Contents lists available at ScienceDirect

Information and Organization

journal homepage: www.elsevier.com/locate/infoandorg

Integrating development and operations teams: A control approach for DevOps

Anna Wiedemann[a,*], Manuel Wiesche[b], Heiko Gewald[c], Helmut Krcmar[d]

[a] Zurich University of Applied Sciences, Theaterstrasse 17, 8401 Winterthur, Switzerland
[b] Technical University of Dortmund, August-Schmidt-Straße 1, 44227 Dortmund, Germany
[c] Neu-Ulm University of Applied Sciences, Wileystr. 1, 89231 Neu-Ulm, Germany
[d] Technical University of Munich, Boltzmannstraße 3, 85748 Garching, Germany

ARTICLE INFO

Keywords:
DevOps
Control theory
Internal IT teams
Case study
Product management
Tensions

ABSTRACT

Information systems (IS) literature has predominantly studied IS project control with a focus on software development projects. However, by virtue of digital transformation, an increasing number of organizations are implementing cross-functional teams, combining software development with software operations tasks. The goal is to react quickly to the ever-changing market requirements.

The DevOps concept aims to effectively orchestrate development and operations activities and smoothly manage tensions within teams, resulting from the heterogeneous composition of skills, responsibilities, and working styles.

In contrast to the predominant project management view of control of prior research, which focuses on software development, this study investigates a different perspective: focusing on exerting control in DevOps teams and simultaneously navigating tensions between software development and operations. Utilizing an inductive theory-building approach, we first identify the four tensions discussed in prior literature—namely, *goal conflict, method discomfort, decision rights,* and *time rhythm*—and then empirically derive corresponding resolutions.

Integrating our findings, we present an empirically derived model that can serve as a DevOps control approach for navigating the tensions between development and operations teams. This model extends our theoretical knowledge about control in DevOps teams and serves to inform IT practitioners, helping them successfully implement and manage DevOps teams.

1. Introduction

Governance in software development project management remains a key concern of information technology (IT) executives and CIOs (Cram et al., 2016a; Gkeredakis & Constantinides, 2019) and has also motivated research on approaches to managing control and collaboration for governing organizational transformation (Agarwal et al., 2022). An increasing number of organizations are implementing agile software delivery teams to manage the software development process and improve their ability to quickly respond to changing customer demands (Dönmez et al., 2016). Although agile methods are commonly applied to software development projects, the operational aspects of software delivery (such as deploying new software features and continuously delivering stable software in

* Corresponding author.
E-mail addresses: anna.wiedemann@zhaw.ch (A. Wiedemann), manuel.wiesche@tu-dortmund.de (M. Wiesche), heiko.gewald@hs-neu-ulm.de (H. Gewald), helmut.krcmar@tum.de (H. Krcmar).

https://doi.org/10.1016/j.infoandorg.2023.100474
Received 2 January 2019; Received in revised form 24 May 2023; Accepted 1 July 2023
Available online 14 July 2023

production environments) are often incompatible with agile working methods (Hemon-Hildgen et al., 2020). To overcome this deficiency, organizations are beginning to implement cross-functional, "DevOps" (i.e., development and operations) teams. The aim is to achieve the best of both worlds: agile response to constantly changing requirements and stability in software production. Achieving this challenging goal could reduce response times for software delivery (Fitzgerald and Stol, 2017; Rodríguez et al., 2017) and improve customer satisfaction (Wiedemann et al., 2019).

Previous studies have investigated the performance of teams by measuring project success based on how well they achieve time- and budget-related objectives (Choudhury & Sabherwal, 2003; Cram & Newell, 2016). However, this approach is less appropriate in the software delivery lifecycle, where a significant portion of the budget and work effort is spent on post-implementation processes such as operations (Banker et al., 1998; Edberg et al., 2012; Stachour & Collier-Brown, 2009). The IT function within an organization typically strives to achieve customer satisfaction through rapid code deployments and stable software operations. Supporting these high-level goals involves both development and operations functions in terms of functional goals, priorities, and practices. Whereas the key contribution of software development is to quickly provide new functionalities, software operations teams deliver the highest value by ensuring stable software operations for key systems. Attempting to achieve these opposing functional goals (speed vs. stability) in an internal cross-functional team is likely to create tensions (Onita & Dhaliwal, 2011; Shaft & Vessey, 2006). Research and practice do not yet fully understand how and why exactly these tensions occur and lack guidance on how to effectively navigate them.

Many scholars have applied control theory to explain the process of managing project teams (Kirsch, 1997; Maruping et al., 2009). In addition to demonstrating a strong focus on control in the project context, extant research has shown that combining different project control styles can resolve tensions—e.g., balancing bureaucratic and collaborative management styles (Gregory & Keil, 2014) supports ambidexterity between exploration and exploitation (Wiener et al., 2016). While the focus on control in the general project context and in software development project management has led to valuable contributions to research and practice, few studies have applied control theory in other IS-related contexts (Wiener et al., 2016). As a step toward filling this research gap, we identify and develop control mechanisms that can help cross-functional teams navigate and manage the complex interrelationships between development and operations. In this study, first, we describe the tensions identified in the existing research on control theory, software development projects, and software operations. Second, by conducting a qualitative study of DevOps team implementation as an empirical phenomenon (Fitzgerald & Stol, 2017; Sebastian et al., 2017), we identify and further develop control mechanisms that can help IT managers and DevOps teams to navigate these tensions. These efforts are guided by two research questions:

RQ1. What are the major tensions that emerge when development and operations teams are combined?

RQ2. What control mechanisms can successfully navigate the tensions in DevOps teams?

To answer our first research question, we introduce the concept of DevOps and explain the insights derived from prior research on development and operations control. To answer our second research question, we take a grounded theory approach (Gioia et al., 2013; Glaser & Strauss, 1967) to derive control mechanisms as aggregated dimensions of cross-functional DevOps teams that can be applied to resolve tensions between development and operations work. We conclude the paper by discussing the practical implications of our DevOps control model for IT managers and DevOps teams and by identifying the limitations of our study and suggesting avenues for further research.

2. Background literature

In this section, we introduce the DevOps concept and present its theoretical underpinnings. We then discuss IS project control, with a particular focus on software development projects (Wiener et al., 2016) and review IS operations literature e.g., Cram et al. (2016a). Based on our review, we identify tensions that may result when development and operational goals, priorities, and practices are combined.

2.1. The DevOps concept

The term DevOps dates back to 2009 (Debois, 2011) and signifies the integration of development and operations tasks into one team. Traditionally, organizations integrate DevOps teams in an attempt to achieve the continuous delivery of software (Fitzgerald & Stol, 2017). The market research firm Gartner estimates that around 90% of organizations optimize their customer value through implementing DevOps (Haight & Spafford, 2022).

DevOps is defined as a cultural and technological approach to integrating the tasks, knowledge, and skills involved in planning, building, and running activities within a single cross-functional team responsible for one or more digital products (Wiedemann et al., 2019). The goal of DevOps is to facilitate collaboration between development and operations, in part by automating tasks for building and deploying code and testing in order to reduce software development time, enable continuous software delivery, and increase software stability and thus customer satisfaction (Fitzgerald & Stol, 2017).

Prior to the 1990s, large firms organized their IT functions around IS development and operations units (Berente & Yoo, 2012; Hemon-Hildgen et al., 2020) based on the separation of knowledge of these units and activities oriented around different goals and values e.g., the speed of software deployments and the stability of running systems (Cross et al., 1997; Luftman et al., 1993). However, customer satisfaction increasingly became a key goal for firms and the continuously changing environments necessitated new requirements and rapid change. Thus, in the 1990s, agile software development became popular for managing projects focusing on customer satisfaction and business/IS development relationships (Cross et al., 1997; Hemon-Hildgen et al., 2020). However, typical operations activities such as software delivery and deployments were often neglected in agile project management frameworks (Hemon-Hildgen et al., 2020). DevOps goes beyond agile software delivery methods, combining development and operations in an agile way of working (Debois, 2008).

According to Kim et al. (2016), DevOps consists of three main principles. First, the principle of flow refers to close cooperation between development and operations to achieve high quality and end-to-end responsibility for software products (Kim et al., 2016). Second, the principle of feedback refers to the use of agile values and automation to achieve the continuous delivery of software (Kim et al., 2016). Third, the principle of continuous learning and an environment of trust supports a culture of organizational learning from success and failure (Kim et al., 2016).

In traditional software development, such as using the waterfall method for software development, there is a clear separation between activities such as planning, analysis, design, and coding that are necessary precursors to production deployments (Fitzgerald & Stol, 2017). After the development work is done, the operations team is responsible for managing support and maintenance (Edberg et al., 2012). This works very well in stable environments, especially when few changes need to be made to the running software (Cross et al., 1997). In contrast, in rapidly changing environments, IT functions have to coordinate and align changes within their departments (Gkeredakis & Constantinides, 2019) because developers act in increasingly complex and dynamic environments and are expected to continuously deliver new solutions and innovations (Dönmez et al., 2016). Hence, DevOps combines development and operations activities to provide flexibility in rapidly changing environments in order to produce complex business- and safety-critical software (Fitzgerald & Stol, 2017).

