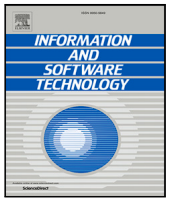
Information and Software Technology 139 (2021) 106672

Contents lists available at ScienceDirect

Information and Software Technology

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



The organization of software teams in the quest for continuous delivery: A grounded theory approach

Leonardo Leite [a,b,*], Gustavo Pinto [c], Fabio Kon [a], Paulo Meirelles [d,a]

[a] University of São Paulo (USP), Brazil
[b] Federal Service of Data Processing (Serpro), Brazil
[c] Federal University of Pará (UFPA), Brazil
[d] Federal University of ABC (UFABC), Brazil

ARTICLE INFO

Keywords:
DevOps
Continuous delivery
Release process
Software teams

ABSTRACT

Context: To accelerate time-to-market and improve customer satisfaction, software-producing organizations have adopted continuous delivery practices, impacting the relations between development and infrastructure professionals. Yet, no substantial literature has substantially tackled how the software industry structures the organization of development and infrastructure teams.
Objective: In this study, we investigate how software-producing organizations structure their development and infrastructure teams, specifically how is the division of labor among these groups and how they interact.
Method: After brainstorming with 7 DevOps experts to better formulate our research and procedures, we collected and analyzed data from 37 semi-structured interviews with IT professionals, following Grounded Theory guidelines.
Results: After a careful analysis, we identified four common organizational structures: (1) siloed departments, (2) classical DevOps, (3) cross-functional teams, and (4) platform teams. We also observed that some companies are transitioning between these structures.
Conclusion: The main contribution of this study is a theory in the form of a taxonomy that organizes the found structures along with their properties. This theory could guide researchers and practitioners to think about how to better structure development and infrastructure professionals in software-producing organizations.

1. Introduction

To remain competitive, many software-producing corporations seek to speed up their release processes [1,2]. Organizations may adopt continuous delivery practices in their quest to accelerate time-to-market and improve customer satisfaction [3]. However, continuous delivery also comes with challenges, including profound impacts on various aspects of the software engineering practice [4]. With an automated deployment pipeline, one can, for example, question the role of an engineer responsible solely for new deployments. Since release activities involve many divisions of a company (e.g., development, operations, and business), adopting continuous delivery impacts organizational structure [3].

Given recent transformations, there is a need to better understand the organizational structures that the software industry adopts for development and infrastructure employees.[1] By organizational structure, we mean the differentiation (division of labor) and integration (interaction) [5] of operations activities (application deployment, infrastructure setup, and service operation in run-time) among development and operations groups.

However, there is no substantial literature tackling how organizations have structured their development and operations groups. The existing literature presents some classifications for organizational structures [6–10]. Still, most of these studies are not based on empirical evidence, which limits the understanding of how the authors conceived their classifications. An exception is the work of Shahin et al. [10], whose focus was to understand how organizations arrange development and operations teams to embrace continuous delivery practices optimally. However, our quest is not centered around such practices, as it is more general about the structuring of development and infrastructure professionals. Also, note that social theories are rarely confirmed but

* Corresponding author at: University of São Paulo (USP), Brazil.
E-mail address: leofl@ime.usp.br (L. Leite).

[1] In the context of this work, development teams (also called product teams) are responsible for developing business services, while the infrastructure staff uniformly provides computational resources for diverse applications.

https://doi.org/10.1016/j.infsof.2021.106672
Received 30 August 2020; Received in revised form 29 April 2021; Accepted 16 June 2021
Available online 24 June 2021

are instead corroborated, confronted, or evolved by new studies [11–13]. Our work is unique in the sense that (1) it analyzes a different sample and (2) it follows a different research method; moreover, (3) we discuss the discovered structures in more depth, and (4) we highlight the similarities and differences between the structures discovered by us and Shahin et al. thus bringing more robustness to the knowledge in the area.

In this way, this paper addresses the following **research questions**:

> **RQ1:** *What organizational structures do software-producing organizations adopt to structure operations activities (application deployment, infrastructure setup, and service operation in run-time) among development and infrastructure groups?*
> **RQ1.1:** *What are the properties of each of these organizational structures?*
> **RQ1.2:** *Are some organizational structures more conducive to continuous delivery than others?*

To answer these questions, we applied Grounded Theory [14], a methodology well-suited for generating theories. The primary outcome of this research approach is a taxonomy, which is our emerging theory. A taxonomy is a classification system that groups similar instances to increase users' cognitive efficiency, enabling them to reason about classes instead of individual instances [15].

We collected preliminary data in brainstorming conversations with seven specialists, who helped us better understand the relevance of the problem and to shape the questions to be asked in follow-up interviews. We then conducted semi-structured interviews with 37 IT professionals. Based on analysis of the interviews, we discovered four organizational structures:

> **(i)** Traditional *siloed departments*, hindering cooperation among development and operations.
> **(ii)** *Classical DevOps*, focusing on communication and collaboration among development and operations.
> **(iii)** *Cross-functional teams*, taking responsibility for both software development and infrastructure management.
> **(iv)** *Platform teams*, providing highly-automated infrastructure services to assist developers.

For each of these organizational structures, we identified *core* and *supplementary* properties. An organization classified as adopting a given structure will present most of the core properties associated with that structure. Supplementary properties, by contrast, support the explanation of more detailed structural patterns, and an organization may (or may not) exhibit them.

This paper contributes to the area by presenting a systematically-derived taxonomy of organizational structures, based on recent field observations and employing a well-accepted methodology. In particular, our taxonomy brings the following key benefits: *(i)* it helps practitioners to differentiate *classical DevOps* from *cross-functional teams*, which were traditionally blended under the term DevOps [2,16], and *(ii)* it highlights the *platform team* as a promising alternative for organizations. Moreover, our taxonomy can support practitioners to discuss the current situation of their corporations, supporting decisions on structural changes. It also supports understanding the state of an unknown organization, which can help, for example, engineers in job interviews to evaluate the suitability of working for a given company.

Some of the findings we discuss in this paper relate to *delivery performance*, a construct composed of quantitative metrics, which we explain in Section 2. We explain our research approach in Section 3, and we present our taxonomy in detail in Section 4. After this, in Section 5 we discuss the feedback from interviewees and our evaluation. Section 6 discusses related work, whereas Section 7 presents the limitations of this work. Finally, we draw our conclusions and plans for future work in Section 8.

2. Background

Delivery performance combines three metrics: frequency of deployment, time from commit to production, and mean time to recovery [17]. It correlates to the organizational capability of achieving both commercial goals (profitability, productivity, and market share) and noncommercial goals (effectiveness, efficiency, and customer satisfaction) [18]. We used this construct as an indication of how successful an organization has been in adopting continuous delivery. We asked each participant in our study about each of these metrics to define the delivery performance in the interviewee's context.

Based on a survey with 27,000 responses, Forsgren et al. [18] applied cluster analysis to these metrics and discovered three groups: *High performers* were characterized as those with multiple deployments per day, commits that take less than 1 h to reach production, and incidents repaired in less than 1 h. *Medium performers* deploying once per week to once per month, had a time from commit to production between one week and one month, and took less than one day to repair incidents. *Low performers* presented the same characteristics of medium performers for deployment frequency and time from commit to production, but take between one day and one week to repair incidents.

In our research, we are interested in distinguishing between high and non-high performers, not in identifying medium or lower performers. However, the above clusters present a problem, because there is a gap in the values used to identify the high and the medium performers clusters. We circumvented this problem by considering an organization as a high performer if *(i)* it is within the boundaries limiting the cluster of high performers defined above, or *(ii)* it violates no more than one high-performance threshold by no more than one point in the scale adopted for the metric. The scales for each metric are:

- *Frequency of deployment:* multiple deploys per day; between once per day and once per week; between once per week and once per month; between once per month and once every six months; fewer than once every six months.
- *Time from commit to production:* less than one hour; less than one day; between one day and one week; between one week and one month; between one month and six months; more than six months.
- *Mean time to recovery:* less than one hour; less than one day; between one day and one week; between one week and one month; between one month and six months; more than six months.

3. Study design

This section presents our research approach, including the process we used to collect and analyze data.

3.1. Grounded theory

Our research aims at generating a theory in the form of a taxonomy for organizational structures in the context of continuous delivery. Broadly speaking, a theory is a system of ideas for explaining a phenomenon [15]. Taxonomies, on the other hand, are classifications, i.e., collections of classes, wherein each class is an abstraction that describes a set of properties shared by the instances of the class [15]. If the taxonomy provides explanation, it can be considered a *theory for understanding*: a system of ideas for making sense of what exists or is happening in a domain [15]. While applying "taxonomy" to denote a theoretical formulation, we employ "classification" in a broader sense, including classification proposals without theoretical formulation.

Grounded Theory (GT) is a well-suited methodology for generating taxonomies [15] and a widely used research approach in software engineering [19–22,16,23]. A grounded theory must fit the data, predict, explain, have relevance to the field, and be modifiable [14]. Its primary advantage is to encourage deep immersion in the data, which

may protect researchers from missing instances or oversimplifying and over-rationalizing processes [15]. GT is also adequate for our purposes since, according to Stol et al. [19], it is well-suited for questions like "what's going on here?". In our case, we want to know what is going on in software-producing organizations that are taking advantage of continuous delivery. Since multiple GT variants exist, it is important to state which variant we adopt. In this paper, we base our approach on the seminal book *The Discovery of Grounded Theory* from Glaser and Strauss [14], which describes what is known as the "classical Grounded Theory" [19]. While a theory itself must emerge from data, Glaser and Strauss do not disallow pre-established research questions.[2]

The *constant comparative method* is the core method for producing a grounded theory. It relies on rigorous analysis of qualitative data, and it is accomplished with *coding*, a process of condensing original data into a few words with conceptual relevance, which give emergence to theoretical concepts. Glaser and Strauss do not prescribe a precise coding format [14], sufficing that the researcher annotates concepts and adheres to the following rules: *(i)* when noting a concept, compare this occurrence with the previous occurrences of the same or similar concepts and *(ii)* while coding, if conflicts and reflections over theoretical notions arise, write a memo on the ideas. A memo is an unstructured note reflecting the researcher's thoughts at a specific point in time. We describe in more detail in Section 3.5 how we applied coding.

