one cluster (e.g., reconfiguring resources) is the input for the regular work of the other cluster (Thompson, 1967), representing a unidirectional, sequential interdependence (McCann & Ferry, 1979; Victor & Blackburn, 1987).

Furthermore, harmonizing interdependencies can create reciprocal ones if, a) multiple interdependencies and their attempts to harmonize come together, and b) this constellation is prone to time pressure. This finding shows that the distinction between sequential and reciprocal interdependence is a temporal problem: If teams Alpha and Beta had not synchronized their release dates, they would have been able to resolve interdependencies sequentially, but this was not possible due to time pressure. Put differently, reciprocal interdependencies emerge when sequential interdependencies are temporally compressed. This insight addresses Raveendran et al.'s (2020) call to examine the directionality of interdependence, by showing that unidirectional interdependencies (i.e., sequential) can become bi-directional (i.e., reciprocal) under time pressure. In sum, these findings show how and why disentangling resources, and harmonizing the corresponding pooled interdependencies, may unintentionally lead to the proliferation of more complex types of interdependence in the process of designing interfaces.

BOUNDARY CONDITIONS AND FUTURE RESEARCH

We developed our theoretical insights based on a single-case study in agile teams of a high-tech manufacturing company. This empirical context is well-suited to understanding the reorganization of work in autonomous routine clusters against a background of emergent interdependencies, because the Scrum framework encourages self-organizing teams and software development is usually highly interdependent. Despite the suitability of this case, several boundary conditions could inspire future research.

First, we studied a setting in which two routine clusters shared scarce resources. Sharing those resources created pooled interdependencies that actors had to harmonize, and which could escalate into more complex types of interdependencies. If management had assigned the Beta cluster with a similar but different set of resources as the Alpha cluster, and if the teams had not started to share actors and artifacts due to expected synergies, those pooled interdependencies might not have emerged. Similarly, resources that can be copied indefinitely (e.g., a simple computer file) might not lead to similar dynamics. Although such cases might exist, we believe that the assumption of resource independence is problematic because most organizations work under resource constraints. Future research could examine the consequences of such resource sharing for routines, for instance, by looking at cases in which more than two clusters share resources, or when management orchestrates resources across multiple clusters. In settings in which agile methods are used at scale, we assume that it is difficult to stabilize the process of designing interfaces because implementing new teams might require further iterations of interface design work. It might also be interesting to perform a processual analysis of settings in which resource allocation changes over time, for instance, when teams start with scarce resources but receive further resources later in the process (or vice versa).

Second, in our study, we observed that actors intended to create autonomous routine clusters. Whereas this is typical for many settings such as agile software development or project-based organizations, scholars might observe different dynamics when autonomy does not matter to the same degree. If the clusters had shared resources but not valued autonomy, they might have ended with one cluster rather than two. Ironically, our teams initially envisioned to develop and release their software products autonomously, but it turned out that they depended on each other in disentangling the two routine clusters. We do not know whether this is more beneficial, but future research could examine how and under which conditions multiple clusters collapse into one. On a related note, we observed that the reorganization involved the replication of Team Alpha's routines in Team Beta. Future research could examine reorganizations involving clusters with entirely different routines. For instance, different interdependencies might emerge when clusters differ in their temporal rhythms.

Third, Scrum organizes development work through a combination of issues (i.e., task decomposition) and meetings. This combination allows actors to repurpose the notion of 'issues' to design interfaces and to process those issues through meetings. In cases in which routines are not tailored to perform generative work, by contrast, our insight that design happens through routine performances might not be applicable, or designing interfaces might unfold differently. Future research could focus on settings in which routines do not rely on Scrum, shedding more light on the role of task decomposition for the process of designing interfaces. Moreover, the Scrum routines incorporate meetings intended to facilitate collective reflection (Bucher & Langley, 2016; Dittrich, Guérard, & Seidl, 2016). In other settings, routines might not allow for reflection to the same degree, which might exacerbate the process of designing interfaces.

Fourth, we observed that the teams in our case wanted to make both clusters work. Notably, Team Alpha performed a lot of work in designing interfaces. By contrast, in cases in which there is less mutual respect (Wiedner & Mantere, 2019), scholars might observe that actors are less willing to work together when problems emerge. It would be interesting to examine whether this leads to breakdowns, or whether there are other solutions for such cases. These cases might draw attention to the power dynamics occurring between routine clusters.

CONCLUSION

This paper unpacks the process through which actors engage in designing interfaces, allowing them to create autonomous routine clusters against a background of emergent interdependencies. The findings contribute to research on routine dynamics, interdependence, and organizational design. In essence, we emphasize the endogenous process of designing interfaces, which reveals how designing may happen within the performances of routines, rather than apart from them, and which may endogenously be carried out by routine participants. Second, we unpack interface design work, revealing that some practices change the patterning of interfaces more sustainably while others play a supportive role. Third, we stress the intricate nature of resources in designing interfaces, which becomes evident in the challenging coordination requirements and the possibility of escalations into more complex types. In conclusion, we encourage future research to pay attention to the emergent and situated nature of designing routine clusters and organizations.

REFERENCES

Adler, P. S. 1995. Interdepartmental interdependence and coordination: The case of the design/manufacturing interface. ***Organization Science***, 6(2): 147–167.

Arregle, J.-L., Dattée, B., Hitt, M. A., & Bergh, D. 2023. Organizational autonomy: A review and agenda for future research. ***Journal of Management***, 49(1): 85–124.

Baham, C. & Hirschheim, R. 2022. Issues, challenges, and a proposed theoretical core of agile software development research. ***Information Systems Journal***, 32(1): 103–129.

Baldwin, C. Y. & Clark, K. B. 2000. ***Design rules. The power of modularity***. Cambridge: MIT Press.

Baldwin, C. Y. 2010. When open architecture beats closed: The entrepreneurial use of architectural knowledge. Working paper no. 10-063, Harvard Business School, Boston, MA.