Scholars generally agree that close collaboration between the development and operations components of the IT function is necessary to improve software delivery, quality, reliability, and resilience and ensure that errors can be fixed quickly (Hemon et al., 2019). Cross-functional DevOps teams require collaboration among diverse experts to achieve these goals (Young-Hyman, 2017). To maximize these benefits, an increasing number of companies are now implementing specialized DevOps teams rather than supporting siloed IT departments (Fitzgerald & Stol, 2017).

2.2. *Information systems control literature*

There is an ample body of research on the challenges of managing effective teamwork (Maruping et al., 2009), which has primarily employed the lens of control theory (Kirsch, 2004). In software development in particular, control is defined as management's *"attempts to ensure that individuals working on organizational projects act according to an agreed-upon strategy to achieve desired objectives"* (Kirsch, 1996, p. 1). However, further research is needed to understand how control is enacted in cross-functional teams with combined development and operations control.

2.3. *Differences in control themes*

In this study, we build on the theoretical framework presented by Wiener et al. (2016), which is based on the work of Jaworski (1988) and Kirsch (2004), to structure control performance (control effects), antecedents (control choices), and changes (control dynamics). Wiener et al. (2016) view control context as a subtheme in control choices. However, we argue that control context differs significantly between development and operations; hence, we consider control context to be a separate theme. Below, we describe the control themes and the differences between development and operations in detail.

IS research on *control effects* focuses on IS project performance as a consequence of control-mode choices measured in terms of quality and efficiency (Barki et al., 2001; Wiener et al., 2016). Quality refers to product performance (Henderson & Lee, 1992) and the extent to which a project/product fulfills its requirements (Kirsch, 1996). Project quality is commonly measured by software defects per line of code (Ethiraj et al., 2005), change request fulfillments (Herbsleb & Mockus, 2003) response times, and software reliability and maintainability (Wallace et al., 2004; Wiener et al., 2016), whereas project performance efficiency refers to how well a project meets time and budget targets (Keil et al., 2013).

Control choices refer to the antecedents of the specific projects' control portfolio configuration with control modes and control amounts (Wiener et al., 2016). Contextual and organizational factors can influence the choice of IS control (Cram et al., 2016b). Early research on control modes concentrated on hierarchical control relationships of internal projects with four different control mode choices (Kirsch, 1996; Kirsch, 1997): *"behavior observability, outcome measurability, controller's IS knowledge, and controllee's IS knowledge"* (Wiener et al., 2016, p. 746). Control antecedents include the controllers' level of trust in controllees and the complexity of project duties (Remus & Wiener, 2012).

Wiener et al. (2016) define two *control contexts*, namely project and stakeholder contexts. The project context refers to the project size, project tasks, etc. (Kirsch, 1996; Kirsch, 1997), whereas the stakeholder context refers to the characteristics of the controller and controllee and their relationships (Kirsch, 2004). The factors affecting the amount of control and appropriate behavior in the project context include task complexity, task uncertainty, and strategic importance (Remus & Wiener, 2012), and the factors affecting the amount of control and appropriate behavior in the stakeholder context include trust, knowledge, priority, and cultural differences (Kirsch, 2004; Remus & Wiener, 2012).

The literature on *control dynamics* specifically concentrates on the style according to which control modes and amounts are exercised in practice (Choudhury & Sabherwal, 2003; Kirsch, 2004; Remus & Wiener, 2012). Project lifecycle changes lead to control dynamics, reflecting the fact that specific project phases need distinct control portfolio configurations (Kirsch, 2004). Furthermore, adaptive learning processes during projects force control dynamics (Mähring, 2002). Control dynamics of portfolio configurations can be encounter-triggered (Choudhury & Sabherwal, 2003; Wiener et al., 2016). IS projects go through phases of equilibrium states

Table 1
Differences between development and operations by IS control themes.

Theme	Development	Operations
Control effects	- High efficiency and quality project outcome (Barki et al., 2001; Wiener et al., 2016) measured by project completeness in terms of time, budget (Keil et al., 2013), and meeting user requirements (Kirsch, 1996) - Provision of rapid software features (Fitzgerald & Stol, 2017)	- Measurement of SLAs via ticket management tools (e.g., ticket resolution time) (Trusson et al., 2014). - Aiming at the stability of running software due to the reduction of production releases (Kim & Westin, 1988)
Control choices	- Combination of control modes (formal: input, behavior, and output control; and informal: clan and self-control) for project control portfolios (Kirsch, 1997). - Control degree and influence on control style choices (Heumann et al., 2015; Kirsch, 1996). - Choice of project management methods such as agile or waterfall (Cram et al., 2016b)	- Application and measurement of standardized operations processes (ITIL) (Pollard & Cater-Steel, 2009; Trusson et al., 2014) and corresponding governance frameworks (COBIT) (April et al., 2005) - Software availability and end-user control (Banker et al., 1994; Nelson et al., 2000). - Avoidance of software outages due to monitoring (Nelson et al., 2000; Shaft & Vessey, 2006).
Control contexts	- Project and stakeholder context (Kirsch, 1996; Kirsch, 1997; Wiener et al., 2016). - Collaboration in teams using agile project management or waterfall methods (Cram et al., 2016b). - Rapid requirement and priority changes (Maruping et al., 2009).	- End-user and stakeholder-oriented - Organization of work in classic hierarchical department structures/ support centers (Nelson et al., 2000). - Usage of IT service management approaches as a management tool (Trusson et al., 2014).
Control dynamics	- Various rapid changes across project lifecycle phases (Wiener et al., 2016). - Changes in control modes, control amounts (Choudhury & Sabherwal, 2003; Kirsch, 2004), and control styles (Choudhury & Sabherwal, 2003; Gregory et al., 2013; Kirsch, 2004)	- Adoption of new or modified demands in terms of technology and platform changes (Edberg et al., 2012; Nelson et al., 2000). - Seeking quality and stability of software (Hemon-Hildgen et al., 2020)

punctuated by decisive events between these phases, which are called encounters (Gregory et al., 2013; Wiener et al., 2016).

The body of research on operations control is smaller; however, Edberg et al. (2012) show that a large share of software delivery expenditure is dedicated to maintenance and operations. They examine strategies for choosing software maintenance methodologies and controlling software maintenance—for instance, by avoiding separation in the structure of IT functions. In practice, control is especially relevant in terms of the monitoring and control processes for version management of software infrastructure and operations (April et al., 2005). Maturity models and frameworks, such as the control objectives for information and related technologies (COBIT) model, serve as governance and control mechanisms for software operations (ISACA, 2012). Service-level agreements (SLAs) are often implemented when software developers transfer the software to operations for routine issues and services (April et al., 2005).

Table 1 below summarizes the findings of our review of the extant literature on IS control themes focusing on development or operations.

2.4. *Tensions resulting from differences in control themes*

A tension is defined as *"stress, anxiety, discomfort, or tightness in making choices, responding to, and moving forward in organizational situations"* (Putnam et al., 2016, p. 67). Extant research has shown that organizations and their employees encounter tensions resulting from incompatibilities and dilemmas, which typically arise due to differences in individual and/or organizational levels (Ajer et al., 2021).

To confirm that tensions arise due to differences between operations and software, we examined prior literature and identified concrete tensions between development and operations along the control topics that we present in Table 1. Our literature review shows that differences in the control themes, as outlined above, tend to lead to tensions in the DevOps environment. For DevOps teams to operate effectively, such tensions need to be carefully managed. Table 2 summarizes the tensions derived from the extant literature concerning development and operations.

Based on the extant literature on the different approaches of software development and operations, we derived four control tensions: *method discomfort, goal conflict, decision rights, and time rhythm.* These tensions naturally have implications for managing the collaboration between these functions in traditionally organized companies. However, when a company decides to combine both functions into one DevOps function, the tensions need to be very carefully assessed and managed—otherwise, the new structure will not be successful.

Some companies have successfully managed the transition toward a combined DevOps model and have had great success in managing the tensions arising within these departments. To learn from such companies and to derive generalizable strategies for successfully managing tensions in DevOps teams, we conducted a qualitative research study, as described in the following section.

3. Research design

In this section, we describe our research design, which comprises an inductive theory-building approach (Gioia et al., 2013) based on qualitative research. We investigate how IT managers successfully navigated and overcome tensions between the development and operations components of combined DevOps teams. Qualitative research methods are well-suited to analyzing novel phenomena and deriving compelling explanations. In particular, case studies are recommended for conducting research to answer "how" or "why" questions about contemporary issues such as DevOps teams and are well-suited for studying real-life events (Yin, 2018). To investigate our research question, we conducted case studies in six organizations to identify successful mechanisms for managing tensions in DevOps teams. Our underlying research philosophy is interpretive epistemology. Interpretive research observes phenomena in social settings and takes a nondeterministic approach attempting to explore them in their natural setting rather than imposing an a priori understanding (Orlikowski & Baroudi, 1991). While the presented background literature guided our study approach, we focused on the data interpretation to build a new model for DevOps control. Accordingly, we collected primary and secondary qualitative data in an

Table 2
Tensions in software development and operations teams.