Besides the rigorous analysis of qualitative data, GT also relies on the researcher's *theoretical sensitivity*; i.e., their capacity to have theoretical insight into a substantive area. Our theoretical sensitivity comes from direct experience in the IT industry and our previous works on DevOps and software engineering [24,25,23], especially a survey on the DevOps literature [26]. This acquired theoretical sensitivity explains our capacity to pose the research questions of this paper.

This article also builds on our recent work, an extended abstract that briefly presents the four organizational structures of our taxonomy [27] and a short paper presenting the platform team structure only [28]. In contrast, the current paper describes in detail all four organizational structures and their properties.

In GT, data collection and analysis co-mingle and build on each other, so the emerging theory guides which data to sample next, considering gaps and questions suggested in prior analysis. This process—called *theoretical sampling*—avoids the usual statistical notions of verificational methods, such as significant sample. Instead, researchers must establish the theoretical purpose of the sample, defining multiple comparison groups, maximizing variation among groups to identify similarities, and minimizing variation to determine differences. We approached theoretical sampling primarily by: *(i)* valuing the diversity of people and organizations in our sample, strengthening the transferability of our theory; and *(ii)* interviewing people in contexts in which we could explore hypotheses weakly supported by the chain of evidence built so far. We elaborate more on the choices of participants in Section 3.3.

Ideally, the researcher continues the analysis until *theoretical saturation* is achieved, which means that new data no longer meaningly impacts the theory. In this work, reaching saturation advances our previous publications [28,27]. Our criteria for saturation is described in Section 3.6. As an ever-evolving entity, rather than a finished product, new data can always be analyzed to alter or expand a grounded theory. Accordingly, practitioners could (and potentially will) adjust the theory when applying it to their concrete scenarios [14]. Therefore, in this work, we present an *emerging theory*, rather than a fully-validated one, which we explain more in Section 5.2.

We applied the GT techniques to data from interviews with IT professionals. In the next sections, we present how we chose our subjects and the design and analysis of these interviews.

3.2. Brainstorming sessions

After drafting our research questions, we conducted "brainstorming sessions" with seven specialists experienced with DevOps. Some of them have witnessed DevOps transformations in large organizations, while others have actively shaped such transformations in large and small companies.

The base script for these sessions aimed to elicit feedback on our research questions and spark discussion of concerns raised by our survey on the DevOps literature [26]. The conversations were essential for us to fine-tune our research questions and research approach. These sessions also helped us to better target the script for the following semi-structured interviews toward concerns learned from these experts. Therefore, the concrete outputs of this phase were: the research questions present in this text and the interview script (explained in Section 3.4).

We did not apply the analysis procedures detailed in Section 3.5 for these preliminary conversations, considering they were not intended to provide answers to our research questions. Nevertheless, we did not dismiss the theoretical insights provided by them. For example, the notion of a *platform team* began to take shape in the brainstorming sessions and, thus, influenced subsequent analysis. After these brainstorming sessions, we started the semi-structured interviews.

3.3. Selecting participants

We sent about 90 interview invitations using a convenience approach: the first invitations were close contacts in our research group's network. We also contacted participants suggested by our interviewees and colleagues. The only requirement was that the participant should work in an industrial context that has adopted continuous delivery or is implementing efforts toward it. Some invited participants did not reply, and some demonstrated interest in participating, but could not make time for it. Ultimately, we interviewed 37 IT professionals (≈ 41%). Following ethical procedures [29], all the interviewees and their organizations are anonymized in this paper. We recorded the interviews for later analysis, keeping the audio records under restricted access. We conducted the interviews from April 2019 to May 2020. Nine interviews were conducted in person, five of which took place at the interviewee's company, and 28 were held online. The sessions took 50 min on average (minimum of 24 and a maximum of 107 min).

We employed several strategies to foster diversity and enhance comparison possibilities in our sample, as recommended by GT guidelines [19]. We aimed to include a broad range of organization and interviewee profiles. For instance, we selected organizations of different sizes[3]: small (30%), medium (32%), and large (38%); of different types: private (90%), governmental (5%), and others (5%); and from different sectors and countries. We included male (73%) and female (27%) professionals, and also chose interviewees with different roles.

Table 1 describes the participants, presenting only an aggregated profile of participants to cope with anonymization [29,22]. Location refers to that of the interviewee's team; we had four participants working remotely for globally distributed teams. When describing roles, enabler team indicates a specialized technical team that supports developers, but without owning any services. For interviews with consultants, the number of employees refers to the size of the companies that contracted the consultants (and not the consultant's employers). The interviewees worked in the following business domains: IoT, finances, defense, public administration, justice, real estate, maps, education, Internet, big data, research, insurance, cloud, games, e-commerce, telecommunication, fashion, international relations, mobility, office

[2] Examples of questions guiding sociological inquiries in the GT context: "Does the incest taboo exist in all societies?", "Are almost all nurses woman?" [14].

[3] We employed 200 and 1000 as size thresholds because they are used in information publicly available on LinkedIn.

Table 1
Number and description of participants and organizations.

Role	**Team location**
17: Developer	21: Brazil
7: Development manager	5: USA
5: External consultant	4: Globally distributed
3: Infrastructure manager	2: Germany
2: Infrastructure engineer	2: Portugal
1: Executive manager	1: France
1: Enabler team member	1: Canada
1: Designer	1: Italy
Gender	**Number of employees**
27: Male	**in the organization**
10: Female	14: More than 1000
Time since graduation	12: From 200 to 1000
15: More than 10 years	11: Less than 200
13: From 5 to 10 years	**Organization type**
9: Less than 5 years	33: Private for profit
Degree	2: Governmental
22: Undergraduate	1: Private nonprofit
13: Masters	1: International organization
2: Ph.D.	

automation, software consulting, inventory management, vehicular automation, team management, and support to software development. Five of the interviewed companies are currently considered unicorn startups, and two of them are tech giants.

We also selected participants with theoretical purposes in mind, thus applying theoretical sampling. We interviewed participants who work in scenarios where it is particularly challenging to achieve continuous delivery (e.g., IoT, games, or defense systems), aiming to understand the limits and eventual corner cases. After the twentieth interview, we more actively sought people: in a cross-functional team, in (or interacting with) a platform team, with no (or few) automated tests, with monolithic systems, and labeled as "full-stack engineers". We adopted these criteria due to hypotheses not well-supported by our chain of evidence at that time.

To support our findings, we included excerpts from these conversations in our chain of evidence (in the accompanying supplementary material) and in this article. The excerpts are formatted in italics and within quotes. Excerpts and other accounts refer to interviews using tokens in the format "#*IN*". Thus, "#I2" refers to the second interview, interviewee, or interviewee's organization. Brainstorming sessions are indicated as "#B*N*". Such excerpts and citations are intended to make readers "feel they were also in the field", a GT recommendation [14].

3.4. Conducting the interviews

Since our goal is to discover existing organizational structures, and not to verify a preconceived set of structures in the field, it would be unsuitable to use only closed questions; instead, we conducted semi-structured interviews. Semi-structured interviews mix closed and open-ended questions, often accompanied by "why" and "how" questions; the interview can deviate from the planned questions, allowing for discussion of unforeseen issues [30], which fits the purpose of theory generation. With semi-structured interviews, we could also focus on different topics in different conversations, according to the relevance of each theme for each context.

Before starting the interviews, we built an interview protocol[4] to guide the process based on our previous experience with interviews [25], on other relevant works [21,30], and on the guidelines offered by a website for journalists, called ijnet.[5]

The interview protocol contains the questions that drove the interviews, which mainly were derived from the brainstorming sessions and our survey of the DevOps literature [26], and hence also grounded in data.[6] The themes addressed by the interview questions include: (1) interviewee company and role; (2) responsibility for deployment, building new environments, non-functional requirements, configuring and tracking monitoring, and incident handling, especially after-hours; (3) delivery performance (using the metrics and scales defined in Section 2); (4) future improvements in the organization; (5) effectiveness of inter-team communication; (6) inter-team alignment for the success of the projects; (7) description of DevOps team or DevOps role, if existing; and (8) the policy for sharing specialized people (e.g., security and database experts) among different teams.

The interview protocol is not a static document. As we conducted interviews, we changed how we asked some questions, focused more on some questions and less on others, and created new questions to explore rising hypotheses. We provide some indications about the evolution of the questions in the interview protocol itself.

3.5. Analyzing the interviews

We followed the core Grounded Theory principles of *constant comparative method* and *coding*, which are intended to discipline the creative generation of theory. During this process, we created two artifacts for each interview: the **transcripts** and the **codes**. We also created two global artifacts: the **comparison sheet** and the **conceptual framework**. Finally, by analyzing, comparing, and using all these artifacts, we elaborated our **taxonomy**, which is the theory itself.

Transcripts. We listened to each audio record and transcribed it. We did not transcribe the full interview. Instead, we summarized relevant parts, excluding minor details and meaningless noise [31]. For instance, we transcribed the following part of a conversation:

"If you break the SLA, there are consequences. You have to improve things; you can't go back to feature development until SLA has recovered. Any problem in final service: developer is paged. If it's infrastructure-related, developers call the infrastructure team. And we solve together. We try to help anyway, because at the end of the day if users can't use the system, we all suffer."

Codes. After transcribing an interview, we then derived the coding by condensing the interviewers' transcripts into a few words (the essence of the interview in relation to our research questions). Each interview has its list of coding, representing the particular reality of that interviewee. The above fragment of transcription, for example, led to the following coding:

Developers → own the availability of their services

Broken SLA → blocks feature development

Broken SLA → page developers

Broken SLA → if needed, call infra

The supplementary material "chain of evidence" presents more examples of the coding we performed.

The comparison sheet. To support the *constant comparison* of different interviews and codings, we summarized the main characteristics of each interview in a spreadsheet. We filled the cells with concise statements, with rows representing interviews and columns including the following characteristics: interview number, organizational structure, supplementary properties, delivery performance, observation, continuous delivery, microservices, cloud, other teams, non-functional requirements (NFRs), monitoring/on-call, alignment, communication, bottleneck, responsibility conflicts, database, security, specialists sharing, and DevOps team/role.