Balogun, J. & Johnson, G. 2004. Organizational restructuring and middle manager sensemaking. ***Academy of Management Journal***, 47(4): 523–549.

Birnholtz, J. P., Cohen, M. D., & Hoch, S. V. 2007. Organizational character: On the regeneration of Camp Poplar Grove. ***Organization Science***, 18(2): 315–332.

Blanche, C. & Cohendet, P. 2021. Replication and routine dynamics. In M. S. Feldman, B. T. Pentland, L. D'Adderio, K. Dittrich, C. Rerup, & D. Seidl (Eds.), ***Cambridge handbook of routine dynamics***: 277–287. Cambridge: Cambridge University Press.

Boe-Lillegraven, S. 2019. Transferring routines across multiple boundaries: A flexible approach. ***Research in the Sociology of Organizations***, 61: 31–53.

Brusoni, S., Henkel, J., Jacobides, M. G., Karim, S., MacCormack, A., Puranam, P., & Schilling, M. 2023. The power of modularity today: 20 years of "design rules". ***Industrial and Corporate Change***, 32(1): 1–10.

Bucher, S. & Langley, A. 2016. The interplay of reflective and experimental spaces in interrupting and reorienting routine dynamics. ***Organization Science***, 27(3): 594–613.

Burns, T. & Stalker, G. M. 1961. ***The management of innovation***. London: Tavistock Publications.

Carroll, T. N., Gormley, T. J., Bilardo, V. J., Burton, R. M., & Woodman, K. L. 2006. Designing a new organization at NASA: An organization design process using simulation. ***Organization Science***, 17(2): 202–214.

Christianson, M. K. 2019. More and less effective updating: The role of trajectory management in making sense again. ***Administrative Science Quarterly***, 64(1): 45–86.

Cloutier, C. & Langley, A. 2020. What makes a process theoretical contribution? ***Organization Theory***, 1(1): 1–32.

Cohen, S. G. & Ledford Jr, G. E. 1994. The effectiveness of self-managing teams: A quasi-experiment. ***Human Relations***, 47(1): 13–43.

Cornes, R. & Sandler, T. 1996. ***The theory of externalities, public goods, and club goods***. Cambridge: Cambridge University Press.

D'Adderio, L. 2011. Artifacts at the centre of routines: Performing the material turn in routines theory. ***Journal of Institutional Economics***, 7(2): 197–230.

D'Adderio, L. & Pollock, N. 2020. Making routines the same: Crafting similarity and singularity in routines transfer. ***Research Policy***, 49(8): 1–15.

Danner-Schröder, A. & Geiger, D. 2016. Unravelling the motor of patterning work: Toward an understanding of the microlevel dynamics of standardization and flexibility. ***Organization Science***, 27(3): 633–658.

Deken, F., Carlile, P. R., Berends, H., & Lauche, K. 2016. Generating novelty through interdependent routines: A process model of routine work. ***Organization Science***, 27(3): 659–677.

Deken, F., Berends, H., Gemser, G., & Lauche, K. 2018. Strategizing and the initiation of interorganizational collaboration through prospective resourcing. ***Academy of Management Journal***, 61(5): 1920–1950.

Dittrich, K., Guérard, S., & Seidl, D. 2016. Talking about routines: The role of reflective talk in routine change. ***Organization Science***, 27(3): 678–697.

Dittrich, K. & Seidl, D. 2018. Emerging intentionality in routine dynamics: A pragmatist view. ***Academy of Management Journal***, 61(1): 111–138.

Dönmez, D., Grote, G., & Brusoni, S. 2016. Routine interdependencies as a source of stability and flexibility. A study of agile software development teams. ***Information and Organization***, 26(3): 63–83.

Dougherty, D. 2001. Reimagining the differentiation and integration of work for sustained product innovation. ***Organization Science***, 12(5): 612–631.

Emerson, R. M., Fretz, R. I., & Shaw, L. L. 2011. ***Writing ethnographic fieldnotes***. Chicago: University of Chicago Press.

Feldman, M. S. & Pentland, B. T. 2003. Reconceptualizing organizational routines as a source of flexibility and change. ***Administrative Science Quarterly***, 48(1): 94–118.

Feldman, M. S. 2004. Resources in emerging structures and processes of change. ***Organization Science***, 15(3): 295–309.

Feldman, M. S. & Orlikowski, W. J. 2011. Theorizing practice and practicing theory. ***Organization Science***, 22(5): 1240–1253.

Feldman, M. S. & Pentland, B. T. 2022. Routine dynamics: Toward a critical conversation. ***Strategic Organization***, 20(4): 846–859.

Garnier, G. H. 1982. Context and decision making autonomy in the foreign affiliates of US multinational corporations. ***Academy of Management Journal***, 25(4): 893–908.

Geiger, D., Danner-Schröder, A., & Kremser, W. 2021. Getting ahead of time—performing temporal boundaries to coordinate routines under temporal uncertainty. ***Administrative Science Quarterly***, 66(1): 220–264.

Glaser, V. L. 2017. Design performances: How organizations inscribe artifacts to change routines. ***Academy of Management Journal***, 60(6): 2126–2154.

Goh, K. & Pentland, B. T. 2019. From actions to paths to patterning: Toward a dynamic theory of patterning in routines. ***Academy of Management Journal***, 62(6): 1901–1929.

Gupta, A., Hoopes, D. G., & Knott, A. M. 2015. Redesigning routines for replication. ***Strategic Management Journal***, 36(6): 851–871.

Hobday, M. 2000. The project-based organisation: An ideal form for managing complex products and systems? ***Research Policy***, 29(7): 871–893.

Hoekzema, J. 2020. Bridging the gap between ecologies and clusters: Towards an integrative framework of routine interdependence. ***European Management Review***, 17(2): 559–571.