Source of Tension	How Tensions Manifest	Literature
	Tension 1: Goal conflict	
Speed vs. *Stability* Although development and operations serve the same overarching goal (to provide the highest customer satisfaction), they experience competing subgoals when working to achieve them. Development is praised for producing new software functionalities quickly, whereas operations values ensuring that software systems and stable and downtimes are infrequent.	High numbers of software changes inherently jeopardize stable production, as every change carries the risk of a system failure. Therefore, operations favor few updates and additions, whereas development seeks exactly the opposite. This obvious goal conflict leads to tensions between both units, as one unit's gain is the other's loss. This tension typically manifests when software deployment schedules are defined in budget/resource meetings.	Edberg et al. (2012); Fitzgerald and Stol (2017); Keil et al. (2013); Maruping et al. (2009); Trusson et al. (2014)
	Tensions 2: Method discomfort	
Agility vs. *Structure* Both functions traditionally have different methodological foundations. Software development follows a project approach—predominantly agile methods, as of late. Operations, in contrast, uses well-tested process methods like ITIL and COBIT to achieve the required stable environment.	The different methodological approaches are a breeding ground for tension. While development heralds a "learn from mistakes" and "fail fast, fail often" culture, operations maintain a "failure is not an option" mindset.	Tarafdar and Tanriverdi (2018); Fitzgerald and Stol (2017); Hemon-Hildgen et al. (2020); Krancher et al. (2018); Väätäjä et al. (2016)
	Tension 3: Decision rights	
Project vs. *Service* Resulting from the different methodologies used (as described above), there are different decision rights and corresponding metrics used in both functions. Development is typically organized in project structures that are actively managed by project managers or agile product owners (PO) who have (comparatively) high degrees of autonomous decision rights and are measured by traditional project management metrics (time, quality, budget). Operations typically works with traditional hierarchical structures led by service managers with comparatively few decision rights. They are typically measured by externally induced metrics like *service level agreements*, which have long durations.	Whenever two management styles differ and the responsible managers are measured by metrics that are not aligned with each other (or even incompatible), tensions arise quickly, as both parties aim to achieve their aims for their work stream. Apart from this obvious potential for conflict, there is also quite often an underlying misunderstanding, if one party is not available (or ignores) the boundaries in decision-making the other party faces. This can lead to a "we against them" attitude, a fertile breeding ground for tensions.	Fitzgerald and Stol (2017); Kirsch et al. (2002); Shaft and Vessey (2006); Wiener et al. (2016) Edberg et al. (2012)
	Tension 4: Time rhythm	
Short Term vs. *Long Term* Software development is project based and typically short-term oriented. Projects are executed and completed, and then the next project begins. Operations, on the other hand, is an ongoing business with a strong long-term orientation. Unless a fundamental change happens, operations are usually not engaged in projects.	Different approaches to the time dimension lead to tensions, as the respective functions are inherently oriented on different time schedules. Development is used to deliver projects quickly, whereas operations seeks to provide long-term stability—leading to distinct tensions. Development might describe operations as "too slow," whereas operation's reply would be "too hectic, not well thought through." These different approaches lead to tensions, especially if one function asserts pressure on the other function without acknowledging the different approach.	Edberg et al. (2012); Kirsch (2004); Trusson et al. (2014)

in-depth field investigation and coded and analyzed the data following the Gioia method (Gioia et al., 2013).

3.1. Data collection

The primary data consisted of a case study comprising 21 semi-structured qualitative interviews (Myers & Newman, 2007) with DevOps team members and managers. The unit of analysis was DevOps teams, which were responsible for the development and operations of one or more software products. Our secondary data was comprised of data published by the firms online, including websites, blogs, and annual reports.

The six firms in our study were selected via theoretical sampling and differed in terms of the number of employees—ranging from 1,000 to more than 100,000 employees. The firms operated in the Retail Food, Retail Store, Retail Non-Food, Banking, Insurance, and Service sectors. All the companies had a significant application of the DevOps concept in their IT organization. Table 3 provides a brief overview of the case study participants. Further information about the cases can be found in Appendix A.

We conducted semi-structured interviews with managerial and technical employees of one DevOps team per firm to identify details relating to the control of development and operations. The interviewees were performed iteratively to gather the most insightful information possible. Questions about software delivery lifecycle details were directed toward team members, while questions about organization, management, and governance were directed toward managers. We asked each informant to explain their respective IT function and their perception of how the DevOps concept was integrated.

The interviews typically lasted around one hour each; most interviews were face-to-face but some were conducted via telephone. We took extensive notes during the interviews; after each interview, we wrote a summary (Urquhart, 2012). All interviews were recorded and transcribed, producing a total of 442 pages of transcripts for use in further analysis.

3.2. Data analysis

From an analytical perspective, we generated theoretical foundations to generalize our research results following the approach presented by Gioia et al. (2013). We sought to identify first- and second-order concepts and develop aggregate dimensions with the corresponding relationships (Gioia et al., 2013; Miles & Huberman, 1994). Appendix B offers coding details and quotes to support our findings. The analysis of the transcripts was guided by the Glaserian grounded theory approach (Glaser & Strauss, 1967) and the inductive analytical approach presented by Gioia et al. (2013). We started the coding process using open coding to derive initial insights using NVIVO software. To derive first-order concepts, we explored the nature of the tensions derived in our review of relevant extant research. We coded statements in line with how control was used in cross-functional teams to manage tensions. In our second-order coding, we iterated between first-order codes, data interpretation, and existing literature to gather insights through different lenses. Our second-level codes served as control mechanisms to address the tensions. Prior research on DevOps and cross-functional teams (Aime et al., 2014), theoretical approaches from control theory (Kirsch, 2004; Wiener et al., 2019), and IS development and project management theory (Lee & Xia, 2010; Matook et al., 2016) were the key theoretical concepts guiding our data analysis. With the help of grounded theory articulation, we formulated dynamic relationships and made several refinement passes guided by the extant literature to define the final aggregated control dimensions (Gioia et al., 2013).

Several preconditions guided our investigation of control in the DevOps teams. As presented above, prior literature has revealed tensions between software development and software operations control perspectives. We argue that these tensions can be managed using control mechanisms. We derived three dimensions of DevOps control that represent forms of control practices relevant to DevOps

Table 3
Investigated teams.

Case	Company Description	DevOps Team	Roles (No. of Interviews)
Retail Food	Company with a focus on groceries and stores in Germany and across Europe with over 100,000 employees.	PO, agile coach, five team members	Team members (4) PO (1) Agile coach (1)
Banking	One of the largest German private financial services organizations with over 25,000 employees.	Group manager, fifteen team members	Former group manager (1) Current group manager (1) Team members (2)
Insurance	One of the largest Germany-based insurance companies with over 100,000 employees.	Head of architecture, PO, team lead, eight team members	Head of Architecture (1) Group lead (1) Team lead (1)
Service	A German internet company with over 1,000 employees that offers services to end users.	Eight team members, team lead	Director IT (1) Team leads (2)
Retail Store	A German store specializing in food and non-food products with over 50,000 employees.	Team lead, PO, seven team members	Team lead (1) Team member (1)
Retail Non-Food	One of the largest German department store chains with over 20,000 employees.	Team lead/PO, six team members	Team lead (1) Team members (2)
		Total number of interviews	21