Conceptual framework. From the constant comparison of codes of different interviews emerged theoretical concepts. These theoretical

[4] http://ccsl.ime.usp.br/devops/2020-06-14/interview-protocol.html
[5] http://ijnet.org/en

[6] GT also considers the library as a source of data.

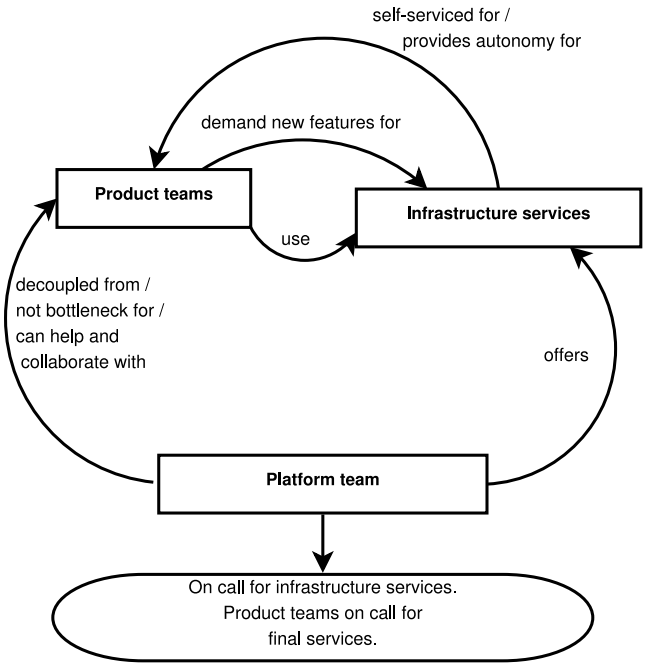


Fig. 1. A fragment of our conceptual framework.

concepts and their relations form the unified *conceptual framework*, which constitutes the shared understanding among the authors about the analyzed data [32]. Our conceptual framework is not a representation of our theory, but rather an intermediary artifact used to consolidate, in a single place, the concepts yielded by the coding process, working as the source of concepts to the shaping of our theory.

We maintained a visual representation of our conceptual framework as the research evolved. We provide all of its versions in the supplementary materials. Fig. 1 provides as example a fragment of our conceptual framework. In that figure, rectangles represent *concepts* abstracted from data, while rounded boxes represent *properties* of these concepts (i.e., an attribute or a characteristic of a concept).

As we evolved the conceptual framework and filled our comparison sheet, we developed our theory by classifying each interview by its organizational structure and its supplementary properties. The classification process was based on the concepts provided by the framework and on the analysis of similarities and differences between the interviews, summarized in the comparison sheet. As we evolved our understanding with new interviews, we revisited the classification of previous interviews to refine our theory. For example, after the emergence of a supplementary property, we checked whether we could classify previous interviews with the new property.

After some interviews, one author developed the first version of the coding lists, the comparison sheet, the conceptual framework, and the taxonomy. Based on the transcriptions, other two authors thoroughly reviewed these artifacts, triggering discussions that affected their evolution. When making a decision that could impact our taxonomy, we involved a fourth author. One example of how we evolved and enhanced our taxonomy is how we developed the supplementary properties after twenty interviews. Additional rounds of analysis, discussions, and theory elaboration were undertaken until the submission of this article. We went through this internal review process to reduce the bias of a single researcher performing analysis with preconceived ideas.

3.6. Theoretical saturation

We consider the size of our conceptual framework as a proxy for how much we have learned so far about our research topic. We define the size of the conceptual framework as the number of elements in the diagram representing it, which counts the number of concepts, conceptual properties, and links. Consider Fig. 2-(a): the x axis represents the number of interviews, while the y axis represents the number of elements in our conceptual framework. In this figure, we see that the last 15 interviews (40% of them) increased the size of our conceptual framework by only 9%. Moreover, the last three interviews did not increase the size of the conceptual framework to any degree. This suggests more interviews would provide only a negligible gain for our framework.

In addition to measuring the size of the conceptual framework, an intermediary artifact, we also measured the growth of the taxonomy itself, our final product. Fig. 2-(b) shows that in the first five interviews we discovered our four organizational structures. It also shows that by interview 22, we already had all but one of the supplementary properties. The last discovered supplementary property is an exceptional case, and it was applied only for one interview. Therefore, the decreasing growth of the taxonomy suggests that the last interviews contributed much less to shaping our theory.

By conjoining these two observations of Fig. 2, we claim to have reached enough of saturation for the purposes of our theory.

3.7. Feedback

GT aims to formulate a theory that has relevance for practitioners, so it is crucial to also investigate whether findings make sense to them [15]. Moreover, practitioners can help to identify taxonomy errors, such as inclusion and exclusion errors [15]. Therefore, we collected feedback on our taxonomy from the study participants, using an online survey.[7] The received feedback is presented in Section 5.1.

4. The taxonomy of organizational structures

In this section, we answer our research questions by presenting our taxonomy of the organization of development and infrastructure teams, including the organizational structures we identified, alongside their core and supplementary properties. Core properties are expected to be found in corporations with a given structure. When applicable, we use core properties to discuss delivery performance. Supplementary properties refine the explanation of a structure, but their association with organizations is noncompulsory.

For each interview, we classified the organizational structure observed. As the differentiation and integration patterns among development and infrastructure may vary for each deployable unit [33], it is not possible to assign a single structure to an organization. So the classification is applied according to the interviewee's context. Moreover, we observed in some cases a process of gradually adopting new structural patterns while abdicating from old ones; we classified this as transitions from one organizational structure to another.

Fig. 3 presents the discovered organizational structures, the primary elements of our taxonomy, alongside the supplementary properties, which qualify the elements pointed out by the arrows. The circles group supplementary properties that can be equally applied for a given element. Table 2 shows the classification (organizational structure and supplementary properties) applied for each interview, alongside the achieved delivery performance, and the number of employees in the corresponding organization.

Our comprehensive chain of evidence, added as supplementary material, indicates how much of our findings are backed by data we collected. The chain of evidence links each organizational structure and supplementary property to supporting coding, memos, and excerpts. Such linkage is critical for the credibility of qualitative research findings [34,15,35].

We now present each one of the organizational structures and their core and supplementary properties.

[7] http://ccsl.ime.usp.br/devops/2020-06-14/feedback-form.html

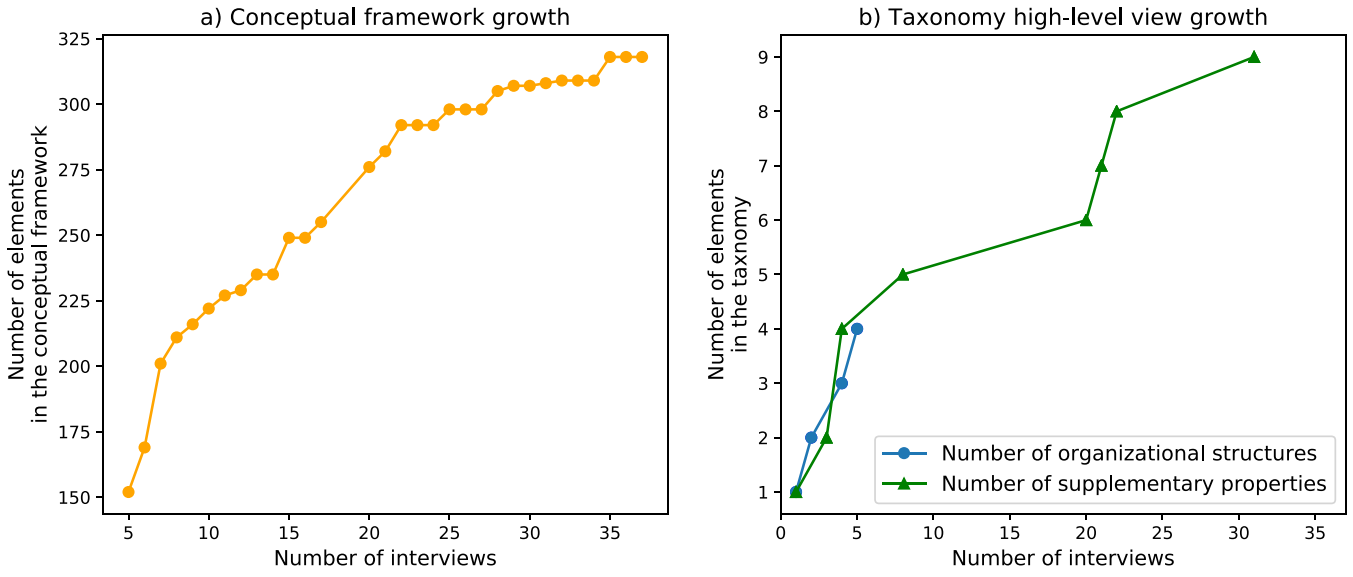


Fig. 2. Evidence of theoretical saturation.

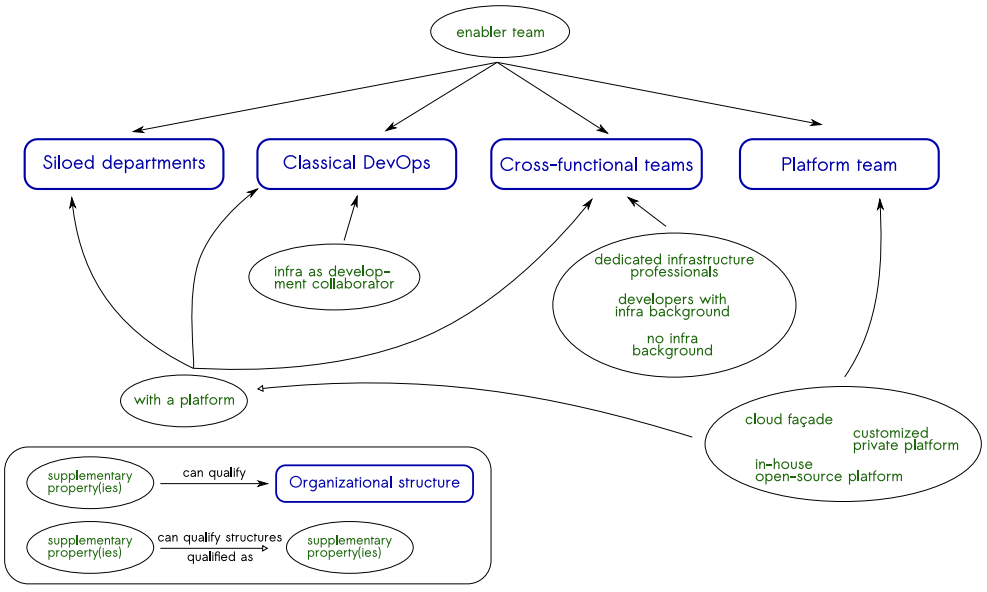


Fig. 3. High-level view of our taxonomy: discovered organizational structures and their supplementary properties.