Kramer, E. H. 2007. ***Organizing doubt: Grounded theory, army units and dealing with dynamic complexity***. Copenhagen: Copenhagen Business School Press.

Kremser, W. & Schreyögg, G. 2016. The dynamics of interrelated routines: Introducing the cluster level. ***Organization Science***, 27(3): 698–721.

Kremser, W., Pentland, B. T., & Brunswicker, S. 2019. Interdependence within and between routines: A performative perspective. ***Research in the Sociology of Organizations***, 61: 79–98.

Kremser, W. & Blagoev, B. 2021. The dynamics of prioritizing: How actors temporally pattern complex role–routine ecologies. ***Administrative Science Quarterly***, 66(2): 339–379.

Kremser, W. & Xiao, J. 2021. Self-managed forms of organizing and routine dynamics. In M. S. Feldman, B. T. Pentland, L. D'Adderio, K. Dittrich, C. Rerup, & D. Seidl (Eds.), ***Cambridge handbook of routine dynamics***: 421–432. Cambridge: Cambridge University Press.

Langfred, C. W. 2007. The downside of self-management: A longitudinal study of the effects of conflict on trust, autonomy, and task interdependence in self-managing teams. ***Academy of Management Journal***, 50(4): 885–900.

Langley, A. 1999. Strategies for theorizing from process data. ***Academy of Management Review***, 24(4): 691-710.

Langley, A. & Tsoukas, H. 2012. Introducing "perspectives on process organization studies". In T. Hernes & S. Maitlis (Eds.), ***Process, sensemaking, and organizing***: 1–26. Oxford: Oxford University Press.

Lawrence, P. R. & Lorsch, J. W. 1967. Differentiation and integration in complex organizations. ***Administrative Science Quarterly***, 12(1): 1–47.

LeBaron, C., Christianson, M. K., Garrett, L., & Ilan, R. 2016. Coordinating flexible performance during everyday work: An ethnomethodological study of handoff routines. ***Organization Science***, 27(3): 514–534.

Lee, M. Y. & Edmondson, A. C. 2017. Self-managing organizations: Exploring the limits of less-hierarchical organizing. ***Research in Organizational Behavior***, 37: 35–58.

Lundin, R. A. & Söderholm, A. 1995. A theory of the temporary organization. ***Scandinavian Journal of Management***, 11(4): 437–455.

Mahringer, C. A. & Pentland, B. T. 2021. Sequence analysis in routine dynamics. In M. S. Feldman, B. T. Pentland, L. D'Adderio, K. Dittrich, C. Rerup, & D. Seidl (Eds.), ***Cambridge handbook of routine dynamics***: 172–183. Cambridge: Cambridge University Press.

Mahringer, C. A., Pentland, B. T., Renzl, B., Sele, K., & Spee, P. 2024. Routine dynamics: Organizing in a world in flux. ***Research in the Sociology of Organizations***, 88: 1–15.

Mayo, A. T. 2022. Syncing up: A process model of emergent interdependence in dynamic teams. ***Administrative Science Quarterly***, 67(3): 821–864.

McCann, J. E. & Ferry, D. L. 1979. An approach for assessing and managing inter-unit interdependence. ***Academy of Management Review***, 4(1): 113–119.

Oliver, C. 1991. Network relations and loss of organizational autonomy. ***Human Relations***, 44(9): 943–961.

Omidvar, O., Safavi, M., & Glaser, V. L. 2023. Algorithmic routines and dynamic inertia: How organizations avoid adapting to changes in the environment. ***Journal of Management Studies***, 60(2): 313–345.

Pentland, B. T. & Rueter, H. H. 1994. Organizational routines as grammars of action. ***Administrative Science Quarterly***, 39(3): 484–510.

Pentland, B. T., Hærem, T., & Hillison, D. 2011. The (n)ever-changing world: Stability and change in organizational routines. ***Organization Science***, 22(6): 1369–1383.

Pentland, B. T., Recker, J., & Wyner, G. 2016. Conceptualizing and measuring interdependence between organizational routines, ***ICIS Proceedings***. Dublin.

Pentland, B. T., Mahringer, C. A., Dittrich, K., Feldman, M. S., & Ryan Wolf, J. 2020. Process multiplicity and process dynamics: Weaving the space of possible paths. ***Organization Theory***, 1(3): 1–21.

Puranam, P., Raveendran, M., & Knudsen, T. 2012. Organization design: The epistemic interdependence perspective. ***Academy of Management Review***, 37(3): 419–440.

Puranam, P. & Raveendran, M. 2013. Interdependence and organization design. In A. Grandori (Ed.), ***Handbook of economic organization***: 193–209. Cheltenham: Edward Elgar.

Raveendran, M., Silvestri, L., & Gulati, R. 2020. The role of interdependence in the micro-foundations of organization design: Task, goal, and knowledge interdependence. ***Academy of Management Annals***, 14(2): 828–868.

Rerup, C. & Feldman, M. S. 2011. Routines as a source of change in organizational schemata: The role of trial-and-error learning. ***Academy of Management Journal***, 54(3): 577–610.

Ritter, F., Danner-Schröder, A., & Müller-Seitz, G. 2024. Agile routines enabling efficiency and flexibility: Demarcating and integrating temporal orientations. ***Research in the Sociology of Organizations***, 88: 151–177.

Rosa, R. A., Kremser, W., & Bulgacov, S. 2021. Routine interdependence: Intersections, clusters, ecologies and bundles. In M. S. Feldman, B. T. Pentland, L. D'Adderio, K. Dittrich, C. Rerup, & D. Seidl (Eds.), ***Cambridge handbook of routine dynamics***: 244–254. Cambridge: Cambridge University Press.

Rudolph, J. W., Morrison, J. B., & Carroll, J. S. 2009. The dynamics of action-oriented problem solving: Linking interpretation and choice. ***Academy of Management Review***, 34(4): 733–756.