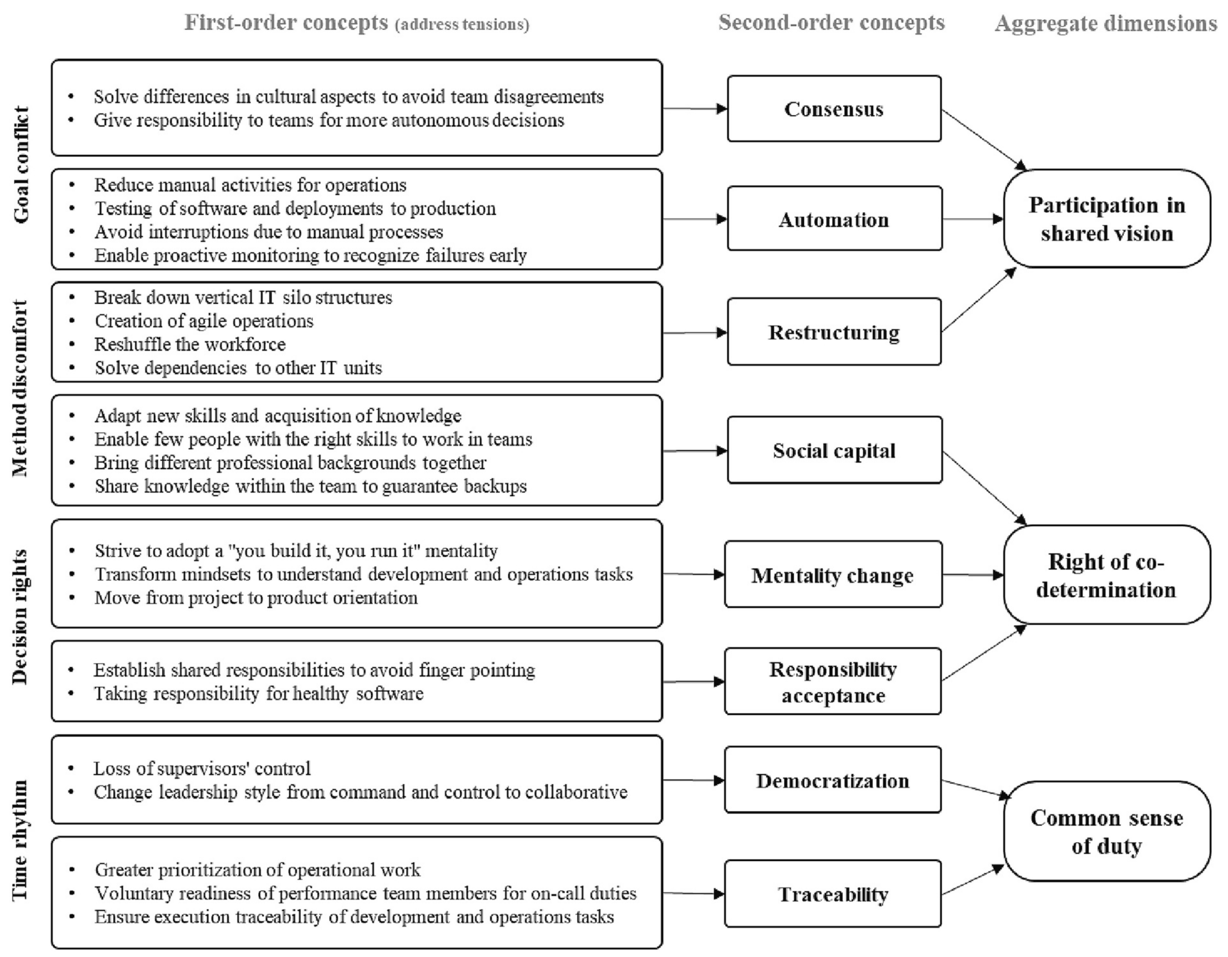
First-order concepts (address tensions)
Second-order concepts
Aggregate dimensions
Goal conflict
Method discomfort
Decision rights
Time rhythm
Solve differences in cultural aspects to avoid team disagreements
Give responsibility to teams for more autonomous decisions
Reduce manual activities for operations
Testing of software and deployments to production
Avoid interruptions due to manual processes
Enable proactive monitoring to recognize failures early
Break down vertical IT silo structures
Creation of agile operations
Reshuffle the workforce
Solve dependencies to other IT units
Adapt new skills and acquisition of knowledge
Enable few people with the right skills to work in teams
Bring different professional backgrounds together
Share knowledge within the team to guarantee backups
Strive to adopt a "you build it, you run it" mentality
Transform mindsets to understand development and operations tasks
Move from project to product orientation
Establish shared responsibilities to avoid finger pointing
Taking responsibility for healthy software
Loss of supervisors' control
Change leadership style from command and control to collaborative
Greater prioritization of operational work
Voluntary readiness of performance team members for on-call duties
Ensure execution traceability of development and operations tasks
Consensus
Automation
Restructuring
Social capital
Mentality change
Responsibility acceptance
Democratization
Traceability
Participation in shared vision
Right of co-determination
Common sense of duty

Fig. 1. Data structure.

teams. Fig. 1 presents our data structure and visualizes the first-order concepts, second-order concepts, and aggregate dimensions. The tables in Appendix B provide further details of how we derived the aggregate dimensions.

4. Mitigating tensions in DevOps teams

In this section, we describe our findings and how tensions can be mitigated by implementing DevOps control mechanisms and dimensions. Table 4 presents and overview of the key mechanisms (aggregate dimensions) and offers insights into the second-order concepts. Deeper elaboration on the dimensions is provided in the sections following the table.

4.1. Participation in shared vision

The dimension *participation in shared vision* describes how leadership can drive the performance of DevOps teams, which can help to resolve the tension of ***goal conflict*** through *consensus* and *automation* and the tension of ***method discomfort*** through *restructuring*. Below, we explain the dimension and second-order mechanisms and provide illustrative examples from our case study. Table 6 in Appendix B provides an overview of detailed coding examples for *participation in shared vision* with the respective first-order concepts and second-order concepts as well as quotations from interviewees.

Consensus: One way that working toward a shared vision can help to resolve tensions within DevOps teams is by using team consensus to mitigate the tension of ***goal conflict***, which is a tension between speed and stability in DevOps teams. Whereas development typically prioritizes speed, operations values stability. To resolve this conflict, DevOps can seek consensus regarding the timing and execution of deployments and how impacts on stability will be managed. In our Retail Food case, the product owner and agile coach worked together to enable and motivate the team to take over responsibility and for making common software decisions. Thus, the team is jointly responsible for product-relevant decisions. For example, if a problem arises after the team conducted a software deployment, the team also solves it together:

Table 4
Key dimensions and mechanisms that mitigate tensions in DevOps teams.

Dimensions	Description	Associated Tensions	Key mechanisms and explanations
Participation in shared vision	DevOps teams develop and enact a common view for a collaborative-working style.	Goal conflict	**Consensus:** DevOps teams assume responsibility for the product and work toward a common goal through coaching.
		Method discomfort	**Automation:** Resolving manual activities and automating DevOps processes such as testing and deploying updates. **Restructuring:** DevOps teams are set up and integrated with established IT functions.
Right of co-determination	DevOps teams determine how to assume responsibility and navigate joint decision rights for the software delivery lifecycle.	Method discomfort	**Social capital:** Depending on the product, DevOps teams are responsible for the necessary development and operations skills that must be learned within the team.
		Decision rights	**Mentality change:** DevOps team members work in teams with end-to-end responsibility for the software delivery lifecycle. **Accepting responsibility:** DevOps team members with different backgrounds are responsible for software quality and for managing all software-relevant problems.
Common sense of duty	DevOps teams achieve ownership through the adaption of the leadership style and transparency for all activities.	Time rhythm stress	**Democratization:** A DevOps leadership style that moves from command-and-control to a more collaborative control approach. **Traceability:** The DevOps team's understanding and assimilation of activities from both development and operations.

Technology decisions can be made autonomously by the team. [...] The responsibilities for the topics are specified in such a way that a team is enabled to make decisions completely independently (Retail Food, Product Owner). We would like to work together on a product in a goal-oriented way (Retail Food, Agile Coach).

In the Retail Food case, the tension of *goal conflict* was mitigated through consensus building; the DevOps team is aiming speed and stability for the software and find consensus for their product-relevant decisions. The case improved its culture of responsibility and worked toward a common goal. DevOps team members assumed responsibility for both development and operations goals.

Automation: The automation of tasks, e.g., test automation and code deployment, facilitates the work in DevOps teams and can help to resolve the tension of *goal conflict* by allowing teams to save time and money or facilitate better effect control (as in the Insurance case), thereby also ameliorating the conflict between speed and stability. For example, DevOps teams can implement different continuous processes to automate software deployments or, as in the Insurance case, push new software code into the production environment with the help of automatic control acceptance steps. The head of architecture of the Insurance firm explained the advantages of automation:

We want to reduce the manual activities that people do when operating the platform [...]. Our results show how automation can reduce the time between recovery. It helps to prevent incidents and improves the quality of the deployment because we no longer operate any kind of deployment manually.

Through automation, the Insurance firm measurably improved both efficiency and software quality. In the Retail Food case, automation helped the DevOps team because they received automated alerts if the planned code deployment was not working during the test phase so that developers could fix the issue before pushing the code to the production environment.

It [the software update] is built, is tested, and goes to the test environment. Then it's also tested again with other services and automatically goes into production. If it wasn't good [...] the developer gets a message during testing, this is broken. What you built was not functional. (Retail Food, Team member).

Automation can mitigate the tension of *goal conflict,* for example, by reducing the need for developers to manually push their new code into production or by automating the detection of software bugs, thereby reducing the effort needed from operations experts to ensure the security of production deployments. These efficiencies can help to resolve the tension of *goal conflict* by allowing DevOps teams to develop new software functionalities more quickly in a secure operational environment.

Restructuring: The tension of *method discomfort* is a tension between *agility* and *structure.* Whereas development generally relies on a project approach, typically using agile methods, operations use well-tested methods such as ITIL and COBIT to ensure a stable structure. Our findings suggest that restructuring within DevOps teams can ameliorate the tension of *method discomfort.* In the words of the Retail Store DevOps team lead:

We're going to dissolve this organizational silo [...] then establish product teams that are no longer assigned to an organizational unit.

Since DevOps is working as a team instead of in separated IT units, the members can decide which best-practice and agile approaches to combine to achieve the best of both worlds. However, this transforms the classic job roles of operations and development. For example, operations must learn to work with agile methods and development must learn to deal with end-user problems such as tickets. DevOps team members can thus learn from other team members to devise new ways of working. For example, as explained by the head of architecture in the Insurance firm, the DevOps team combines operations control choices such as ticket resolution with agile development methods:

We basically try to structure our work so there is also capacity for tickets say 20 h a week. […] We are also contributing to propose new ways to adapt our agile performance management processes in order to fully embrace the DevOps principles. So, ensure that regardless of the fact that people are belonging in two different pillars they can have still the same targets aligned to a product.

Thus, DevOps team restructuring can allow teams to combine both agile methods (development) and best-practice approaches (operations) to navigate the tension of *method discomfort.*

4.2. Right of co-determination

The *right of co-determination* dimension relates to the ***method discomfort*** and ***decision rights*** tensions. Appendix B Table 7 provides more detailed findings and further illustrative quotations. Our findings indicate that these tensions can be mitigated by *social capital, mentality change,* and *accepting responsibility,* which we explain in more detail below.