4.1. Siloed departments

With siloed departments, developers and the infrastructure staff are segregated from each other, with little direct communication among them. Frictions occurs among silos, since developers want to deliver as much as possible, whereas operations target stability and block deliveries. The DevOps movement was born in 2008 [36] to handle such problems. We found seven organizations adhering to this structure, and six others transitioning out of this structure.

While supplementary properties did not emerge for this structure, we found seven core properties for corporations with siloed departments:

→ Developers and operators have **well-defined and differentiated roles**; as stated by #I20: *"the wall was very clear: after committing, our work [as developers] was done"*. Therefore, there are no conflicts concerning attributions. Well-defined roles and pipelines can decrease the need for inter-departmental direct collaboration (#I10).

→ **Each department is guided by its own interests**, looking for local optimization rather than global optimization, an old and problematic pattern [37]. Participant #I26 told us *"there is a big war there... the security, governance, and audit groups must still be convinced that the tool [Docker/Kubernetes] is good"*.

→ **Developers have minimal awareness of what happens in production** (#I26). So monitoring and handling incidents are mostly done by the infrastructure team (#I5).

→ **Developers often neglect non-functional requirements (NFRs)**, especially security (#I5). In #I30, conflicts among developers and the security group arise from disagreement on technical decisions. In other cases, developers have little contact with the security group (#I26).

→ **Limited DevOps initiatives,** centered on adopting tools, do not improve communication and collaboration among teams (#I30) or spread awareness about automated tests (#I5, #I15). In #I30, a "DevOps team" maintaining the deployment pipeline behaves as another silo, sometimes bottlenecking the delivery [38].

Table 2
Interview's classification. In the second column, "*to*" indicates transitioning from one structure to another.

Interview	Organizational structure	Supplementary properties	Delivery performance	Organization size
#I1	Cross-functional	With dedicated infra professionals	High	> 200 and < 1000
#I2	Classical DevOps		High	> 200 and < 1000
#I3	Cross-functional	Without infra background	Not-high	< 200
#I4	Platform team	Cloud façade With enabler team	High	> 1000
#I5	Siloed departments		Not-high	> 1000
#I6	Classical DevOps		Not-high	> 200 and < 1000
#I7	Siloed departments *to* Classical DevOps		Not-high	> 200 and < 1000
#I8	Siloed departments *to* Platform team	With a customized private platform	Not-high	> 1000
#I9	Platform team	Cloud façade With enabler team	High	> 200 and < 1000
#I10	Siloed departments		Not-high	< 200
#I11	Classical DevOps	With enabler team	Not-high	> 200 and < 1000
#I12	Platform team	With a customized private platform With enabler team	High	> 200 and < 1000
#I13	Siloed departments		Not-high	> 1000
#I14	Classical DevOps *to* Platform team	Cloud façade	Not-high	> 200 and < 1000
#I15	Siloed departments *to* Classical DevOps		Not-high	> 200 and < 1000
#I16	Cross-functional *to* platform team	With dedicated infra professionals With a customized private platform With enabler team	Not-high	> 1000
#I17	Classical DevOps	With enabler team	High	> 200 and < 1000
#I18	Classical DevOps	With enabler team	Not-high	> 1000
#I19	Siloed departments		Not-high	< 200
#I20	Siloed departments *to* Platform team	With an in-house open source platform	High	> 1000
#I21	Classical DevOps *to* Cross-functional	With developers having infra background	High	< 200
#I22	Classical DevOps	With a platform With a customized private platform	Not-high	> 1000
#I23	Siloed departments		Not-high	< 200
#I24	Siloed departments *to* cross-functional	With dedicated infra professionals	High	> 200 and < 1000
#I25	Cross-functional	With developers having infra background With a platform Cloud façade	Not-high	< 200
#I26	Siloed departments *to* Classical DevOps		Not-high	> 1000
#I27	Cross-functional	With dedicated infra professionals	Not-high	< 200
#I28	Cross-functional	With dedicated infra professionals	Not-high	> 200 and < 1000
#I29	Classical DevOps		High	< 200
#I30	Siloed departments	With enabler team With a platform With an in-house open source platform	Not-high	> 1000
#I31	Classical DevOps	infra as development collaborator With enabler team	Not-high	> 1000
#I32	Cross-functional	Without infra background	Not-high	< 200
#I33	Platform team	Cloud façade	High	> 1000
#I34	Classical DevOps		Not-high	< 200
#I35	Cross-functional	With dedicated infra professionals	Not-high	< 200
#I36	Classical DevOps		Not-high	> 1000
#I37	Siloed departments		Not-high	> 1000

→ **Organizations are less likely to achieve high delivery performance** as developers need bureaucratic approval to deploy applications and evolve the database schema (#I5, #I30). Table 3 shows that only two of 13 siloed organizations presented high delivery performance, and these two were already transitioning to other structures. However, we observed cases in which low delivery performance was not a problem, such as short-lived research experiments (#I13) and releases of new phases of a game not requiring code changes (#I10). Network isolation policies may also hinder frequent deployment (#B1, #I7).

→ We observed a **lack of proper test automation** in many organizations (#I5, #I15, #I23, #I26). In #I26, developers automate only unit tests. Organization #I15 was leaving test automation only for QA people, which is not suitable for TDD or unit tests. Although siloed organizations are not the only ones that lack test automation (#I3, #I32, #I35), in this structure developers can even ignore its value (#I5, #I23, #I37). We notice that some of the observed scenarios were more challenging for test automation, such as games.

4.2. Classical DevOps

The classical DevOps structure focuses on collaboration among developers and the infrastructure team. It does not eliminate all conflicts, but promotes a better environment to deal with them (#I34). We named this structure "Classical DevOps" because we understand that a collaborative culture is the core DevOps concern [23,26,39]. We classified ten organizations into this structure. We also observed three organizations transitioning to this structure and three transitioning out of this structure.

The eight core properties observed for organizations adopting classical DevOps are as follows:

Table 3
Organizational structures and delivery performance observed in our interviews.

Organizational structure	Delivery performance	Number of interviews
Siloed departments	Not-high	7
Classical DevOps	High	3
Classical DevOps	Not-high	7
Cross-functional	High	1
Cross-functional	Not-high	6
Platform team	High	4
Siloed departments *to* Classical DevOps	Not-high	3
Siloed departments *to* Cross-functional	High	1
Siloed departments *to* Platform team	High	1
Siloed departments *to* Platform team	Not-high	1
Classical DevOps *to* Cross-functional	High	1
Classical DevOps *to* Platform team	Not-high	1
Cross-functional *to* Platform team	Not-high	1

→ We observed that, in classical DevOps settings, many practices foster **a culture of collaboration**. We saw the sharing of database management: infrastructure staff creates and fine tunes the database, whereas developers write queries and manage the database schema (#I17). We heard about open communication among developers and the infrastructure team (#I2, #I6, #I17, #I22, #I31, #I36). Participant #I2 highlighted that: *"Development and infrastructure teams participate in the same chat; it even looks like everyone is part of the same team"*. Developers also support the product in its initial production (#I31).

→ Roles remain well-defined, and despite the collaboration on some activities, there are usually **no conflicts over who is responsible for each task**.

→ Developers feel relieved when they can rely on the infrastructure team (#I17). Participant #I31 claimed that his previous job in a cross-functional team had a much more stressful environment than his current position in a development team in a classical DevOps environment. On the other hand, **stress can persist at high levels for the infrastructure team** (#I34), especially *"if the application is ill-designed and has low performance"* (#I36).

→ In this structure, the project's success depends on the **alignment of different departments**, which is not trivial to achieve. In #B3, different teams understood the organization's goals and the consequences of not solving problems, like wrongly computing amounts in the order of millions of dollars. Moreover, #I7 described that alignment emerges when employees focus on problem-solving rather than role attributions.

→ **Development and infrastructure teams share NFR responsibilities** (#17). For example, in #I2, both were very concerned with low latency, a primary requirement for their application.

→ Usually, **the infrastructure staff is the front line of tracking monitoring and incident handling** (#I2, #I11, #I29, #I31, #I36). However, if needed, developers are summoned and collaborate (#I17, #I34). In #I34, monitoring alerts are directed to the infrastructure team but copied to developers. However, in some cases developers never work after-hours (#I2, #I22).

→ Humble expects a culture of collaboration among developers and the infrastructure staff to **prescind from a "DevOps team"** [38]. We understand this criticism applies to DevOps teams with dedicated members, such as we saw in #I30, since they behave as new silos. However, we found in #I36 a well-running DevOps team working as a committee for strategic decisions — a forum for the leadership of different departments. We also found DevOps groups working as guilds (#I4, #I8), supporting knowledge exchange among different departments [40].

→ Collaboration and delivery automation, critical values of the DevOps movement, are **not enough to achieve high delivery performance**. Of 10 classical DevOps organizations not transitioning from or to other structures, only three presented high delivery performance (Table 3). One possible reason is the lack of proper test automation (#I22, #I36) [41]. Another limitation for delivery performance is the adoption of release windows (#I11, #I31, #I14, #I36), which seek to mitigate deployment risk by restricting software delivery to periodic time slots. Release windows are adopted by considering either the massive number of users (#I31) or the system's financial criticality (#I36). Release windows may also result from fragile architectures (#I37) or the monolith architectural style (#I11) since any deployment has an increased risk of affecting the whole system.

Supplementary properties

For classical DevOps organizations, we found one supplementary property that we describe in the following.