Sailer, P., Loscher, G. J., & Kaiser, S. 2023. Coordinated interdependence: How patterning governs flexibility in a routine cluster. ***Journal of Management Studies***, 61(5): 1884–1915.

Salancik, G. R. & Pfeffer, J. 1978. A social information processing approach to job attitudes and task design. ***Administrative Science Quarterly***, 23(2): 224–253.

Salvato, C. & Rerup, C. 2018. Routine regulation: Balancing conflicting goals in organizational routines. ***Administrative Science Quarterly***, 63(1): 170–209.

Schwaber, K. & Sutherland, J. 2020. The Scrum guide. The definite guide to Scrum: The rules of the game. https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. Access date: 27.01.2021.

Sele, K. & Grand, S. 2016. Unpacking the dynamics of ecologies of routines: Mediators and their generative effects in routine interactions. ***Organization Science***, 27(3): 722–738.

Sele, K., Mahringer, C. A., Danner-Schröder, A., Grisold, T., & Renzl, B. 2024. We are all pattern makers! How a flat ontology connects organizational routines and grand challenges. ***Strategic Organization***, 22(3): 530–549.

Simon, H. A. 1996. ***The sciences of the artificial***. Cambridge, MA: MIT Press.

Smith, A. 1776. ***An inquiry into the nature and causes of the wealth of nations***. London: Strahan & Cadell.

Spee, P., Jarzabkowski, P., & Smets, M. 2016. The influence of routine interdependence and skillful accomplishment on the coordination of standardizing and customizing. ***Organization Science***, 27(3): 759–781.

Sydow, J., Lindkvist, L., & DeFillippi, R. 2004. Project-based organizations, embeddedness and repositories of knowledge. ***Organization Studies***, 25(9): 1475–1489.

Thompson, J. D. 1967. ***Organizations in action: Social science bases of administrative theory***. New York: McGraw-Hill.

Tsoukas, H. & Chia, R. 2002. On organizational becoming: Rethinking organizational change. ***Organization Science***, 13(5): 567–582.

Turner, S. F. & Rindova, V. 2012. A balancing act: How organizations pursue consistency in routine functioning in the face of ongoing change. ***Organization Science***, 23(1): 24–46.

Van de Ven, A. H., Delbecq, A. L., & Koenig Jr, R. 1976. Determinants of coordination modes within organizations. ***American Sociological Review***, 2(41): 322–338.

Victor, B. & Blackburn, R. S. 1987. Interdependence: An alternative conceptualization. ***Academy of Management Review***, 12(3): 486–498.

Wegener, F. & Glaser, V. L. 2021. Design and routine dynamics. In M. S. Feldman, B. T. Pentland, L. D'Adderio, K. Dittrich, C. Rerup, & D. Seidl (Eds.), ***Cambridge handbook of routine dynamics***: 301–314. Cambridge: Cambridge University Press.

Wiedner, R., Barrett, M., & Oborn, E. 2017. The emergence of change in unexpected places: Resourcing across organizational practices in strategic change. ***Academy of Management Journal***, 60(3): 823–854.

Wiedner, R. & Mantere, S. 2019. Cutting the cord: Mutual respect, organizational autonomy, and independence in organizational separation processes. ***Administrative Science Quarterly***, 64(3): 659–693.

Wren, D. A. 1967. Interface and interorganizational coordination. ***Academy of Management Journal***, 10(1): 69–81.

Yamauchi, Y. & Hiramoto, T. 2020. Performative achievement of routine recognizability: An analysis of order taking routines at sushi bars. ***Journal of Management Studies***, 57(8): 1610–1642.

TABLES AND FIGURES

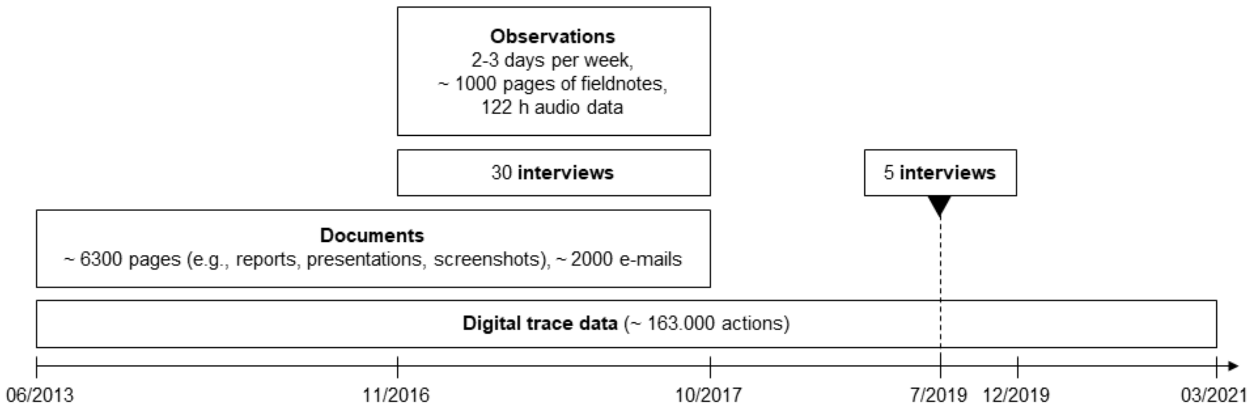
TABLE 1
Overview of Informants and Formal Interviews

