A theory of organizational structures for development and infrastructure professionals

Leonardo Leite, Nelson Lago, Claudia Melo, Fabio Kon, Paulo Meirelles

Abstract—DevOps and continuous delivery have impacted the organizational structures of development and infrastructure groups in software-producing organizations. Our research aims at revealing the different options adopted by the software industry to organize such groups, understanding why different organizations adopt distinct structures, and discovering how organizations handle the drawbacks of each structure. We interviewed 68 carefully-selected IT professionals, 45 working in Brazil, 10 in the USA, 8 in Europe, 1 in Canada, and 4 in globally distributed teams. By analyzing these conversations through a Grounded Theory process, we identified conditions, causes, reasons to avoid, consequences, and contingencies related to each discovered structure (segregated departments, collaborative departments, API-mediated departments, and single department). In this way, we offer a theory to explain organizational structures for development and infrastructure professionals. This theory can support practitioners and researchers in comprehending and discussing the DevOps phenomenon and its related issues, and also provides valuable input to practitioners' decision-making.

Index Terms—DevOps, Software Teams, Organizational Structures, Continuous Delivery, Software Engineering, Grounded Theory

1 Introduction

Over the last decade, seeking to accelerate time-to-market and improve customer satisfaction, software-producing organizations have adopted DevOps and continuous delivery. The automation and collaboration brought by such initiatives have impacted the way these companies have arranged development and infrastructure groups regarding operational activities and the setup of their underlying infrastructure[1] [3]. For example, with an automated deployment pipeline, one can question the role of an engineer exclusively assigned to command deployments. Therefore, due to such recent changes, there is a need to investigate how these companies are structuring development and infrastructure groups regarding operations activities, such as provisioning, deployment, and monitoring. Such investigation is relevant for practitioners, especially considering that commonly adopted structures may not necessarily yield the best results [4].

Empirical software engineering studies have focused on identifying the different organizational structures adopted in the industry to arrange development and infrastructure professionals [5], [6], [7], [8]. However, the current literature does not reveal why these diverse organizational structures exist. This is a missing key for enabling practitioners to make good use of such results. Decision-makers in software companies are highly interested in knowing "what other companies are doing" and "why are they doing it" to support their decisions. Moreover, scholars should strive to deeply comprehend the software production phenomenon so they can adequately teach software engineering considering the reality of the industry abstracted under adequate theories. Therefore, in this paper, we focus on the following research questions:

> **RQ1:** *Why do different software organizations adopt different organizational structures regarding development and infrastructure groups?*
> **RQ2:** *How do organizations handle the drawbacks of each organizational structure?*

We investigate these questions in the context of software-producing organizations responsible for deploying the software they produce. For brevity, we refer to them in this paper simply as "organizations" or "companies."

To answer the research questions above, we conducted interviews with 68 IT professionals in 54 organizations. We analyzed these interviews by following a Grounded Theory process [9] to build a *theory to explain* [10] the organizational structures of development and infrastructure professionals in the context of contemporary software production. Our findings present concerns that affect structure selection (e.g., handling delivery bottlenecks, enforcing corporate standards for infrastructure, and having some parity between infra and development personnel) and consequences of structure adoption (e.g., conflicts due to blurred responsibilities and developers with reckless behavior when using internal infrastructure platforms that are too abstract). We also reveal strategies used by companies to handle the drawbacks of different structures (e.g., tuning the abstraction level of internal infrastructure platforms).

We proceed by presenting related works in Section 2 and

- *L. Leite, N. Lago, C. Melo, F. Kon, and P. Meirelles are with the Department of Computer Science at the University of São Paulo (IME-USP), São Paulo, Brazil.*
E-mail: {leofl, lago, claudia, kon, paulormm}@ime.usp.br,
Published as: *L. Leite, N. Lago, C. Melo, F. Kon, and P. Meirelles, "A theory of organizational structures for development and infrastructure professionals," in IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 1898-1911, 1 April 2023, doi: 10.1109/TSE.2022.3199169.*

1. According to the ITIL glossary [1], infrastructure refers to all of the hardware, software, networks, facilities, etc. that are required to build and operate IT services. In our context, it is helpful to add to this definition the notion that infrastructure is a layer of the software stack that can be provided as a commodity [2]. Although infrastructure must meet application non-functional requirements, it usually is provided regardless of the application domain, enabling organizations to offer it for different applications in a standardized manner.

the research design in Section 3. Then, we present our results in Sections 4 and 5, and discuss them in Section 6. We debate our quality criteria and threats to validity in Sections 7 and 8. Finally, we draw our conclusions in Section 9.