Infra as development collaborator. The infrastructure staff contributes to the application code to optimize the system's performance, reliability, stability, and availability. Although this aptitude requires advanced coding skills from infrastructure professionals, it is a suitable strategy for maintaining large-scale systems, like the ones owned by #I31.

4.3. Cross-functional teams

In our context, a cross-functional team takes responsibility for both software development and infrastructure management. This structure aligns with the Amazon motto "*You built it, you run it*" [42] and with the "autonomous squads" at Spotify [40]. This gives more freedom to the team, along with a great deal of responsibility. As interviewee #I1 described: *"it is like each team is a different mini-company, having the freedom to manage its own budget and infrastructure"*. We found seven organizations with this structure, two organizations transitioning to this structure, and one transitioning out of it.

The four core properties found for cross-functional teams are as follows:

→ **Independence among teams may lead to misalignment.** Lack of communication and standardization among cross-functional teams within a single organization may lead to duplicated efforts (#I28). However, this is not always a problem (#I1).

→ **It is hard to ensure a team has all the necessary skills.** For instance, we interviewed two cross-functional teams with no infrastructure expertise (#I3, #I32). Participant #I27 recognizes that *"there is a lack of knowledge"* on infrastructure, deployment automation, and monitoring. A possible reason for such adversity is that, as #I29 taught us, it is hard to hire infrastructure specialists and senior developers.

→ We expected cross-functional teams to provide too much idle time for specialists, as opposed to centralized pools of specialization. However, we find **no evidence of idleness for specialists**. From #I16, we heard quite the opposite: the infrastructure specialists were too busy to be shared with other teams. Having the infrastructure specialists code features in their spare time avoids such idleness (#I35).

→ **Most of the cross-functional teams we interviewed were in small organizations** (Table 4), likely because there is no sense in creating multiple teams in too small organizations.

Supplementary properties

Dedicated infra professionals. The team has specialized people dedicated to infrastructure tasks. In #I1, one employee specializes in physical infrastructure, and another is "the DevOps", taking care of the deployment pipeline and monitoring. In this circumstance, the infrastructure specialists become the front-line for tackling incidents and monitoring (#I28, #I35).

Developers with infra background. The team has developers knowledgeable in infrastructure management; these professionals are also called full-stack engineers or even DevOps engineers (#I25). Participant #I25 is a full-stack engineer and claimed to *"know all the*

Table 4
Organizational structures and organization size observed in our interviews.

Organizational structure	Organization size	Number of interviews
Siloed departments	< 200	3
Siloed departments	> 1000	4
Classical DevOps	< 200	2
Classical DevOps	> 200 and < 1000	4
Classical DevOps	> 1000	4
Cross-functional	< 200	5
Cross-functional	> 200 and < 1000	2
Platform team	> 200 and < 1000	2
Platform team	> 1000	2
Siloed departments *to* Classical DevOps	> 200 and < 1000	2
Siloed departments *to* Classical DevOps	> 1000	1
Siloed departments *to* Cross-functional	> 200 and < 1000	1
Siloed departments *to* Platform team	> 1000	2
Classical DevOps *to* Cross-functional	< 200	1
Classical DevOps *to* Platform team	> 200 and < 1000	1
Cross-functional *to* Platform team	> 1000	1

involved technologies: front-end, back-end, and infrastructure; so I am the person able to link all of them and to firefight when needed". Participant #I29, a consultant, is skeptical regarding full-stack engineers and stated that *"these people are not up to the task"*. He complained that developers are usually unaware of how to fine tune the application, such as configuring database connections.

No infra background. Product teams manage the infrastructure without the corresponding expertise. We saw this pattern in two places. One was a very small company and had just released their application, having only a few users (#I32) and being uncertain about hiring specialized people soon. Interviewee #I3 understands that operations work (e.g., spotting errors during firmware updates in IoT devices and starting Amazon VMs for new clients) is too menial for software engineers, taking too much of their expensive time. So the organization was planning the creation of an operations sector composed of a cheaper workforce. Interviewee #I19 argued that such an arrangement could not sustain growth in his company in the past.

4.4. Platform teams

Platform teams are infrastructure teams that provide highly automated infrastructure services that can be self-serviced by developers for application deployment. The infrastructure team is no longer a "support team"; it behaves like a product team, with the "platform" as its product and developers as internal customers. In this setting, infrastructure specialists need coding skills; product teams have to operate their business services; and the platform handles much of the non-functional concerns. We found four organizations fully embracing this model and four others in the process of adopting it. We also observed the platform team pattern in three of the brainstorming sessions.

The core properties of platform teams are as follows:

→ **Product teams are fully accountable for the non-functional requirements of their services.** They become the first ones called when there is an incident, which is escalated to the infrastructure people only if the problem relates to an infrastructure service (#I8, #I9, #I12, #I33).

→ Although the product team becomes fully responsible for NFRs of its services, it is not a significant burden that developers try to refuse (#I33). **The platform itself handles many NFR concerns**, such as load balancing, auto-scaling, throttling, and high-speed communications between data-centers (#I4, #I8, #I16, #I33). As participant #I33 told us, *"you do not need to worry about how things work, they just work"*. Moreover, we observed infrastructure people willingly supporting developers for the sake of services availability, performance, and security (#I9, #I14).

→ **Product teams become decoupled from the members of the platform team.** Usually, the communication among these teams happens when developers and infrastructure people gather to solve incidents (#I8, #I9); when infrastructure people provide consulting for developers to master non-functional concerns (#I9); or when developers demand new capabilities from the platform (#I8, #I12). In this way, the decoupling between the platform and product teams does not imply the absence of collaboration among these groups.

→ **The infrastructure team is no longer requested for operational tasks.** The operational tasks are automated by the platform. Therefore, one cannot merely call platform-team members "operators", since they also engineer the infrastructure solution. We remark that, in other industries, "operator" is a title attributed to menial workers.

→ The platform **avoids the need for product teams to have infrastructure specialists**. Participant #I33 expressed wanting to better understand what happens *"under the hood"* of the platform, which indicates how well the platform relieves the team from mastering infrastructure concerns. On the other hand, since developers are responsible for the deployment, they must have some basic knowledge about the infrastructure and the platform itself.

→ **The platform may not be enough to deal with particular requirements.** Participant #I16 stated that *"if a lot of people do similar functionality, over time usually it gets integrated to the platform... but each team will have something very specialized..."* to explain the presence of infrastructure staff within the team, even with the usage of a platform, considering the massive number of virtual machines to be managed.

→ If the organization develops a new platform to deal with its specificities, **it will require development skills from the infrastructure team**. Nevertheless, even without developing a new platform, the infrastructure team must have a "dev mindset" to produce scripts and use infrastructure-as-code [43] to automate the delivery path (#I14). One strategy we observed to meet this need was to hire previous developers for the infrastructure team (#I14).

→ All four organizations that have fully embraced the **platform team structure are high performers**, while no other structure provided such a level of success (Table 3). An explanation for such a relation is that this structure decouples the infrastructure and product teams, which prevents the infrastructure team from bottlenecking the delivery path. As stated by #I20: *"Now developers have autonomy for going from zero to production without having to wait for anyone"*. This structure also contributes to service reliability by letting product teams handle non-functional requirements and incidents.

→ In Table 4, among the interviewed organizations with platform teams, none had less than 200 employees. Since assembling a platform team requires dedicated staff with specialized knowledge, it makes sense that such a structure is **not suitable for small companies**.

Supplementary properties

The description of a platform can be refined by applying one of the following supplementary properties:

Cloud façade. The platform ultimately deploys applications on public clouds, such as Amazon WS, Google Cloud, or Azure. Although these clouds allow easier deployment when compared to managing physical servers, they still offer dozens of services and a multitude of configurations. The in-house platform standardizes the usage of public cloud vendors within the organization, so developers do not need to understand many details about the cloud (#I4, #I14, #I33); therefore *"enhancing the usability of the [cloud] infrastructure"* (#I16).

Customized private platform. The platform is built on top of internal physical servers (#B1, #I1, #I8, #I12, #I20), hiding infrastructure complexities from developers, such as the use and even the existence of Kubernetes, an open-source platform used for managing the lifecycle of Docker containers.

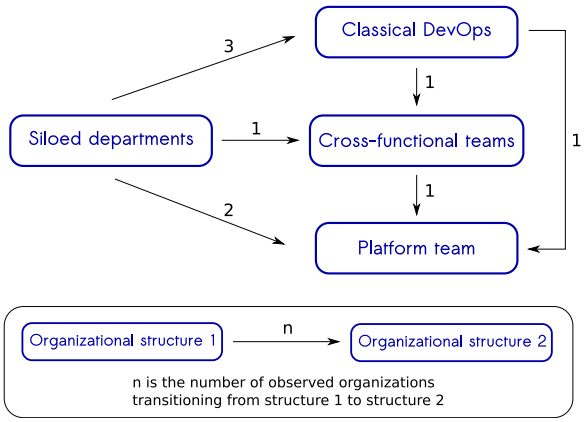


Fig. 4. Observed transition flows.

In-house open-source platform. The platform is an open-source software deployed on-premise. Organization #I20 uses Rancher,[8] a graphical interface for developers to interact with Kubernetes.

4.5. Shared supplementary properties

This section presents the found supplementary properties that are not linked to one organizational structure only. These properties are relevant since they are shared among multiple organization structures, as depicted in Fig. 3.

Enabler team. An enabler team provides consulting and tools for product teams but does not own any service. Consulting can be on performance (#I18) or security (#I9, #I16, #I31), for example. Tools provided by enabler teams include the deployment pipeline (#I4, #I30), high-availability mechanisms (#I11), monitoring tools (#I12), and security tools (#I17). We found them in every organizational structure. We learned the term "enabler team" during our interview with #I11.

With a platform. The organization possesses a platform that can provide deployment automation, but not following the patterns of human interaction and collaboration described by the core properties of platform teams. Participant #I25 developed an *"autonomous IaaC for integration and deployment with Google Cloud"*, which provides a platform's capabilities to other developers of the team. However, since in this context there is a single cross-functional team, it cannot be called a "platform team". We classified organization #I30 as a siloed structure, even with a platform team, since developers and the platform team have a conflicted relationship. The supplementary properties of platform teams can also be applied to organizations with a platform.