Unit	Roles	Pseudonym	Interview dates (Duration in minutes)
Interviews relevant for the interface design process			
Team Alpha	Product Owner	Andrew	21.11.2016 (85 min), 13.12.2016 (29 min), 03.07.2017 (58 min), 28.08.2017 (55 min), 10.10.2017 (44 min), 17.07.2019 (21 min)
Team Alpha	Developer & Scrum Master	Alex	28.11.2016 (84 min)
Team Alpha	Developer		08.02.2017 (75 min)
Team Alpha	Developer, specialized in testing		24.01.2017 (105 min), 17.07.2019 (60 min)
Team Alpha	Developer		30.01.2017 (53 min)
Team Alpha	Developer, specialized in testing		31.01.2017 (79 min)
Teams Alpha & Beta	Developer, Team Alpha & Scrum Master, Team Beta	Austin-Blake	13.12.2016 (90 min)
Team Beta	Product Owner	Brandon	29.11.2016 (87 min), 17.07.2019 (75 min)
Team Beta	Developer		22.03.2017 (102 min), 17.07.2019 (48 min)
Team Beta	Developer		27.06.2017 (74 min)
Team Beta	Developer		21.02.2017 (71 min)
Interviews related to the broader context			
Department ‘base technology’	Department manager		22.11.2016 (30 min), 17.07.2019 (32 min)
Other software development teams	Team managers, developers, internal clients		14.02.2017 (62 min), 20.02.2017 (64 min), 27.02.2017 (38 min), 01.08.2017 (56 min)
Machine development project	Project manager, developers		01.08.2017 (38 min), 14.08.2017 (44 min), 14.08.2017 (40 min), 15.08.2017 (39 min), 15.08.2017 (80 min), 15.08.2017 (48 min), 29.08.2017 (56 min), 11.09.2017 (50 min), 20.09.2017 (52 min), 08.12.2017 (30 min)

TABLE 2
Practices of Interface Design Work

		Durability of interface design work	
		Transitory work Work aimed at solving situated problems that emerge in the process of designing interfaces	**Patterning work** Work that stabilizes interfaces
Design elements	**Artifacts** Designing divisible artefacts used at interfaces	**Analyzing interdependencies** • *Definition*: unpacking and describing resource-related, pooled interdependencies across routine clusters to inform further interface design work • *Addressed interdependence type:* pooled interdependencies • *Consequences:* may render additional pooled interdependencies salient	**Reconfiguring resources** • *Definition*: making (relatively) sustainable changes to shared, scarce resources to make them available across clusters • *Addressed interdependence type:* pooled interdependencies • *Consequences:* may lead to sequential interdependencies because resources are reconfigured through routine performances, which might conflict with other routine performances
	Timing Designing the timing of actions when resources are indivisible	**Mitigating interferences** • *Definition*: increasing or decreasing the priority of work to account for emergent interdependencies that result from designing interfaces across routine clusters • *Addressed interdependence types:* sequential or reciprocal interdependencies • *Consequences:* may lead to reciprocal interdependencies when time pressure creates the need for multiple routines to be performed simultaneously	**Reconfiguring schedules** • *Definition*: making (relatively) sustainable changes to the timing of patterns of actions to enable smooth coordination of routine performances at the interfaces between routine clusters • *Addressed interdependence types:* pooled, sequential, or reciprocal interdependencies • *Consequences:* may lead to reciprocal interdependencies when time pressure creates the need for multiple routines to be performed simultaneously

FIGURE 1
Overview of the Data



Observations
2-3 days per week,
~ 1000 pages of fieldnotes,
122 h audio data
30 **interviews**
5 **interviews**
Documents
~ 6300 pages (e.g., reports, presentations, screenshots), ~ 2000 e-mails
Digital trace data (~ 163.000 actions)
06/2013
11/2016
10/2017
7/2019
12/2019
03/2021

FIGURE 2
Temporal Organization of the Findings

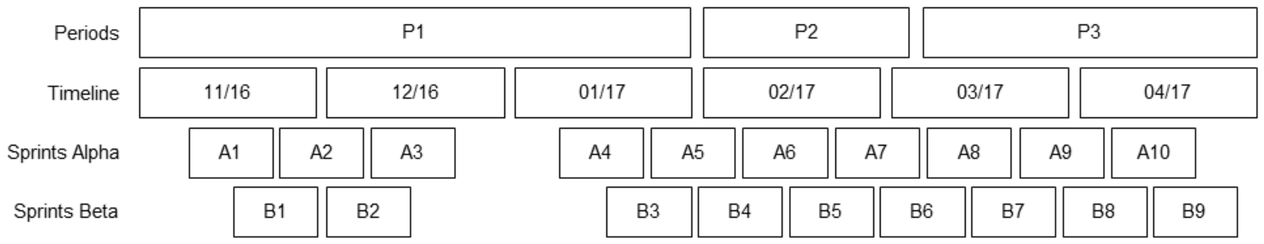
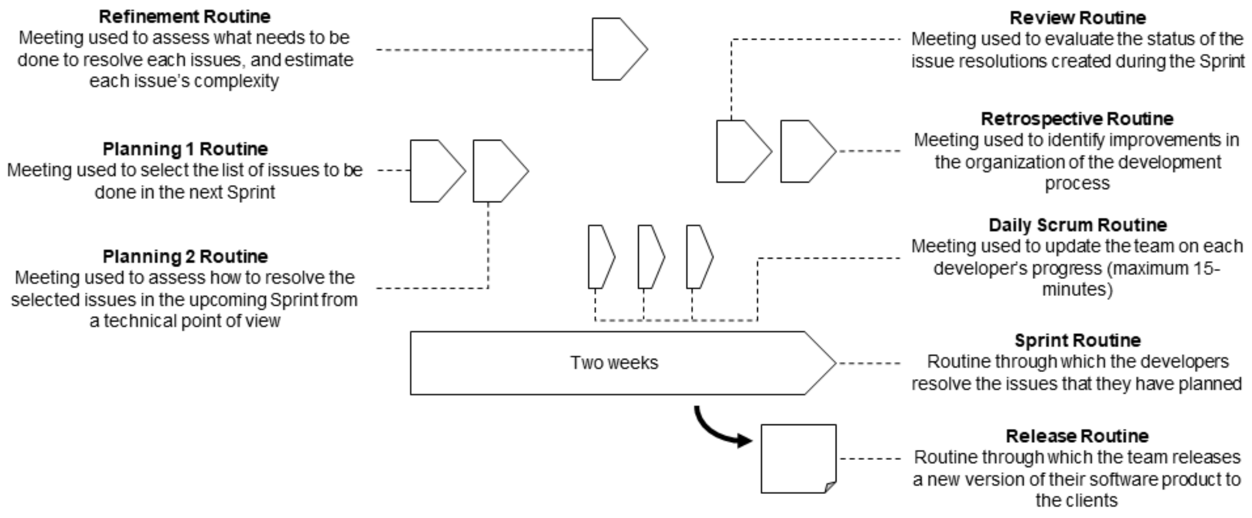