Social capital: Our findings show that DevOps teams develop a high level of shared social capital (communication, knowledge, and trust) through the acquisition of new skills, which can help bridge the different cultural mindsets of development and operations and mitigate the *method discomfort* tension, which involves a tension between agility and structure. In our service case, the firm delivered a very successful internet platform with a huge number of digital services to its customers. In this case, DevOps team leaders had complete responsibility for everything that happened in the team and for performing all relevant tasks for the internet platform service, which required an immense training effort. The team leads can represent each other. In addition, the team leader must train the team members to develop new skills and acquire new knowledge within the team to ensure a backup option in future:

The team leader is the [developer and] system administrator for his application—it plays a dual role. In my opinion, this is classic DevOps. […] Each of these team leaders can represent each other's team (Service, Director IT).

Another case in the Retail Non-Food area indicates that team members were expanding their knowledge—rather than being a specialist in one area (e.g., either development or operations) they became generally competent in both areas. Hence, team members helped each other develop general knowledge, which is necessary to manage the software delivery lifecycle and overcome cultural obstacles that appear due to different professional backgrounds. For example, in the Retail Non-Food case, the team was responsible for building and running a platform and the team members mainly had operations backgrounds. Hence, as one Retail Non-Food team member explained:

The hardest thing for me personally is that I was never a programmer, and, with some concepts, I still have a bit of a hard time learning it all now.

DevOps team members can thus mitigate the *method discomfort* by gaining expertise in both development and operations. Team members gradually develop a common skill set by working together, which reduces the strain between using best-practice versus agility approaches. All team members have the same focus (the product) which leads to a common control choice.

Mentality change: Our research suggests that DevOps teams can mitigate the *decision rights* tension, which involves a project vs. service conflict, through a mentality change that enables collaboration and communication across different cultural and professional backgrounds. Importantly, DevOps teams are responsible for managing different management styles (project vs. service management) and control contexts. The mentality change can reduce management style conflicts. For example, because code deployments can be executed without downtimes by developers within the DevOps team, service managers need to worry less about downtimes that could potentially affect software stability:

Tasks are appropriately small, but we have much better control over what happens within the release (Service, Team Lead). You can migrate with the "snap of a finger" and therefore we need no downtime (Service, Director IT).

Likewise, classic project management control becomes less important because DevOps teams work in a product-oriented setup concentrating on the efficient management of the software delivery lifecycle instead of focusing on projects with strict budgetary and time constraints. In the words of a Retail Store team lead:

For me, that's where the product differs from the project. A product runs until sundown until it is no longer needed.

Since DevOps teams focus on the software delivery lifecycle, DevOps teams are managed differently from project teams. In traditional teams, *method discomfort* (agility vs. structure) and *decision rights* (project vs. service) tensions occurred because once development completed their software project, the software was handed over to operations for support and maintenance. In DevOps teams, the product can be continuously developed and supported: DevOps employees work together on the data and the methodical procedure is defined together for the team. Hence, the mentality change facilitated by DevOps teams can alleviate the *method discomfort* and *decision rights* tensions.

Accepting responsibility: In DevOps teams, the conflict between project and service characterizing the *decision rights* tension can also be mitigated by accepting responsibility. Our Retail Food and Banking cases showed that cross-functional working styles are helpful for DevOps teams to organize their work as one common team. The Retail Food firm was developing a mobile app for customers but also mobile hardware for delivery personnel. Ideally, new orders or order changes would be visible to delivery personnel as quickly as possible. While mobile apps in the Retail Food case are developed by DevOps teams, great effort is necessary to integrate cross-functional teams:

> *We would like to work together […] on the product and thus we need to manage our culture more consistently* (Retail Food, Agile Coach).

Often, several development teams work on a common infrastructure (e.g., a cloud platform). Sometimes, problems occur and cannot be located. Hence, it is often unclear who is responsible for this problem. As a team member from Retail Food highlighted:

> *The biggest challenge now is making sure that the entire platform is in a healthy state. So, this topic: "How do I localize when a problem occurs where this problem has its cause?"*

In DevOps teams, the responsibility for problems is accepted by the team as a whole rather than any particular component of the team being blamed. In the Banking case, the firm was working to improve the sense of responsibility and combining the power of development and operations in DevOps teams. IT functions with separate units for development and operations often suffer from a problem of managers blaming the other unit when problems occur. The former group lead of Banking stated the need to

> *Get people to feel end-to-end collectively responsible for lead time, both ways, and then stop pointing fingers at each other.*

DevOps teams navigate the decision rights problem by *accepting responsibility* at the team level. For example, as the Banking case indicates, if a problem occurs, the team needs to identify the cause and fix it. If several DevOps teams are working on the same platform, every team must be responsible for identifying the cause of problems.

4.3. Common sense of duty

The *common sense of duty* dimension describes how the team overcomes deeply ingrained work attitudes by integrating a common mindset and a team spirit with a willingness to learn and cooperate. Appendix B Table 8 provides an overview of coding examples for *common sense of duty* with the respective first-order concepts and second-order concepts as well as quotations from our case study. Our cases indicate that *democratization* and *traceability* help DevOps teams and managers navigate the ***time rhythm*** tension, which is related to a conflict between short-term and long-term orientation.

Democratization: The Banking and Insurance cases emphasized the need to democratize their leadership style to guide and enable the DevOps teams, which is especially important for large firms. In DevOps teams, democratization can reduce the stresses associated with the *time rhythm* tension because development is no longer managed in projects with rapid lifecycle changes whereas the control dynamics appear frequently. Operations used to work in stable IT units whereas control dynamics occur in the form of punctuated equilibrium. These have long stable control phases and rarely control changes. The adoption of a more democratic leadership style in DevOps teams is a possible solution for this tension. Our Insurance case, is a huge firm with global reach. Their IT function was previously organized by many development teams independently working on projects. These projects were supported by a separated operations unit organized in silos within their IT function. More recently, they implemented DevOps teams and reduced the pressure and dependencies between the different units of development and operations. This was made possible by leaders who were trained to be less directive and more collaborative by enabling the DevOps team to find a common way to align the tasks in terms of time:

> *The world evolved to a more, to a faster environment, supported by technology of course, to develop things in a faster way […]. This also changes the leadership style from directive to more collaborative and consensual (Insurance, Head of Architecture).*

Hence, through the democratization of control, DevOps teams can mitigate the *time rhythm* tension. Different time rhythms are aligned in DevOps teams and are oriented around the decisions and dynamics associated with the software delivery lifecycle for which DevOps teams are responsible.

Traceability: The time rhythm tension can also be mitigated by the high level of traceability of activities in DevOps teams. Our findings indicate that both development and operations need to understand how, when, and by whom activities are performed. In traditional IT functions, development focuses on its own plan, and if operations problems occur, it may take a long time before they are prioritized. DevOps teams can ameliorate this tension because team members develop a common sense of duty for all product-relevant tasks. The Retail Store case integrated DevOps teams, and they visualized all tasks related to development and operations. This has released the *time rhythm* tension by giving higher priority to the often long-term orientated operations tasks. As a result, operational tasks are processed more quickly:

> *Operation[s] has become a much higher priority with all the issues. The whole monitoring issue is something I notice because the quality of many topics is getting better and better every day, whenever something pops up* (Retail Store, Team Member).

In the Retail Non-Food case, the DevOps journey began from an operational perspective. Originally, the DevOps team members were operations employees with a strong focus on system administration tasks. They started to implement an online platform for digital services. The platform team recognized the need for coding and customizing and also had to be agile and flexible since their infrastructure was now running on the cloud and the platform was online 24/7. The platform and the digital services were eventually delegated to DevOps teams. One challenge the firm had to overcome was aligning the teams not only internally but also across teams using the platform. The short- vs. long-term orientation is solved in DevOps teams because the tasks are discussed and coordinated within the team and with other teams. Deployments are coordinated, and planning steps in the Non-Food Retail case changed significantly, which meant convincing the IT function:

> *New ideas are constantly being added, and the biggest challenge was simply to convince the IT people, who have a very fixed planning rhythm, that we have a completely different speed here* (Retail Non-Food, Team Lead).

Hence, DevOps teams can encourage common time rhythms and the problem of different control dynamics is diminished because activities are aligned via a high level of traceability.

5. Discussion

In this study, we identify four main tensions between development and operations activities that can be addressed through implementing cross-functional DevOps teams. Our empirical study of six DevOps teams led to the identification of three dimensions controlling DevOps teams navigating these tensions. To further develop these findings, we conceptualize DevOps control as a series of frequent adjustments between different control mechanisms. This differs significantly from prior research, which has primarily paid attention to the contextual antecedents and performance consequences of control modes or the degree of control portfolio configuration (Choudhury & Sabherwal, 2003; Kirsch, 1996; Kirsch, 1997).

The model of DevOps control in internal IT teams introduced in this paper explains the coexistence of controls used successfully in practice and provides insights into the control of DevOps teams as a dynamic and iterative approach, rather than a linear process (Choudhury & Sabherwal, 2003; Gregory et al., 2013; Kirsch, 2004). Fig. 2 below illustrates the three control mechanisms that help to cope with the tensions that arise in DevOps teams.