4.6. Transitioning

Organizations are not static. We identified nine of them transitioning from one structure to another. Considering the transition flows in Fig. 4, we perceive that *(i)* no organization is transitioning *to* siloed departments, *(ii)* most of the transitions are *from* siloed departments, and *(iii)* no organization is transitioning *out of the* platform team. These observations agree with our theoretical considerations about the problems of siloed structures and the promises of platform teams.

Nonetheless, transitioning organizational structures is a hard endeavor, as confirmed by some interviewees. Although his organization did an excellent job transitioning to a platform structure and achieving high-delivery performance, interviewee #I20 claims that the *"old world"* still coexists with the *"new one"*. In the same way, as reported by Nybom et al. [6], there are some responsibility conflicts and *"dissident forces"*: some operations personnel do not like developers with administrative powers, while some developers do not want such powers. The interviewee declared that *"it is not yet everybody together"*.

Similarly, interviewee #21 stated that *"There are two worlds... one was born in the cloud, and it is nice that it influences the legacy system to become more robust. There are many worlds we wish to bring together. However, we need to rewrite even the culture; we must reset everything"*. These examples also show how culture is a crucial factor for change.

4.7. Summary

We close the description of our taxonomy by highlighting the key differences among its organizational structures. Table 5 summarizes, for each structure, *(i)* the differentiation between development and infrastructure groups regarding operations activities (deployment, infrastructure setup, and service operation in run-time); and *(ii)* how these groups interact (integration).

5. Discussion

In this section, we discuss feedback sessions and our emerging theory in light of the received feedback.

5.1. Feedback

We sent to each one of the 37 first interviewees a feedback form asking whether the interviewee agreed with the chosen classification (organizational structure and supplementary properties) for its context using the following Likert scale: strongly agree, weakly agree, I do not know, weakly disagree, strongly disagree. In case of disagreement, there were free text fields for explanation. We also asked the interviewees whether they perceived our taxonomy as comprehensive and whether they would add or remove elements. Finally, we also left a free field for general comments. We sent the form in four batches of twenty, five, five, and seven emails spread across the last five months of our interviewing period; we used the feedback incrementally to refine our theory. We also attached to the emails a digest describing our taxonomy.

We received 11 answers. Nine participants strongly agreed with the received classification regarding the organizational structure, while two of them weakly agreed with it. No one disagreed. Five participants strongly agreed with the received classification regarding the supplementary properties, while one of them weakly agreed with it. Regarding our model's comprehensiveness, seven participants strongly agreed with it, three of them weakly agreed with it, and one of them did not know. This result suggests resonance of the participants with our theory, which refers to the degree to which findings make sense to participants.

The free-text answers from participants were valuable in refining the taxonomy. We conceived the supplementary properties from the analysis of the first round of feedback. One interviewee's comments helped us improve our taxonomy digest (we refined a figure to better express the idea of platform team). Moreover, some participants raised concerns about how different parts of the organization act under different patterns, and how this evolves. We considered such concerns since our classification is not for the whole organization, but for the interviewee's context at a point in time.

5.2. Evaluating our emerging theory

According to Ralph, good taxonomies should increase cognitive efficiency and assist reasoning by facilitating more inferences [15]. However, evaluating whether a taxonomy satisfies such quality criteria demands a different type of research approach than we employed; e.g., a case study [34]. Although case studies can test theories [44],

[8] http://rancher.com

Table 5
Summary of organizational structures.

Organizational structure	Development differentiation	Infrastructure differentiation	Integration
Siloed departments	Just builds the application package	Responsible for all operations activities	Limited collaboration among the groups
Classical DevOps	Participates/collaborates in some operations activities	Responsible for all operations activities	Intense collaboration among the groups
Cross-functional teams	Responsible for all operations activities	Does not exist	–
Platform teams	Responsible for all operations activities With the platform support	Provides the platform automating much of the operations activities	Interaction happens for specific situations, not on a daily basis

they are more useful for demonstrating how a current theory is incomplete or inadequate to explain the observed case [12,11]. A single case study does not prove a social theory. Moreover, even after decades of robust tests confirming theories, new work can still expand them [13].

On the other hand, grounded Theory (GT) focuses on generating theory rather than validating preconceived hypotheses. Although researchers can be tempted to try to validate their theory as soon as it is born, Glaser and Strauss caution that verificational approaches hinder theory development too early [14]. Therefore, in this paper, we present only incipient steps related to theory assessment: hearing feedback on our taxonomy (Section 5.1) and comparing it to other existing classifications (Section 6). Nonetheless, such steps evidence the resonance of our theory.

By following Guba's framework for naturalistic research evaluation [35], practitioner and literature resonance also provide some credibility (internal validity, how plausible or true the findings are) and confirmability (objectivity, opportunities for correcting research bias). The remaining of Guba's criteria are dependability (reliability), which is provided by our chain of evidence, and transferability (external validity or generalizability), which is supported by our diverse selection of participants.

Though it is inadequate to evaluate our emerging theory as a finished product, we can *evaluate our process of generating the theory* [14,15]. This can be done by checking adherence to GT prescriptions and Ralph's recommendations for taxonomy generation [15]. We described the adherence to GT guidelines (i.e., theoretical sampling, coding, theoretical sensitivity, and theoretical saturation) in Section 3. In the following, we discuss our adherence to Ralph's recommendations.

A warning from Ralph is that theory should explain, not prescribe. In this way, it is essential to note that although our theory is intended to be used by practitioners in practical settings, as is the goal of a grounded theory, the theory itself provides an explanation of the world, not a guide for action.

A more severe impact on our work comes from Ralph's advice for favoring first-hand observations over interviews due to interviewees' biases. Although we acknowledge his concerns, in our case organizational structures are too abstract to grasp only by observation, even meetings observation, without further conversations with observed people. Our context thus differs from other software engineering situations, such as observing a pair-programming session. Nonetheless, anonymity reduces these biases by favoring an open attitude from the participants. We took care not to ask the research questions directly to participants, which would force our preconceptions onto them [15]. We carefully crafted second-level questions [34], which are more objective than the research questions. Our interview protocol[9] presents each interview question, followed by its rationale.

We acknowledge the importance of triangulating results with other kinds of data and with other participants in the same organizations. However, we spare these strategies for the future phase of our research. Observing more scenarios is more valuable for this initial phase of theory elaboration than interviewing more people in each organization, which would lead to interviews in fewer companies.

Ralph still cautions that researchers should take care in selecting evaluation criteria to assess a theory development. Since multiple criteria exist and there is no universally accepted set of criteria, researchers must choose criteria that make sense for the emerging theory [15]. Nonetheless, we apply to our research three crucial criteria suggested by Ralph: *(i)* the empirical study is well-executed and clearly described, *(ii)* there is an explicit chain of evidence from results back to supporting data (which does not exclude the existence of non-replicable intuitive leaps [15]), and *(iii)* the need for the proposed theory must be clear. The present text and the complete chain of evidence, provided as supplementary material, must suffice to the reader to judge these concerns.

6. Related work

Recent research has discussed the benefits and challenges of continuous delivery [3,45,46,1,47]. Among the challenges, Chen et al. include organizational issues related to tensions among groups within an organization [3], which demands studies on the organization of teams. Some of these studies have focused on how developers migrate from teams or companies [48,22,49]. Nonetheless, the literature about the inter-team arrangements for managing IT infrastructure in a continuous delivery context is still limited. In the following, we discuss this existing literature [2,6–10] by comparing their classifications of inter-team relations to our taxonomy.

In one of the foundational writings on DevOps [2], Humble and Molesky start by criticizing the siloed department structure and management by project. They then follow by advocating cross-functional teams and management by product. However, they also suggest practices for strengthening collaboration among development and operations, which makes sense in the classical DevOps structure. Such practices include operators attending agile ceremonies and developers contributing to incident solving. Humble and Molesky also envision operation groups offering support services (e.g., continuous integration) and infrastructure as a service to product teams, which relates in our taxonomy to enabler teams and the platform team structure.

Nybom et al. present three distinct approaches to DevOps adoption [6]: (i) assigning development and operations responsibilities to all engineers; (ii) composing cross-functional teams of developers and operators; and (iii) creating a DevOps team to bridge development and operations. However, the article is about a case study matching the first approach only; developers undertook operational tasks, and collaboration was promoted among the development and operations departments. According to our taxonomy, such a scenario was an attempt to migrate from siloed departments to classical DevOps. However, despite some perceived benefits, new sources of friction emerged among the departments, and several employees disagreed with the adopted approach. We associate these sub-optimal results to the reported lack of automation investments, which suggests that trying any DevOps adoption without aiming for continuous delivery is not promising.

The 2018 State of DevOps Report surveyed respondents about the organizational structures used in their DevOps journeys [7], offering

[9] http://ccsl.ime.usp.br/devops/2020-06-14/interview-protocol.html

a closed set of alternatives: *cross-functional teams responsible for specific services or applications*, *dedicated DevOps teams*, *centralized IT teams with multiple application development teams*, *site reliability engineering teams*, and *service teams providing DevOps capabilities (e.g., building test environments, monitoring)*. However, the text does not further describe such options. Thus, associating our structures to the options presented by the survey would be an error-prone activity.

Skelton and Pais present nine "DevOps topologies" and seven anti-patterns [8], as the most informal of our comparison sources – a blog post. The presentation of each topology and anti-pattern is short, lacking details about how corporations apply them. We now present the correspondences among the DevOps topologies/anti-patterns and our taxonomy. *Dev and ops silos* corresponds to our siloed departments structure. *Dev do not need ops* corresponds to our cross-functional teams with no infra background. In some organizations adopting classical DevOps, we saw the rebranding of the infrastructure team to SRE or DevOps (#I2, #I6, #I31), but this situation did not entirely match the *rebranded sysadmin* anti-pattern since there were cultural changes in the observed cases. *Ops embedded in dev team* corresponds to our cross-functional teams with dedicated infra professionals, although we saw positive results with this configuration in #I1. In a siloed organization (#I30), we also observed the *DevOps team silo*. *Dev and ops collaboration* corresponds to our classical DevOps structure. *Ops as infrastructure-as-a-service (platform)* corresponds to our platform teams. We consider *SRE team* topology to match our classical DevOps structures with infra as development collaborator, although the topology description does not include this SRE activity of coding the application to improve its non-functional requirements [50].