FIGURE 3
Overview of Alpha's Routine Cluster



Refinement Routine
Meeting used to assess what needs to be done to resolve each issues, and estimate each issue's complexity
Planning 1 Routine
Meeting used to select the list of issues to be done in the next Sprint
Planning 2 Routine
Meeting used to assess how to resolve the selected issues in the upcoming Sprint from a technical point of view
Review Routine
Meeting used to evaluate the status of the issue resolutions created during the Sprint
Retrospective Routine
Meeting used to identify improvements in the organization of the development process
Daily Scrum Routine
Meeting used to update the team on each developer's progress (maximum 15-minutes)
Two weeks
Sprint Routine
Routine through which the developers resolve the issues that they have planned
Release Routine
Routine through which the team releases a new version of their software product to the clients

FIGURE 4
Analytical Summary of the Findings

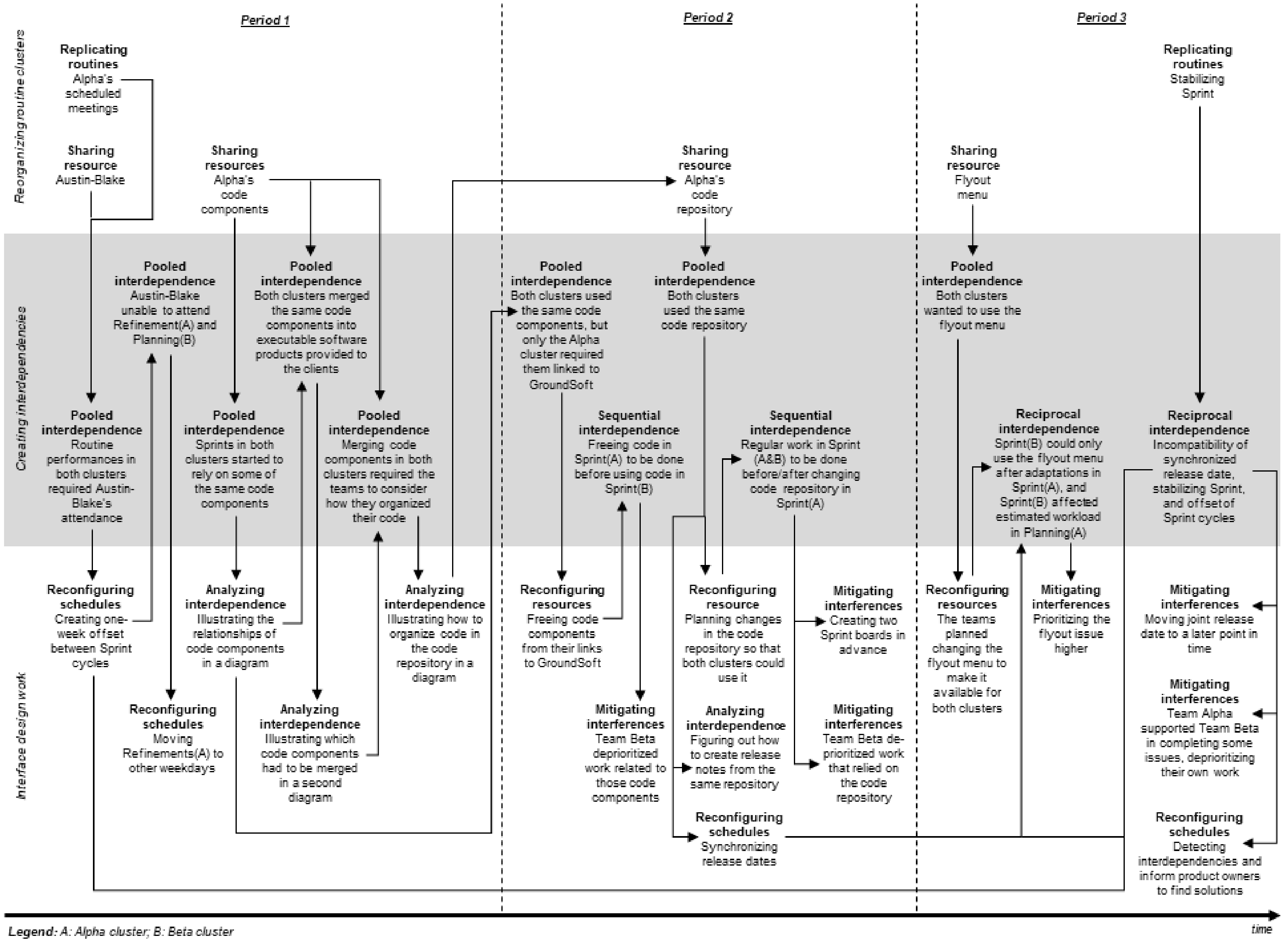
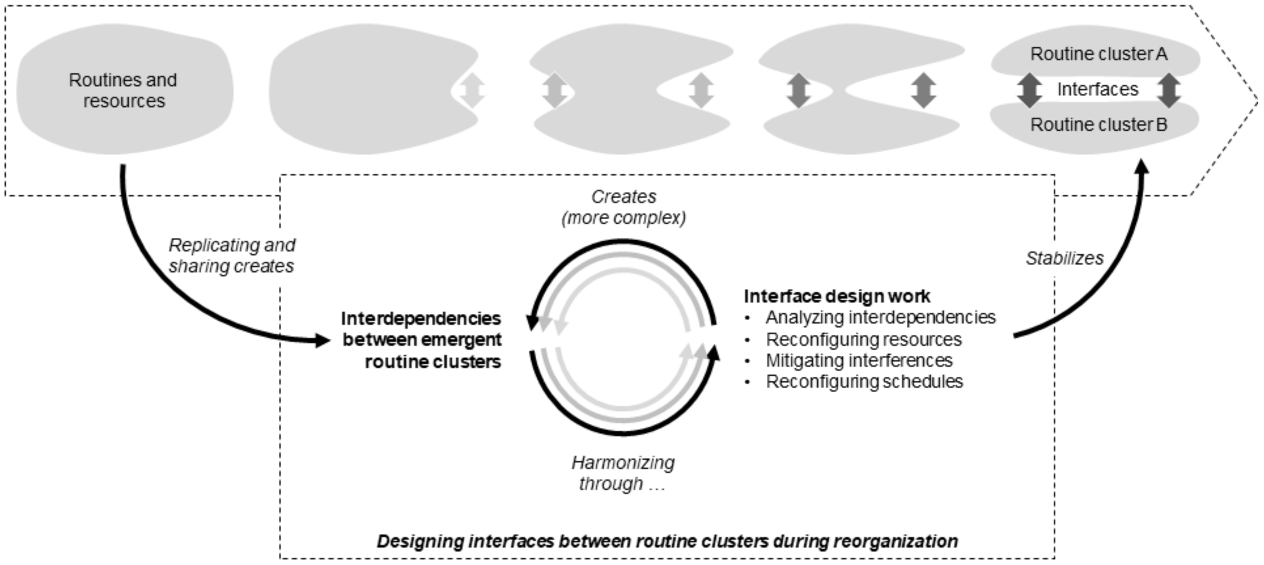