Participation in shared vision: This dimension develops and enacts a common view for the future shared orientation in DevOps teams. Our findings indicate that DevOps teams' control themes include *consensus, automation,* and the *restructuring* of activities and processes for managing the software delivery lifecycle. The team achieves its goals and team members come to an agreement regarding their usage of methods by making joint decisions regarding the software product. Some of our cases still had a formal team leader to make certain high-level decisions. However, our cases indicate that classic tight project control in terms of day-to-day business needs hinders the development of a common mindset. Control effects and control choices in development and operations units differ significantly. While both focus on different goals such as efficiency and quality (Kirsch, 2004) vs. stability and ticket resolution (Trusson et al., 2014). DevOps teams need to organize many tasks that were previously performed by highly specialized people from several IT functions and units. Hence, DevOps teams work toward a common goal and combine the goals of both worlds to ensure quality, speed, and stability. The *method discomfort* tension between development and operations disappears since the control method is based on project approaches (development) and best-practices (operations). DevOps mitigates this discomfort by enabling operations to work with agile methods and development to benefit from using best practices for ticket resolution. The dimension of participation in a shared vision connects goals and methods from both sides in DevOps teams to develop an effective approach to control.

Right of co-determination: The dimension determines how to assume responsibility and navigate decision rights for the software delivery lifecycle in DevOps teams. The establishment of cross-functional DevOps teams is one way of using control mechanisms to navigate *method discomfort* and *decision rights* tensions. Through fostering social capital in DevOps teams, team members are enabled to adopt new best-practices and agile methods. The control context has different characteristics for development and operations (Kirsch et al., 2002; Shaft & Vessey, 2006). DevOps teams can mitigate the *decision rights* tension, which involves project vs. service management because the organizational setup of established IT functions changes becomes oriented toward teams and products rather than projects and services. This movement toward product orientation leads to a change of control context and brings different managers closer together. Decisions are no longer made in different IT units; DevOps team members are open to learning new types of work and team leaders are open to adopting new leadership styles that demand more responsibility from the team. The *right of co-determination* dimension suggests that DevOps teams have an active voice and make most product-related decisions themselves. Hence, the classic decision rights tension between project and service managers becomes blurred.

Common sense of determination: Describes the dimension by which a sense of ownership is achieved through the adaption of the leadership style and transparency for all activities of the DevOps team. Our findings highlight the shift toward the management of

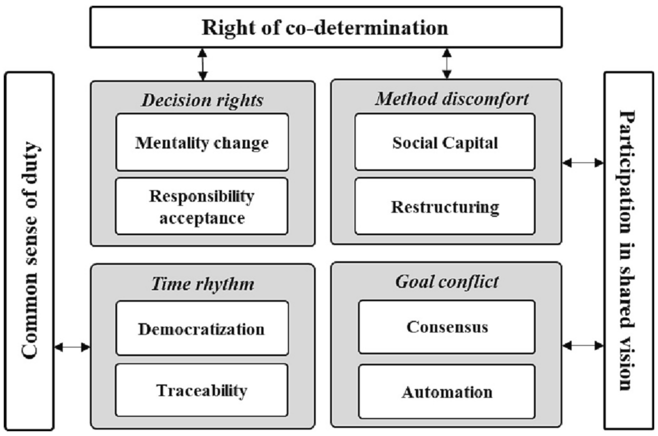

Fig. 2. Mitigation of tensions through control mechanisms.

software delivery, such that there is no date at which projects will be shut down. DevOps navigates the time rhythm tension through the democratization of control and higher traceability for development and operations tasks. Our study suggests that DevOps teams outperform other teams not in *project* setups but in *product* setups. Control dynamics in development and operations differ because development control dynamics appear due to rapid project lifecycle whereas operation dynamics manifest in long-term stability phases with few changes. DevOps teams can solve the *time rhythm* tension associated with short- vs. long-term orientation in that development and operations tasks are now mapped together in the software delivery lifecycle for which DevOps teams are responsible.

5.1. *Implications for research*

Based on our findings, we present a model that serves as a DevOps control approach for navigating development and operations tensions. Our research complements prior literature focusing on control in project management (Remus et al., 2020) and digital agility in software development projects (Salmela et al., 2022) by showing how these tensions can be mitigated in DevOps teams.

Specifically, our model illustrates how IT managers and DevOps teams can use control to navigate tensions between development and operations in three different ways. First, the mechanism *participation in shared vision* extends prior literature. Our findings support Hemon-Hildgen et al. (2020) observation that classical silo setups in IT functions do not work well with DevOps approaches. Removing organizational silos enables team collaboration and fosters the cultural mindset of the team members to work toward a common goal—the product. One key concern of DevOps teams is automating manual processes whenever possible. Automation is recommended not only for continuous testing and code delivery, as presented by Fitzgerald and Stol (2017), but also for monitoring activities such as sending automated alerts in case of software malfunctions, etc. Minimizing standardized, highly repetitive manual processes provides DevOps teams with more time and freedom to work on product enhancements and team culture.

The second implication addresses control choices and context (Kirsch, 2004; Wiener et al., 2016). Our model shows the transformation from project control to product-oriented control through the dimension *right of co-determination*. DevOps teams require redesigning processes and establishing end-to-end IT product responsibility. This is enabled by bringing people together into a single team responsible for the entire software delivery lifecycle. Our findings indicate that the tasks, knowledge, and skills of DevOps teams depend on the product. Team leads train team members to ensure a balanced representation of both development and operations needs—even in small teams. This builds on extant research on skills and collaboration in DevOps teams, such as work by Hemon et al. (2019) and Wiedemann and Wiesche (2018).

Third, the mechanism of *common sense of duty* builds on and extends prior research, recommending shifting from command-and-control to a more collaborative and enabling leadership style. This is in line with prior literature in the area of agile and self-organization, such as work by Matook et al. (2016) and DevOps studies e.g., Hemon et al. (2019). Successful DevOps teams have members with deep expertise and skills in technology and social competency. Our focus on aligning development and operations extends existing literature focusing on software development project control (Cram et al., 2016a; Kirsch et al., 2002; Wiener et al., 2019) to consider how software operations can be integrated and controlled. Considering operations work fully in the planning and development process leads to a smoother-running software product. Our results show that when operations is involved in planning meetings, the team leaves room for unplanned operations work needed to solve problems and repair software defects.

5.2. *Implications for practice*

Our study has several important implications for managers and practitioners. For development teams, we recommend prioritizing the rapid development of new software features and updates (McAvoy & Butler, 2009), i.e., teams need permission to decide on the technology and tasks instead of opening a ticket. Based on our findings, high levels of autonomy are guaranteed within DevOps teams and team leads can decide how much autonomy the team requires. Thus, enabling work within the team rather than in separated IT units leads to better collaboration and communication between team members, and conflicts can be resolved through orienting teams toward the product.

An increasing number of companies are currently implementing DevOps teams to achieve speed and stability and other benefits that are key to digital innovation. However, integrating DevOps is associated with its own tensions. Our findings indicate that not every tension appears in every context. Our control approach can help management overcome tensions that may arise in DevOps teams through the use of dedicated control mechanisms. Practitioners can use our control approach to guide and manage DevOps implementation and improve existing DevOps teams.

In our case studies of DevOps teams in firms across a variety of different industries, the type and intensity of tensions varied across DevOps teams. We recommend applying the control mechanisms in a dynamic, iterative, and adaptive way. DevOps integration must be well considered and is highly dependent on the software product. Not every product is suitable for DevOps.

5.3. *Limitations and further research*

Our research has certain limitations, which point to avenues for further research. We examined the control themes of DevOps teams from IT function perspectives, implicitly excluding other control relationships, such as business stakeholders. While we made inroads into the neglected field of enacting control in DevOps in practice, our study concentrates more on the process of implementing and controlling DevOps teams. Further research is needed to flesh out our initial findings on how to successfully integrate control mechanisms. Our research focuses on control and relationships, considering DevOps teams as product management rather than project management entities. Future investigations could examine suitable measures of success and failure for DevOps teams and address the influence of control characteristics. Based on the sets of individual tensions and control mechanisms summarized in this study, future research could also consider control configurations. Finally, our study focused on intraorganizational control; we neglected the potential role of interorganizational control and its effect on control.

6. Conclusion

This research provides insights into the neglected area of control in cross-functional teams, elucidating the dynamic and iterative nature of the control of DevOps teams. Our review of relevant extant research into a collaboration between IS development and operations revealed four control-related tensions: *goal conflict, method discomfort, decision rights,* and *time rhythm*. We derived three aggregated dimensions (*participation in a shared vision, right of co-determination,* and *common sense of duty*) for controlling DevOps teams that can be used to successfully navigate these tensions. Our DevOps control model illustrates the interplay of dimensions, mechanisms, and control tensions and offers suggestions on how tensions can be iteratively and dynamically controlled. We hope that researchers and practitioners will apply our findings to improve interfirm relationships, overcome tensions stemming from cross-functional teams, and integrate DevOps teams successfully in order to foster digital innovations and products within their organizations.

CRediT authorship contribution statement

Anna Wiedemann: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Formal analysis, Investigation, Resources. **Manuel Wiesche:** Conceptualization, Writing – original draft, Writing – review & editing, Methodology, Investigation. **Heiko Gewald:** Conceptualization, Writing – original draft, Writing – review & editing, Resources. **Helmut Krcmar:** Conceptualization, Writing – original draft, Writing – review & editing, Resources.