The *Team Topologies* book [9], from the same authors of the DevOps topologies blog post, presents four dynamic patterns of teams for software-producing corporations:
stream-aligned teams, delivering software in a business-aligned constant flow; *complicated sub-system teams*, with highly specialized people working on a complicated problem; *enabler teams*, providing consulting for stream-aligned teams in a specific technical or product domain; and *platform teams*, providing internal services for self-service by stream-aligned teams, abstracting infrastructure and increasing autonomy for stream-aligned teams.

The *stream-aligned team* corresponds to what we call a product team or development team in this paper. The *complicated sub-system team* is not considered in our taxonomy since it is related to splitting work within development only. The *enabler team* proposed by Skelton and Pais is very close to what we also call an enabler team (e.g., interviewee #I18 was in an enabler team providing consulting on performance for product teams); however, Skelton and Pais advocate that such consulting must be time-bounded, which is an aspect absent from our observed enabler teams. Finally, the *platform team* proposed by Skelton and Pais includes our notion of platform team; however, our concept is restricted to the services related to the execution environment of the applications, i.e., while we considered teams providing pipeline services as enabler teams, Skelton and Pais would consider them as platform teams.

Another significant difference from team topologies to our work is that the book seeks to present *things how they should be*, while we try to summarize *thing how they are*. In this sense, although we acknowledge the existence of the *classical DevOps* structure, it is not included in the team topologies, since the authors discourage the handover it causes. We also note that the terms "platform team" and "enabler team" emerged from our interviewees, without the Teams Topologies book's direct influence.

Most of the discussed work so far [6,2,7,8] presents sets of organizational structures without an empirical elaboration of how such sets were conceived. The *Team Topologies* book suggests that the proposed topologies emerged from field observations, but lacked a scientific methodology. In this way, Shahin et al. [10] present the closest work to ours, with a set of structures based on field data and scientific guidelines.

Shahin et al. [10] conducted semi-structured interviews in 19 organizations and surveyed 93 practitioners to empirically investigate how development and operation teams are organized in the software industry toward adopting continuous delivery practices. They found four types of team structures: *(i) separate Dev and Ops teams with higher collaboration*, *(ii) separate Dev and Ops teams with a facilitator(s) in the middle*; *(iii) small Ops team with more responsibilities for Dev team*, and *(iv) no visible Ops team*. Structures *i* and *iii* map to classical DevOps in our taxonomy. Structure *ii* corresponds to the adoption of a DevOps team as a bridge between development and operations. One of our interviewees reported that such a pattern occurred in the past in his organization (#I4) and we also observed the DevOps team as a committee bringing together development and operations leadership in another organization (#I36); therefore, DevOps teams serving as bridges are likely no longer common. Finally, structure *iv* maps to cross-functional teams. Shahin et al. did not identify the platform teams structure, the most promising alternative we found in relation to delivery performance. Moreover, their work does not present the notion of transitioning between the structures.

Shahin et al. [10] also explored the size of the companies adopting each structure. They found that structure *i* is mainly adopted by large corporations, while structure *iv* was observed mainly in small ones. These findings are corroborated by our data in Table 4: classical DevOps was less observed in small organizations, while cross-functional teams were not prevalent in large organizations. However, other factors may be involved in adopting an organizational structure. Better understanding these factors and their forces are in our plans for future work.

In a recent paper, Shahin and Babar recommend adding operation specialists to the teams [41], which favors cross-functional teams with dedicated infra professionals.

7. Limitations

The reader must take into account the typical limitations of taxonomy theories. The most important limitation is that they rarely make probabilistic predictions in terms of dependent and independent variables, as variance theories, common in Physics, do [15]. Thus, it is not proper to discuss, for a grounded theory, the number of interviewees in terms of statistical sampling.

A limitation for credibility and dependability [35] is the anonymity of our participants, since it is not possible to replicate the interviews with the same people. However, there is a crucial trade-off here: some participants might not accept an invitation to participate in a non-anonymous interview. Moreover, anonymity copes with social desirability bias [51]; interviewing the same person after some time does not guarantee the same responses.

Although we relied on previous work [18] to adopt the delivery performance construct, we needed to adapt the original thresholds due to the reasons exposed in Section 2. We also did not differentiate low from medium performers, as we judged that such differentiation would not help distinguish how the structures contribute to delivery performance, partly because low and medium performers are much more similar among themselves than when compared to high performers [52]. Therefore, by changing some decisions related to delivery performance, another researcher could reach different conclusions based on the same data. We handled this concern mainly by stating our definitions regarding this topic. Nonetheless, GT does not guarantee that two researchers working in parallel with the same data would achieve identical results [14].

In an ideal process, different researchers would analyze the interviews independently, avoiding some possible agreement bias favored by reviews. However, the cost for independent analysis would be too onerous (analyzing one interview takes hours). Regarding the review

process, instead of relying on the transcripts produced by a single author, all researchers could have listened to the original records. Yet, such an approach also would be too costly in terms of time commitment. Alternatively, two co-authors took part in a few initial interviews to standardize the transcription procedures and assess how the first author conducted these interviews.

We are aware that large companies usually present groups at different maturity levels, and that such groups could be classified differently if we had chosen different interviewees. To verify this, we interviewed two persons from the same company (#I16 and #I18) working in different teams. We noted that, indeed, the organizational patterns were not identical. This effect also happens when transitioning from one structure to another, since transitioning can be a long process and take different paces at different organization segments. Therefore, the reader must note that our descriptions characterize our respondents' contexts, not the organizations in their totality.

Finally, one particular characteristic of software development teams is that they are not fixed; developers often move in and out of a team. Therefore, there is no reason to think about the proposed organizational structures as immutable. Instead, readers should consider our organizational structures when moving toward a continuous delivery scenario. After becoming comfortable and starting to excel with continuous delivery, if needed, practitioners could adapt the organizational structure employed to become effective in other contexts.

8. Conclusion

In this paper, we presented an emerging grounded theory addressing the research question "*What organizational structures do software-producing organizations adopt to structure operations activities (application deployment, infrastructure setup, and service operation in run-time) among development and infrastructure groups?*" We found four organizational structures:

1. Siloed departments (with seven core properties);
2. Classical DevOps (with eight core properties and one supplementary property);
3. Cross-functional teams (with four core properties and three supplementary properties);
4. Platform teams (with nine core properties and three supplementary properties).

Beyond the seven supplementary properties directly related to one structure each, we also found two other supplementary properties applicable to more than one structure. The core properties describing each organizational structure plus the supplementary properties answer the sub-research question "*What are the properties of each of these organizational structures?*"

Moreover, this study answers the sub-research question "*Are some organizational structures more conducive to continuous delivery than others?*" by raising the emerging hypothesis that the **platform team** structure is more suitable for achieving continuous delivery given its relationship with delivery performance. We also observed that siloed departments relate more to organizations with less-than-high performance. Additionally, there was no apparent relation between delivery performance and the other structures (classical DevOps and cross-functional teams). Nonetheless, we critically state that reaching high delivery performance is not a need for every software-producing organization, as adopting platform teams is not adequate for every organization. A promising topic for future research is investigating whether the organization domain influences the need of high delivery performance.

Our work has implications for **practice**. Software companies could realize that there are several kinds of organizational structures they could adopt to excel in continuous delivery and plan which organizational structure they are interested in moving to, maximizing their chances to succeed in the transition. Further, we clarified the roles that the participants have to play in each organizational structure. This evidence could help practitioners cooperate with less friction toward organizational transformation.

Our work also has implications for **research**. The elements of our taxonomy, in Fig. 3, provide a common vocabulary to support the formulation of new research questions. For example, researchers can investigate the impact of each property on other perspectives (e.g., software architecture, security, database management). Another valuable endeavor would be to investigate the relation between our taxonomy's elements and the internal organization of development and operation groups, especially platform teams.

In addition, we observed that test automation is still not adequately practiced in many software companies, as also noted by Olsson et al. [46], and that the lack of tests is a limiting factor for achieving high delivery performance. This suggests research opportunities for proposing automated test generation techniques, especially in environments in which tests are intrinsically difficult, such as games or IoT. Moreover, we noticed participants practicing DevOps while maintaining a monolithic application, which contrasts with the literature that strongly associates DevOps with microservices. This suggests researchers should further investigate the differences in applying DevOps to monoliths and microservices-based systems. Similarly, since we also observed some participants maintaining a monolith core with peripheral microservices and achieving high delivery performance with the microservices, researchers could propose novel techniques and tools that could (semi-) automatically extract peripheral services from monolith applications.

For future work, we plan to better delineate the forces that drive organizations to choose different structures. We also intend to employ our taxonomy to discuss scenarios of corporations not visited by us yet, which can corroborate the theory's resonance.

CRediT authorship contribution statement

Leonardo Leite: Conceptualization, Methodology, Investigation, Resources, Data curation, Writing - original draft, Writing - review & editing, Visualization, Project administration. **Gustavo Pinto:** Resources, Data curation, Writing - review & editing, Visualization. **Fabio Kon:** Conceptualization, Methodology, Resources, Writing - review & editing, Visualization, Supervision, Funding acquisition. **Paulo Meirelles:** Conceptualization, Methodology, Resources, Data curation, Writing - review & editing, Visualization, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the support of the Brazilian Service of Federal Data Processing (Serpro), CNPq proc. 309032/2019-9 and proc. 465446/2014-0, CAPES – Finance Code 001, and FAPESP proc. 14/50937-1 and proc. 15/24485-9.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.infsof.2021.106672.

References

[1] G. Schermann, J. Cito, P. Leitner, H.C. Gall, Towards quality gates in continuous delivery and deployment, in: 24th IEEE International Conference on Program Comprehension, ICPC, 2016, pp. 1–4, http://dx.doi.org/10.1109/ICPC.2016.7503737.

[2] J. Humble, J. Molesky, Why enterprises must adopt DevOps to enable continuous delivery, Cut. IT J. 24 (8) (2011) 6–12.