Period 1
Period 2
Period 3
Reorganizing routine clusters
Creating interdependencies
Interface design work
Replicating routines Alpha's scheduled meetings
Sharing resource Austin-Blake
Sharing resources Alpha's code components
Sharing resource Alpha's code repository
Sharing resource Flyout menu
Replicating routines Stabilizing Sprint
Pooled interdependence Austin-Blake unable to attend Refinement(A) and Planning(B)
Pooled interdependence Both clusters merged the same code components into executable software products provided to the clients
Pooled interdependence Both clusters used the same code components, but only the Alpha cluster required them linked to GroundSoft
Pooled interdependence Both clusters used the same code repository
Pooled interdependence Both clusters wanted to use the flyout menu
Pooled interdependence Routine performances in both clusters required Austin-Blake's attendance
Pooled interdependence Sprints in both clusters started to rely on some of the same code components
Pooled interdependence Merging code components in both clusters required the teams to consider how they organized their code
Sequential interdependence Freeing code in Sprint(A) to be done before using code in Sprint(B)
Sequential interdependence Regular work in Sprint (A&B) to be done before/after changing code repository in Sprint(A)
Reciprocal interdependence Sprint(B) could only use the flyout menu after adaptations in Sprint(A), and Sprint(B) affected estimated workload in Planning(A)
Reciprocal interdependence Incompatibility of synchronized release date, stabilizing Sprint, and offset of Sprint cycles
Reconfiguring schedules Creating one-week offset between Sprint cycles
Reconfiguring schedules Moving Refinements(A) to other weekdays
Analyzing interdependence Illustrating the relationships of code components in a diagram
Analyzing interdependence Illustrating which code components had to be merged in a second diagram
Analyzing interdependence Illustrating how to organize code in the code repository in a diagram
Reconfiguring resources Freeing code components from their links to GroundSoft
Mitigating interferences Team Beta deprioritized work related to those code components
Reconfiguring resource Planning changes in the code repository so that both clusters could use it
Analyzing interdependence Figuring out how to create release notes from the same repository
Reconfiguring schedules Synchronizing release dates
Mitigating interferences Creating two Sprint boards in advance
Mitigating interferences Team Beta de-prioritized work that relied on the code repository
Reconfiguring resources The teams planned changing the flyout menu to make it available for both clusters
Mitigating interferences Prioritizing the flyout issue higher
Mitigating interferences Moving joint release date to a later point in time
Mitigating interferences Team Alpha supported Team Beta in completing some issues, deprioritizing their own work
Reconfiguring schedules Detecting interdependencies and inform product owners to find solutions
time
Legend: A: Alpha cluster; B: Beta cluster

FIGURE 5
Process Model of Designing Interfaces


Routines and resources
Routine cluster A
Interfaces
Routine cluster B
Replicating and sharing creates
Creates (more complex)
Interdependencies between emergent routine clusters
Interface design work
• Analyzing interdependencies
• Reconfiguring resources
• Mitigating interferences
• Reconfiguring schedules
Harmonizing through …
Stabilizes
Designing interfaces between routine clusters during reorganization

APPENDIX A: EXEMPLARY INTERVIEW PROTOCOL

1. **Introduction**
   1.1. Overview of the aims of the project.
   1.2. Indication that interviews are treated anonymously.
   1.3. Asking for consent to audio-tape interview.

2. **General assessment of agile development**
   2.1. Which aspects of agile development work particularly well in your team?
   2.2. Which aspects of agile development could be improved in your team?
   2.3. How did the implementation of agile development change how you work? Which aspects did you experience as unfamiliar?

3. **Routines in agile software development**
   3.1. Please describe the major steps of your software development process.
   3.2. How do you label step A?
   3.3. What exactly do you do in step A?
   3.4. Who is involved in step A?
   3.5. What are the inputs and outputs of step A?

   Questions 3.2 to 3.5 are repeated for each step.

4. **Questions related to specific events**
   4.1. Who defined the issues for the BetaSoft project?
   4.2. What is the 'deepzoom issue' about? Why is it difficult to estimate story points for this issue?
   4.3. What is the value of bugfix issues? Since when does your team use bugfix issues?