Acknowledgements

The authors gratefully acknowledge the constructive input of the senior editor which greatly helped us to improve the paper. The authors extend their gratitude to all participants involved in the data collection activities for their collaboration and contribution during this study. We thank the German Federal Ministry of Education and Research (BMBF) for funding this research under grant code 03FH005PX4.

Appendix A. Appendix

Table 5
Case description.

Case	DevOps Service Description and Team Setting
Retail Food	The objective is to build and operate a delivery service app. The team consists mainly of developers. They are responsible for developing features, conducting continuous delivery, monitoring, and providing daytime support. Decentralized support units provide on-call duty outside normal business hours.
Banking	This team runs a securities management system that provides tools and is highly concentrated on integrating automated processes across several environments. DevOps activities are automated for the packaging and the automation of tests at the integration stage, release management, monitoring, end-user communication, and product support (first- and second-level support); third-level support is provided by the developer teams.
Insurance	This team builds and runs an internal delivery platform for evaluating infrastructure, including development tasks such as building features, operations tasks such as maintenance upgrades, capacity scaling, security fixes, and strategic work. They have rotating responsibilities for on-call duty.
Service	This team builds and runs web applications that provide customers with ratings of different products. The team members are responsible for developing the application, monitoring, bug fixes, and quality assurance. The team lead is responsible for managing on-call duty and training team members for on-call duty.
Retail Store	This team builds and runs an online shop. The members are responsible for the check-out process. The main tasks are front-end and back-end development, quality assurance, automation implementation, and operations tasks like monitoring. The team is responsible for on-call duty.
Retail Non-Food	This team builds and runs an internal platform, which is the foundational structure of the online shop of the company. On-call duty is organized within the team. Team members are mainly system administrators who assumed the development activities concerning the platform. The main tasks are developing and providing platform support, integration of continuous delivery, integration of infrastructure as a code, building interfaces for development teams, cooperation with vertical developer teams for testing, and the automation of system configuration.

Appendix B. Appendix

Table 6
Aggregate Dimension 1: Participation in shared vision.

Aggregate dimension: Participation in shared vision
1. **Consensus (addresses goal conflict)**
a) Solve differences in cultural aspects to avoid team disagreements *"We have a challenge, especially in the cultural aspect, and this is reflected at various levels. In budget discussions, in discussions about understanding, and so on."* (Banking, Group Manager)
b) Give responsibility to teams for more autonomous decision-making *"A team can make simple technology decisions autonomously […] it's a good thing that the team's autonomy is relatively high."* (Retail Food, Team Member)
2. **Automation (addresses goal conflict)**
c) Reduce manual activities for operations *"It helps to prevent incidents and improves the quality of the deployment because we no longer operate any kind of deployment manually."* (Insurance, Head of Architecture)
d) Testing of software and production deployments *"It [the software update] is built, is tested, and goes to the test environment. Then it's also tested again with other services and automatically goes into production. If it wasn't good […] the developer gets a message during testing, this is broken. What you built was not functional."* (Retail Food, Team Member)
e) Avoid interruptions due to manual processes *"We don't want anything to be done manually, we want everything to be done* via *automation […] in the end I don't know, 80% were automated, and then I had to write a ticket."* (Retail Store, Team Lead)
f) Enable proactive monitoring to recognize failures early *"Automated monitoring for the features,* i.e.*, for certain threshold values, alerts are triggered or, if there is too much CPU load,* i.e.*, a server that suddenly becomes very slow, then, of course, we also receive alerts for this."* (Retail Food, Team Member)
3. **Restructuring (addresses method discomfort)**
g) Break down vertical IT silo structures *"This is then reflected in decisions such as these, that perhaps areas are moving closer together and that we are trying to break down these vertical silos bit by bit."* (Retail Food, Product Owner)
h) Creation of agile operations *"We are also contributing to propose new ways to adapt our agile performance management processes to fully embrace the DevOps principles. So ensure that regardless of the fact that there are people belonging in two different pillars they can have still the same targets aligned to a product, to the manager to help."* (Insurance, Head of Architecture)
i) Reshuffle the workforce *"This is probably the most difficult component because this will mean really to reshuffle the entire workforce or part of the workforce but if you want to leverage to this practice." (Insurance, Head of Architecture)*
j) Solve dependencies on other IT units *"The teams are all on the same level, regardless of which squat or tribe they are in, and then above that there are the respective heads of who then take over a bit of the disciplinary leadership, but the technical decisions or something like that remains within the teams."* (Retail Food, Team Member)

Table 7
Aggregated Dimension 2: Right of co-determination.

Aggregate dimension: Right of co-determination
4. **Social capital (addresses method discomfort)**
k) Adapt new skills and acquisition of knowledge *"What is hardest for me is still that I have never been a programmer and probably will never become a full programmer because I have done ops for too long and it is not the speaking of the language that is the problem but just some of the concepts."* (Retail Store, Team Member)
l) Enable few people with the right skills to work in teams

(*continued on next page*)

Table 7 (*continued*)

Aggregate dimension: Right of co-determination
"We have a very big skills problem. There are very, very few people, especially in application management, who can work analytically with such data." (Banking, Team Member)
m) Bring different professional backgrounds together *"I had to learn over the years what kind of rough edges there are in such a system. Today, I would say that I could do sys-admin. […] It's not my focus topic, but I also say that I'm quite good at 70% of it."* (Service, Team Lead)
n) Share knowledge within the team to ensure backup options if the team lead is unavailable *"We have mixed expertise and backgrounds on this team from the classic sys admins who used to set up servers and build things, to the developers like my person."* (Retail Food, Team member)
5. **Mentality change (addresses decision rights)**
p) Strive to adopt a "you build it, you run it" mentality *"What can we put into the teams to make that possible and more interesting? For example, this was a topic that we started last year called "You build it, you run it," where we simply want to make the teams feel more responsible for the product."* (Retail Food, Team Member)
q) Transform mindsets to understand development and operations tasks *"You need exactly this mindset, that you say: okay, when I program something, I also have to think about how it will work under load in production around the clock."* (Retail Store, Team Lead)
r) Move from project to product orientation *"For me, that's where the product differs from the project. A product runs until sundown, until it is no longer needed."* (Retail Store, Team lead)
6. **Accepting responsibility (addresses decision rights)**
s) Establish shared responsibilities to avoid finger pointing *"Get people to feel end-to-end collectively responsible for lead time, both ways, and then stop pointing fingers at each other."* (Banking, Former Group Lead)
t) Taking responsibility for software quality *"The biggest challenge now is making sure that the entire platform is in a healthy state. So, this topic: How do I localize when a problem occurs where this problem has its cause?"* (Retail Food, Team Member)

Table 8
Aggregate Dimension 3: Common sense of duty.

Aggregate Dimension: Common sense of duty
7. **Democratization (addresses time rhythm)**
u) Loss of supervisors' control *"But the problem that comes then […] is that you have a loss of control, especially from superiors towards now the DevOps team."* (Retail Food, Team Member)
v) Change leadership style from command-and-control to collaborative *"The world evolved to a more, to a faster environment, supported by technology of course, to develop things in a faster way […]. This also changes leadership style from directive to more collaborative and consensual."* (Insurance, Head of Architecture)
8. **Traceability (addresses time rhythm)**
w) Greater prioritization of operational work *"Operation[s] has become a much higher priority with all the issues. The whole monitoring issue is something I notice brutally because the quality of many topics is getting better and better every day, whenever something pops up."* (Retail Store, Team Member)
x) Voluntary readiness of performance team members for on-call duties *"Many people want to take on stand-by because they also want to take responsibility for what they have built."* (Retail Food, Team Member)
y) Ensure execution traceability of development and operations tasks *"Now there is a massive difference because this operating share is also included in the planning to a much greater extent and is simply also seen."* (Retail Store, Team Lead)

References

Agarwal, N., Soh, C., & Yeow, A. (2022). Managing paradoxical tensions in the development of a telemedicine system. *Information and Organization, 32*(1).

Aime, F., Humphrey, S., DeRue, D. S., & Paul, J. B. (2014). The riddle of heterarchy: Power transitions in cross-functional teams. *Academy of Management Journal, 57*(2), 327–352.

Ajer, A. K. S., Hustad, E., & Vassilakopoulou, P. (2021). Enterprise architecture operationalization and institutional pluralism: The case of the Norwegian hospital sector. *Information Systems Journal, 31*(4), 610–645.

April, A., Huffman Hayes, J., Abran, A., & Dumke, R. (2005). Software maintenance maturity model (Smmm): The software maintenance process model. *Journal of Software Maintenance and Evolution: Research and Practice, 17*(3), 197–223.

Banker, R. D., Davis, G. B., & Slaughter, S. A. (1998). Software development practices, software complexity, and software maintenance performance: A field study. *Management Science, 44*(4), 433–450.

Banker, R. D., Slaughter, S., Swatman, P., Wagenaar, R., & Wrigley, C. (1994). Project size and software maintenance productivity: Empirical evidence on economies of scale in software maintenance. In *International conference on information systems. New York, USA*.

Barki, H., Rivard, S., & Talbot, J. (2001). An integrative contingency model of software project risk management. *Journal of Management Information Systems, 17*(4), 37–69.