2 Related work

Based on several studies [11], [12], [13], Oliveira and Takahashi summarize the basic elements of organizational structures as differentiation (division of labor) and integration (coordination) [14]. Such definition aligns well with how Conway's Law compares graphs abstracting system structures (subsystems' interconnections) and organization structures (communication paths among groups of people) [15]. Therefore, "organizational structure" (or just "structure" in this paper) can be understood as "differentiation and integration patterns".

Organizations and their structures have been extensively studied not only in the management and administration fields [16], [17], [18], [19], but also in the software engineering context. Seaman and Basili consider an organizational structure to be a network of relationships of different types (e.g., collaborating and reporting) among developers in a company [20]. They investigate the association between organizational attributes and developers' communication efforts in code inspection meetings. Herbsleb and Roberts study how developers can optimally coordinate decision-making by considering the interplay among multiple decisions [21]. Nagappan et al. provide eight metrics to quantify organizational complexity, seeking to associate such metrics with software quality [22]. Tamburri et al. report a list of organizational social structures for the software engineering context, including communities, networks, groups, and teams [23]. Their structures could be helpful especially for classifying associations among developers of different teams, such as: communities of practice, informal communities, learning communities, networks of practice, informal networks, social networks, and workgroups. However, these earlier works focus on development activities, without considering infrastructure or operations concerns.

More recent works have proposed taxonomies [24] for the interactions between development and infrastructure professionals [5], [6], [7], [8], [25], [26], [27]. However, most of them do not focus on the conception of their taxonomies and instead take them as a starting point for their work. Nonetheless, Shahin et al. [6] systematically conducted interviews and surveys, and found four types of team structures: *i) separate Dev and Ops teams with higher collaboration*, *ii) separate Dev and Ops teams with a facilitator in the middle*; *iii) small Ops team and more responsibilities for the Dev team*, and *iv) no visible Ops team*.

Independently from Shahin et al., in our previous work [28], we created a taxonomy of structures in use by the industry to organize development and infrastructure professionals, which mostly coincides with that of Shahin et al. In particular, a significant difference between our taxonomy and theirs is that we identified the *platform teams*, a pattern recently advocated by practitioners [26], [29]. Thus, in our preliminary results [3], [28], [30], [31], we identified the following structures (which we renamed in this paper, as described in Sections 3.3 and 4):

- **Segregated departments** (originally "siloed departments"), with highly bureaucratized cooperation among development and operations.
- **Collaborating departments** (originally "classical DevOps"), focusing on facilitated communication and collaboration among development and operations.
- **API-mediated departments** (originally "platform teams"), in which the infrastructure team provides highly-automated infrastructure services to assist developers.
- **Single departments** (originally "cross-functional teams"), in which teams take responsibility for both software development and infrastructure management.

We presented such structures as a grounded theory [9] in the taxonomy form (i.e., as a classification system) [24]. We observed that professionals may be organized differently for different deployable units and that an organization can be in a transitional state from one structure to another. We also found evidence that **API-mediated departments** promote better delivery performance [32]. Table 1 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).

More on the differences between our taxonomy and those of others are published elsewhere [28]. However, Lopez-Fernandez et al. developed a taxonomy concurrently with ours [7]. Although they organize their taxonomy differently from ours, both results present essential common elements, encompassing the ideas of collaborating departments, cross-functional teams, and the provisioning of platforms. They also corroborate our finding that platform teams foster delivery performance. Lopez-Fernandez et al. [7] acknowledge the common points among the works of Shahin et al. [6], ours [28], and theirs, highlighting different insights brought by them and stressing how these models can be appreciated in conjunction, providing data and methods triangulation.