   Questions 4.1 to 4.3 are examples; these questions were tailored to each interviewee.

APPENDIX B: IDEALIZED DESCRIPTIONS OF ALPHA'S ROUTINE CLUSTER

Routine	Pattern of actions	Actors
Sprint	The developers sit at their computer terminals in their office space. Each developer selects an issue from the Sprint backlog, self-assigns to the issue in Jira, and marks it as 'in progress'. He/she copies the code file on a computer terminal, modifies the code, and uploads the changes into the collective code repository in 'Git'. He or she may ask colleagues for help if needed. The developer sets the status of the issue to 'resolved' and removes herself/himself as an assignee. Then, another developer checks and tests every set of changes, either manually or via automated tests. If the submitted code fails the tests, it is rejected and the developer who created or modified it has to revise and improve it until it passes all tests. If the code passes the tests, the testing developer approves the changes and merges the code into the 'master' code (i.e., the code that will eventually constitute the next release of the software) to update it.	Developers, Product Owner, Scrum Master
Refinement	The team regularly meets to refine issues (i.e., three times in two weeks; timeboxed to one hour). The Product Owner presents to the team an issue for refinement, which he has selected prior to the meeting. The team then clarifies and documents what needs to be done to satisfy the users' requirements (e.g., they extend the documentation, create subtasks), which influences the complexity of solving the issue. The developers, then, estimate 'story points,' which indicate the complexity of an issue. The Product Owner adds the story point estimate and the 'refined' tag to the issue in Jira. The team proceeds with the next issue, until the Refinement meeting is over.	Developers, Product Owner, Scrum Master
Planning 1	The team meets every second Tuesday morning to plan the upcoming Sprint. Before the meeting, the Product Owner creates a digital Sprint backlog as well as issues for unplanned bug fixes, code review, Scrum improvements and unplanned tasks. He also calculates the developers' capacity for the upcoming Sprint in Jira. When the team meets, the developers first decide how many story points they will complete in the upcoming Sprint, and how many story points they reserve for potential bugs (i.e., bugfix issue). Then, the Product Owner selects an issue from the product backlog, according to priority, for the developers to resolve in the upcoming Sprint. The Product Owner presents the issue and, if necessary, specific points in the Product Owner's presentation are clarified. The team defines whether the developers will be able to resolve the issue in the upcoming Sprint and drags and drops it into the digital Sprint backlog. The team compares the intended number of story points defined for a Sprint with the total story points in the Sprint backlog and stops when the numbers match. Finally, the Product Owner asks the developers for their commitment to the composition of the Sprint backlog, and the team defines a meaningful name for the Sprint.	Developers, Product Owner, Scrum Master
Planning 2	The developers meet every second Tuesday afternoon for Planning 2. First, the team assigns organizing the 'code review' to a developer. The developers then select an issue from the Sprint backlog. They clarify how to resolve the issue in the Sprint from a technical point of view and split each issue in the backlog into smaller subtasks and document them in Jira. The developers, finally, start the Sprint.	Developers, Scrum Master
Daily Scrum	During each Sprint, the team meets every morning for the Daily Scrum. During a Daily Scrum, each developer updates the team on his or her progress since the last Daily Scrum and on what he or she plans to work on until the next Daily Scrum, and on any problems he or she has encountered. The Daily Scrum is complete when all developers have updated the team.	Developers, Product Owner, Scrum Master
Review	In each Sprint, the developers meet on the Friday of the second week to prepare a presentation summarizing the Sprint for the entire team. The team, then, meets on the Monday morning of the second week for their Review. The developers present the results of the Sprint. The Product Owner checks whether the resolved issues meet the acceptance criteria and if they have, closes the issue.	Developers, Product Owner, Scrum Master
Retro-spective	In each Sprint, the team meets on the Monday afternoon of the second week for their scheduled Retrospective meeting. The team then reflects on positive, neutral and	Developers, Scrum Master

	negative aspects of the completed Sprint and notes these aspects on cards that they pin on a whiteboard. Each developer presents a card that she or he has pinned on the whiteboard, and the developers discuss, clarify and evaluate the aspect of the Sprint the cards concern. The Scrum Master creates a summary of the discussion in an issue in Jira. The process is repeated until all pinned cards have been discussed.	
Release	The Product Owner, in agreement with the developers, defines a release date (ca. 3-4 times per year), and they define which issue resolutions the release needs to include. When the code is ready, the developers create an executable file to be provided to the clients. The developers also create release notes that describe which issue resolutions the release includes.	Developers, Product Owner, clients

AUTHOR BIOGRAPHIES

Christian A. Mahringer is a postdoctoral researcher at the University of Stuttgart School of Management and a project leader at the WIN Kolleg of the Heidelberg Academy of Sciences and Humanities. He earned his doctoral degree from the University of Stuttgart, Germany. His research focuses on: managing organizational change and cultivating agility, organizing with digital technologies, organizing for grand challenges, and methodological opportunities of digital trace data.

Anja Danner-Schröder is an associate professor for Management Studies at the RPTU Kaiserslautern, Germany. She received her PhD from the University of Hamburg, Germany. Her current research focuses on organizational routines, agile management, and grand challenges.

ACKNOWLEDGEMENTS

We are grateful for the excellent guidance of deputy editor Ann Langley and three anonymous reviewers. We also thank Katharina Dittrich, Daniel Gäckle, Waldemar Kremser, Brian Pentland, Birgit Renzl, Martin Rost, Laura Schmiedle, and Nico Walleser for their feedback on earlier versions of this paper, as well as the members of TIC, who allowed us to gain in-depth insights into their daily routines. This research was funded by the WIN program of the Heidelberg Academy of Sciences and Humanities, financed by the Ministry of Science, Research and the Arts of the State of Baden-Württemberg, Germany. Grammarly was used to check spelling and grammar.