[3] L. Chen, Continuous delivery: Huge benefits, but challenges too, IEEE Softw. 32 (2) (2015) 50–54, http://dx.doi.org/10.1109/MS.2015.27.

[4] G. Schermann, J. Cito, P. Leitner, U. Zdun, H.C. Gall, An empirical study on principles and practices of continuous delivery and deployment, PeerJ PrePrints 4 (2016) e1889, http://dx.doi.org/10.7287/peerj.preprints.1889v1.

[5] N. Oliveira, N. Takahashi, Organizational structure, format, shape design and architecture, in: Automated Organizations: Development and Structure of the Modern Business Firm, Springer, 2012.

[6] K. Nybom, J. Smeds, I. Porres, On the impact of mixing responsibilities between devs and ops, in: International Conference on Agile Software Development, XP 2016, Springer International Publishing, 2016, pp. 131–143, http://dx.doi.org/10.1007/978-3-319-33515-5_11.

[7] A. Mann, M.S.A. Brown, N. Kersten, 2018 State of DevOps Report, Puppet + Splunk, 2018, https://puppet.com/resources/whitepaper/2018-state-of-devops-report. (Accessed July 2019).

[8] M. Skelton, M. Pais, DevOps topologies, 2013, https://web.devopstopologies.com/. (Accessed July 2019).

[9] M. Skelton, M. Pais, Team Topologies: Organizing Business and Technology Teams for Fast Flow, IT Revolution Press, 2019.

[10] M. Shahin, M. Zahedi, M.A. Babar, L. Zhu, Adopting continuous delivery and deployment: Impacts on team structures, collaboration and responsibilities, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE'17, ACM, 2017, pp. 384–393, http://dx.doi.org/10.1145/3084226.3084263.

[11] P.A. Anderson, Decision making by objection and the Cuban missile crisis, Adm. Sci. Q. 28 (2) (1983) 201–222, http://dx.doi.org/10.2307/2392618.

[12] L.T. Pinfield, A field evaluation of perspectives on organizational decision making, Adm. Sci. Q. 31 (3) (1986) 365–388, http://dx.doi.org/10.2307/2392828.

[13] D.G. Sirmon, M.A. Hitt, R.D. Ireland, B.A. Gilbert, Resource orchestration to create competitive advantage: Breadth, depth, and life cycle effects, J. Manag. 37 (5) (2011) 1390–1412, http://dx.doi.org/10.1177/0149206310385695.

[14] B. Glaser, A. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Transaction, 1999.

[15] P. Ralph, Toward methodological guidelines for process theories and taxonomies in software engineering, IEEE Trans. Softw. Eng. 45 (7) (2019) 712–735, http://dx.doi.org/10.1109/TSE.2018.2796554.

[16] B.B.N. de França, H. Jeronimo, G.H. Travassos, Characterizing DevOps by hearing multiple voices, in: Proceedings of the 30th Brazilian Symposium on Software Engineering, SBES '16, ACM, 2016, pp. 53–62, http://dx.doi.org/10.1145/2973839.2973845.

[17] N. Forsgren, M.A. Rothenberger, J. Humble, J.B. Thatcher, D. Smith, A taxonomy of software delivery performance profiles: Investigating the effects of devops practices, in: AMCIS 2020 Proceedings, no. 8, 2020.

[18] N. Forsgren, J. Humble, G. Kim, Measuring performance, in: Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations, IT Revolution Press, 2018.

[19] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: A critical review and guidelines, in: 2016 IEEE/ACM 38th International Conference on Software Engineering, ICSE '16, 2016, pp. 120–131, http://dx.doi.org/10.1145/2884781.2884833.

[20] M. Waterman, J. Noble, G. Allan, How much up-front?: A grounded theory of agile architecture, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, ICSE '15, 2015, pp. 347–357, http://dx.doi.org/10.1109/ICSE.2015.54.

[21] R. Hoda, J. Noble, Becoming agile: A grounded theory of agile transitions in practice, in: 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE '17, 2017, pp. 141–151, http://dx.doi.org/10.1109/ICSE.2017.21.

[22] R. Santos, F. Silva, C. Magalhaes, C. Monteiro, Building a theory of job rotation in software engineering from an instrumental case study, in: 2016 IEEE/ACM 38th International Conference on Software Engineering, ICSE '16, 2016, pp. 971–981, http://dx.doi.org/10.1145/2884781.2884837.

[23] W.P. Luz, G. Pinto, R. Bonifácio, Adopting devops in the real world: A theory, a model, and a case study, J. Syst. Softw. 157 (2019) 110384, http://dx.doi.org/10.1016/j.jss.2019.07.083.

[24] R. Siqueira, D. Camarinha, M. Wen, P. Meirelles, F. Kon, Continuous delivery: Building trust in a large-scale, complex government organization, IEEE Softw. 35 (2) (2018) 38–43, http://dx.doi.org/10.1109/MS.2018.111095426.

[25] D. Cukier, F. Kon, A maturity model for software startup ecosystems, J. Innov. Entrep. 7 (2018) http://dx.doi.org/10.1186/s13731-018-0091-6.

[26] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles, A survey of DevOps concepts and challenges, ACM Comput. Surv. 52 (6) (2019) 127:1–127:35, http://dx.doi.org/10.1145/3359981.

[27] L. Leite, F. Kon, G. Pinto, P. Meirelles, Building a theory of software teams organization in a continuous delivery context, in: 42nd International Conference on Software Engineering Companion, ICSE '20 Companion, 2020, pp. 294–295, http://dx.doi.org/10.1145/3377812.3390807.

[28] L. Leite, F. Kon, G. Pinto, P. Meirelles, Platform teams: An organizational structure for continuous delivery, in: IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, 2020, pp. 505–511, http://dx.doi.org/10.1145/3387940.3391455.

[29] P.E. Strandberg, Ethical interviews in software engineering, in: International Symposium on Empirical Software Engineering and Measurement 2019, ESEM '19, 2019, http://dx.doi.org/10.1109/ESEM.2019.8870192.

[30] W.C. Adams, Conducting semi-structured interviews, in: Handbook of Practical Program Evaluation, third ed., Jossey-Bass, 2010.

[31] S. Georgieva, G. Allan, Best practices in project management through a grounded theory lens, Electron. J. Bus. Res. Methods 6 (1) (2008) 43–52.

[32] M. Miles, M. Huberman, Focusing and bounding the collection of data - the substantive start, in: Qualitative Data Analysis: A Methods Sourcebook, third ed., Sage Publications, 2013, Chapter 2.

[33] M. Shahin, M. Zahedi, M.A. Babar, L. Zhu, An empirical study of architecting for continuous delivery and deployment, Empir. Softw. Eng. 24 (2019) 1061–1108, http://dx.doi.org/10.1007/s10664-018-9651-4.

[34] R.K. Yin, Case Study Research, Design and Methods, fourth ed., SAGE Publications, 2009.

[35] E. Guba, Criteria for assessing the trustworthiness of naturalistic inquiries, Educ. Technol. Res. Dev. 29 (1981) 75–91, http://dx.doi.org/10.1007/BF02766777.

[36] P. Debois, Just enough documented information, in: Agile 2008 Toronto, 2008.

[37] E.M. Goldratt, J. Cox, The Goal: A Process of Ongoing Improvement, North River Press, 2014, 30th anniversary edition.

[38] J. Humble, There's no such thing as a "devops team", 2012, https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team. (Accessed August 2018).

[39] J. Davis, R. Daniels, Effective DevOps: building a Culture of Collaboration, Affinity, and Tooling At Scale, O'Reilly Media, 2016.

[40] H. Kniberg, Spotify engineering culture (part 1), 2014, https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1. (Accessed August 2019).

[41] M. Shahin, M.A. Babar, On the role of software architecture in DevOps Transformation: An industrial case study, in: Proceedings of the International Conference on Software and System Processes, in: ICSSP '20, ACM, 2020, pp. 175–184, http://dx.doi.org/10.1145/3379177.3388891.

[42] J. Gray, A conversation with Werner Vogels, ACM Queue 4 (4) (2006) 14–22, http://dx.doi.org/10.1145/1142055.1142065.

[43] K. Morris, Infrastructure as Code: Managing Servers in the Cloud, O'Reilly Media, 2016.

[44] K. Eisenhardt, Building theories from case study research, Acad. Manag. Rev. 14 (4) (1989) 532–550, http://dx.doi.org/10.5465/amr.1989.4308385.

[45] M. Leppanen, S. Makinen, M. Pagels, V. Eloranta, J. Itkonen, M.V. Mantyla, T. Mannisto, The highways and country roads to continuous deployment, IEEE Softw. 32 (2) (2015) 64–72, http://dx.doi.org/10.1109/MS.2015.50.

[46] H.H. Olsson, H. Alahyari, J. Bosch, Climbing the "stairway to heaven" – A mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software, in: 38th Euromicro Conference on Software Engineering and Advanced Applications, 2012, pp. 392–399, http://dx.doi.org/10.1109/SEAA.2012.54.

[47] S. Neely, S. Stolt, Continuous delivery? easy! just change everything (well, maybe it is not that easy), in: 2013 Agile Conference, 2013, pp. 121–128, http://dx.doi.org/10.1109/AGILE.2013.17.

[48] A. Mockus, Organizational volatility and its effects on software defects, in: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE '10, ACM, 2010, pp. 117–126, http://dx.doi.org/10.1145/1882291.1882311.

[49] M. Hilton, A. Begel, A study of the organizational dynamics of software teams, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18, ACM, 2018, pp. 191–200, http://dx.doi.org/10.1145/3183519.3183527.

[50] B. Beyer, C. Jones, J. Petoff, N.R. Murphy, Site Reliability Engineering: How Google Runs Production Systems, O'Reilly Media, 2016.

[51] D. Dodou, J. de Winter, Social desirability is the same in offline, online, and paper surveys: A meta-analysis, Comput. Hum. Behav. 36 (2014) 487–495, http://dx.doi.org/10.1016/j.chb.2014.04.005.

[52] N. Forsgren, A. Brown, J. Humble, N. Kersten, G. Kim, 2017 State of DevOps Report, Puppet + Dora (DevOps Research & Assessment), 2017, https://puppet.com/resources/whitepaper/2017-state-of-devops-report. (Accessed December 2020).