However, the above related works present taxonomies to *describe* structures. They do not *explain* why different companies adopt different organizational structures, which is our primary research goal in this paper. Nonetheless, Erich et al. [33] authored an earlier article closer to this goal, in which they seek to provide an explanation for DevOps. For six interviewed organizations, they explored: why and how to adopt DevOps, and the consequences (problems and results) of adopting it. They also explored organizational structures (e.g., discussing DevOps teams and the distribution of responsibilities). Still, their analysis is targeted at specific organizations, with no theory building. In other words, they do not provide a taxonomy of DevOps structures, but simply describe the DevOps structures of six organizations. Moreover, causes and consequences are related to the "DevOps adoption" category, not to different structures, which we discuss here.

3 Methodology and research design

We applied classic Grounded Theory (GT) [9], [34] to build a theory based on data retrieved from semi-structured interviews conducted in real-world organizations. GT is a

TABLE 1
Operations responsibilities and interactions in each organizational structure

Organizational structure	*Development differentiation*	*Infrastructure differentiation*	*Integration*
Segregated departments	Just builds the application package	Responsible for all operations activities	Limited collaboration among the groups
Collaborating departments	Participates/collaborates in some operations activities	Responsible for all operations activities	Intense collaboration among the groups
Single departments	Responsible for all operations activities	Does not exist	—
API-mediated departments	Responsible for all operations activities with the platform support	Provides the platform, automating much of the operations activities	Interaction happens in specific situations, not on a daily basis

TABLE 2
Description of participants and organizations

Revisit	*Organizational structure*	*Number of employees in the organization*	*Reference codes and roles of interviewee*	*Interviewee location*
No	Single depart.	> 1000	I38) Developer	USA
No	Segregated to collaborating depart.	> 1000	I39) Developer	Brazil
Yes	Segregated to API-mediated depart.	> 1000	I40) Developer I41) Infrastructure manager	Brazil
Yes	Collaborating depart.	$[200, 1000]$	I42) Infrastructure manager I43) Infrastructure engineer	USA
No	API-mediated depart.	$[200, 1000]$	I44) Development manager	Brazil
No	Single depart.	< 200	I45) Developer	Brazil
Yes	API-mediated depart.	> 1000	I46) Development manager	Brazil
No	Single depart.	< 200	I47) Development manager	Brazil
Yes	API-mediated depart.	$[200, 1000]$	I48) Infrastructure manager I49) Development manager	Spain
No	Collaborating depart.	$[200, 1000]$	I50) Developer	Brazil
Yes	Segregated to collaborating depart.	$[200, 1000]$	I51) Infrastructure engineer	Brazil
No	Collaborating depart.	> 1000	I52) Infrastructure manager I54) Development manager	Brazil
No	API-mediated depart.	$[200, 1000]$	I53) Infrastructure manager	Brazil
Yes	Segregated to API-mediated depart.	> 1000	I55) Infrastructure manager	Brazil
No	API-mediated depart.	> 1000	I56) Consultant	Brazil
No	Single depart.	> 1000	I57) Infrastructure engineer	Brazil
Yes	Single depart.	$[200, 1000]$	I58) Infrastructure manager	Brazil
No	Collaborating to API-mediated depart.	< 200	I59) Infrastructure manager	Brazil
No	Collaborating depart.	< 200	I60) Developer	Brazil
No	API-mediated depart.	$[200, 1000]$	I61) Infrastructure manager I62) Infrastructure manager	Brazil
No	Single to API-mediated depart.	< 200	I63) Development manager	USA
No	Single depart.	< 200	I64) CTO	USA
No	Collaborating to single depart.	$[200, 1000]$	I65) Developer I67) Infrastructure engineer	Brazil
No	Segregated to API-mediated depart.	> 1000	I66) Architecture manager	Brazil
Yes	Collaborating depart.	> 1000	I68) Development manager	Brazil

methodology originated in social sciences, but its use in the software engineering field is already commonplace [7], [8], [34], [35]. GT supports researchers to build theories based typically on qualitative data. We can also categorize our study as a field study [36], a category of knowledge-seeking studies, in opposition to solution-seeking studies [36]. This means that, rather than a straightforward guide to action, our theory aims to guide reasoning.