Berente, N., & Yoo, Y. (2012). Institutional contradictions and loose coupling: Postimplementation of Nasa's enterprise information system. *Information Systems Research, 23*(2), 376–396.

Choudhury, V., & Sabherwal, R. (2003). Portfolios of control in outsourced software development projects. *Information Systems Research, 14*(3), 291–314.

Cram, A., & Newell, S. (2016). Mindful revolution or mindless trend? Examining agile development as a management fashion. *European Journal of Information Systems, 25*(2), 154–169.

Cram, W. A., Brohman, K., & Gallupe, R. B. (2016a). Information systems control: A review and framework for emerging information systems processes. *Journal of the Association for Information Systems, 17*(4), 216–266.

Cram, W. A., Brohman, M. K., & Gallupe, R. B. (2016b). Hitting a moving target: A process model of information systems control change. *Information Systems Journal, 26*(3), 195–226.

Cross, J., Earl, M. J., & Sampler, J. L. (1997). Transformation of the IT function at British petroleum. *MIS Quarterly, 21*(4), 401–423.

Debois, P. (2008). Agile infrastructure and operations: How infra-gile are you?. In *Agile 2008 conference* (pp. 202–207). Toronto, Canada: IEEE.

Debois, P. (2011). Opening statement. *Cutter IT Journal, 24*(8), 3–5.

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.

Edberg, D., Ivanova, P., & Kuechler, W. (2012). Methodology mashups: An exploration of processes used to maintain software. *Journal of Management Information Systems, 28*(4), 271–304.

Ethiraj, S. K., Kale, P., Krishnan, M. S., & Singh, J. V. (2005). Where do capabilities come from and how do they matter? A study in the software services industry. *Strategic Management Journal, 26*(1), 25–45.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176–189.

Gioia, D. A., Corley, K. G., & Hamilton, A. L. (2013). Seeking qualitative rigor in inductive research: Notes on the Gioia methodology. *Organizational Research Methods, 16*(1), 15–31.

Gkeredakis, M., & Constantinides, P. (2019). Phenomenon-based problematization: Coordinating in the digital era. *Information and Organization, 29*(3).

Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. New York, NY: Aldine Publishing Company.

Gregory, R. W., Beck, R., & Keil, M. (2013). Control balancing in information systems development offshoring projects. *MIS Quarterly, 37*(4).

Gregory, R. W., & Keil, M. (2014). Blending bureaucratic and collaborative management styles to achieve control ambidexterity in is projects. *European Journal of Information Systems, 23*(3), 343–356.

Haight, C., & Spafford, G. (2022). *Research roundup for Devops, 2022*. Gartner.

Hemon-Hildgen, A., Rowe, F., & Monnier-Senicourt, L. (2020). Orchestrating automation and sharing in Devops teams: A revelatory case of job satisfaction factors, risk and work conditions. *European Journal of Information Systems, 29*(5), 474–499.

Hemon, A., Lyonnet, B., Rowe, F., & Fitzgerald, B. (2019). From agile to devops: Smart skills and collaborations. *Information Systems Frontiers*, 1–19.

Henderson, J. C., & Lee, S. (1992). Managing I/S design teams: A control theories perspective. *Management Science, 38*(6), 757–777.

Herbsleb, J. D., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering, 29*(6), 481–494.

Heumann, J., Wiener, M., Remus, U., & Mähring, M. (2015). To coerce or to enable? Exercising formal control in a large information systems project. *Journal of Information Technology, 30*(4), 337–351.

ISACA. (2012). *Cobit - 5th edition*. USA: Rolling Meadows.

Jaworski, B. J. (1988). Toward a theory of marketing control: Environmental context, control types, and consequences. *The Journal of Marketing, 52*(3), 23–39.

Keil, M., Lee, H. K., & Deng, T. (2013). Understanding the most critical skills for managing IT projects: A Delphi study of IT project managers. *Information & Management, 50*(7), 398–414.

Kim, C., & Westin, S. (1988). Software maintainability: Perceptions of Edp professionals. *MIS Quarterly, 12*(2), 167–185.

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The Devops handbook*. Portland, USA: IT Revolution Press, LLC.

Kirsch, L. J. (1996). The management of complex tasks in organizations: Controlling the systems development process. *Organization Science, 7*(1), 1–21.

Kirsch, L. J. (2004). Deploying common systems globally: The dynamics of control. *Information Systems Research, 15*(4), 374–395.

Kirsch, L. J., Sambamurthy, V., Ko, D.-G., & Purvis, R. L. (2002). Controlling information systems development projects: The view from the client. *Management Science, 48*(4), 484–498.

Kirsch, L. S. (1997). Portfolios of control modes and IS project management. *Information Systems Research, 8*(3), 215–239.

Krancher, O., Luther, P., & Jost, M. (2018). Key affordances of platform-as-a-service: Self-organization and continuous feedback. *Journal of Management Information Systems, 35*(3), 776–812.

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

Luftman, J. N., Lewis, P. R., & Oldach, S. H. (1993). Transforming the enterprise: The alignment of business and information technology strategies. *IBM Systems Journal, 32*(1), 198–221.

Mähring, M. (2002). It project governance: A process-oriented study of organizational control and executive involvement. *Business Administration, 15*.

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

Matook, S., Soltani, S., & Maruping, L. (2016). Self-organization in agile ISD teams and the influence on exploration and exploitation. In *International conference on information systems. Dublin, Ireland*.

McAvoy, J., & Butler, T. (2009). The role of project management in ineffective decision making within agile software development projects. *European Journal of Information Systems, 18*(4), 372–383.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. Thousands Oaks, Carlifornia: Sage Publications.

Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization, 17*(1), 2–26.

Nelson, K. M., Nadkarni, S., Narayanan, V. K., & Ghods, M. (2000). Understanding software operations support expertise: A revealed causal mapping approach. *MIS Quarterly, 24*(3), 475–507.

Onita, C., & Dhaliwal, J. (2011). Alignment within the corporate IT unit: An analysis of software testing and development. *European Journal of Information Systems, 20*(1), 48–68.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research, 2*(1), 1–28.

Pollard, C., & Cater-Steel, A. (2009). Justifications, strategies, and critical success factors in successful ITIL implementations in US and Australian companies: An exploratory study. *Information Systems Management, 26*(2), 164–175.

Putnam, L. L., Fairhurst, G. T., & Banghart, S. (2016). Contradictions, dialectics, and paradoxes in organizations: A constitutive approach. *The Academy of Management Annals, 10*(1), 65–171.

Remus, U., & Wiener, M. (2012). The amount of control in offshore software development projects. *Journal of Global Information Management (JGIM), 20*(4), 1–26.

Remus, U., Wiener, M., Saunders, C., & Mähring, M. (2020). The impact of control styles and control modes on individual-level outcomes: A first test of the integrated IS project control theory. *European Journal of Information Systems, 29*(2), 134–152.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., et al. (2017). Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study. *J. Syst. Softw., 123*, 263–291.

Salmela, H., Baiyere, A., Tapanainen, T., & Galliers, R. D. (2022). Digital agility: Conceptualizing agility for the digital era. *Journal of the Association for Information Systems, 23*(5), 1080–1101.

Sebastian, I. M., Ross, J. W., Beath, C., Mocker, M., Moloney, K. G., & Fonstad, N. O. (2017). How big old companies navigate digital transformation. *MISQ Executive, 16*(3), 197–213.

Shaft, T. M., & Vessey, I. (2006). The role of cognitive fit in the relationship between software comprehension and modification. *MIS Quarterly*, 29–55.

Stachour, P., & Collier-Brown, D. (2009). You don't know Jack about software maintenance. *Communications of the ACM, 52*(11), 54–58.

Tarafdar, M., & Tanriverdi, H. (2018). Impact of the information technology unit on information technology-embedded product innovation. *Journal of the Association for Information Systems, 19*(8).

Trusson, C. R., Doherty, N. F., & Hislop, D. (2014). Knowledge sharing using IT service management tools: Conflicting discourses and incompatible practices. *Information Systems Journal, 24*(4), 347–371.

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

Väätäjä, H. K., Hildén, E., Roto, V., & Uusitalo, K. (2016). A case study on participatory approach to support shift to experience design of work tools in B2b context. In *International conference on information systems. Dublin, Ireland*.

Wallace, L., Keil, M., & Rai, A. (2004). How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model. *Decision Sciences, 35*(2), 289–321.

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

Wiedemann, A., & Wiesche, M. (2018). Are you ready for Devops? Required skill set for DevOps teams. In *European conference on information systems. Portsmouth, UK*.

Wiener, M., Mähring, M., Remus, U., Saunders, C., & Cram, W. A. (2019). Moving IS project control research into the digital era: The "why" of control and the concept of control purpose. *Information Systems Research, 30*(4), 1387–1401.

Wiener, M., Mähring, M., Remus, U., & Saunders, C. S. (2016). Control configuration and control enactment in information systems projects: Review and expanded theoretical framework. *MIS Quarterly, 40*(3), 741–774.

Yin, R. K. (2018). *Case study research and applications: Design and methods*. Los Angeles: SAGE Publication Inc.

Young-Hyman, T. (2017). Cooperating without co-laboring: How formal organizational power moderates cross-functional interaction in project teams. *Administrative Science Quarterly, 62*(1), 179–214.