We segmented our research into two phases: the first focuses on developing an initial taxonomy (already published [28]), and the second focuses on answering RQ1 and RQ2 (the present paper).

In the first phase, after initial brainstorming sessions with seven highly-skilled DevOps specialists, we interviewed 37 IT professionals of different roles (including 24 with development-related roles, 5 with infrastructure-related roles, and 11 managers) and experience (15 with more than 10 years of experience, while 9 with less than 5 years; 13 masters and 2 PhDs). These 37 interviewees were working in companies of different domains, countries (21 in Brazil, 5 in the USA, 6 in the Western Europe, 1 in Canada, and other 4 in globally distributed teams), and sizes (14 with more than 1,000 employees and 11 with less than 200).

We analyzed these 37 interviews to discover the different organizational structures used by the industry to organize development and infrastructure professionals. We analyzed such semi-structured interviews with an open coding process until we reached theoretical saturation [9]. We also got feedback from participants on our emergent structures through online surveys.

In the remainder of this section, we present the design of our second research phase. We also present, in Figure 1,

a diagram summarizing the steps and products of our research.

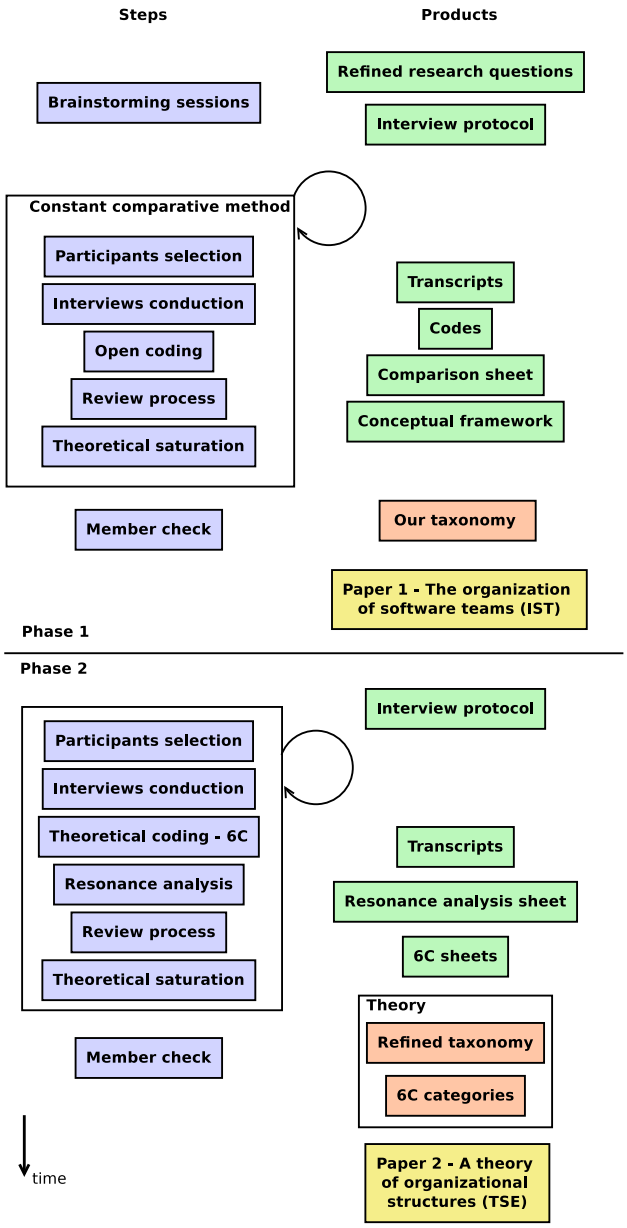

Fig. 1. Steps and products of our research

3.1 Sample

To understand "why different organizations adopt different structures" (RQ1), it is crucial to analyze organizations adopting different structures. Therefore, we selected companies already visited by us in the first research phase [28] to make sure that we would get a good coverage of all the different structures. Since company size is expected to be a relevant factor regarding our research question [6], [28], we also aimed to select organizations of different sizes. To accomplish that, we:

- Considered transitional situations as distinctive structures (e.g., **collaborating departments** is one structure, while transitioning from **collaborating** to **API-mediated departments** is another one).
- Grouped the visited organizations by structure and size (small, medium, large).
- Discarded the groups (structure and size) with only one member (less relevant groups).
- Aimed to visit one company from each remaining group.

Such a selection strategy left us with the goal to revisit 11 organizations within the following groups: large **collaborating departments**, medium **collaborating departments**, small **collaborating departments**, medium **single departments**, small **single departments**, large **API-mediated departments**, medium **API-mediated departments**, large **segregated departments**, small **segregated departments**, medium **segregated departments** transitioning to **collaborating departments**, and large **segregated departments** transitioning to **API-mediated departments**.

Considering our need for increasing generalizability, we also visited **new** organizations. Since it was not possible to know in advance what a company's structure would be, we selected them taking into account only their sizes. We did not preestablish a target on the number of **new** organizations to be visited – interviews continued until we reached saturation (see Section 3.6). As a form of triangulation, we also tried to interview a developer and an infrastructure professional from each organization, whenever it was applicable and possible.

Nonetheless, we note that having access to IT professionals willing to expose organizations' internal affairs can not be guaranteed in advance. Therefore, although the above strategy guided our selection, we could not strictly stick to it. In this way, Table 2 presents the 31 carried interviews in this phase, which extend the previous 37 interviews to the total of 68 interviews we analyzed in our research. For each interview/interviewee, we also assigned an identifier (e.g., I38) to reference in this text. We conducted these interviews from May 2020 to October 2021, and they took from 20 to 63 minutes (median of 35 minutes).

The interviewees worked in the following business domains: education, health, logistic, telecommunications, public administration, development support, hiring, marketplace, travel, manufacturing, finances, mobility, games, defense, networking, semiconductors, IoT, and cloud computing. One of the companies was a Tech Giant (among the top 4 IT companies in the world), while three were unicorns (startups with a valuation over U$1 billion).

Although we did not plan to balance the structures among **new** companies, in the end, our sample was reasonably balanced. From 17 **new** companies, four presented the **collaborating departments** structure, six showed the **single department** structure, and seven had an **API-mediated** structure. For this counting, we considered a company transitioning from structure X to Y as in Y. In particular, all the interviewed companies with **segregated departments** were already transitioning to some other structure.

3.2 Interview procedure

In the conducted semi-structured interviews, after presenting our taxonomy, we approached: which was the organizational structure in the context of the interviewee according to their opinion; why the organization adopted such a

structure over the others; whether another structure would be more suitable for the interviewee context; what were the perceived (or expected) disadvantages of the discussed structures; and what were the strategies to handle such disadvantages. During interviews, we followed Adams's guidelines [37]. We provide our complete script with the rationale of each question as supplementary material (File 1).

3.3 Resonance analysis

Initially, 37 semi-structured interviews provided us with enough data to build the first version of our taxonomy of organizational structures. In such conversations, we employed second-level questions [38] to avoid exposing the emerging structures to interviewees. After this, we performed a brief and limited member check [39] through online surveys. The following and necessary step was to observe the use of our taxonomy in practice, verifying whether it achieves the goal of a classification (support reasoning by increasing users' cognitive efficiency [24]) and serves as a grounded theory (being applicable by practitioners [9]). To answer RQ1 and RQ2, we set up a favorable context to apply and refine our taxonomy – we had to use its concepts during interviews on the second research phase (Section 3.2).

We refer to the taxonomy refinement process as *resonance analysis*, where "resonance" alludes to the degree to which findings are understandable to participants [24]. By assessing resonance in the Grounded Theory context, we do not aim to determine whether the taxonomy is "valid" or not. Under GT principles, when faced with new conflicting evidence (e.g., a taxonomy misuse), we adapted and evolved our theory, strengthening it. We note that social theories are rarely confirmed but are instead corroborated, confronted, or evolved by new studies [40], [41], [42]. Steinmacher, for example, gathered qualitative data to refine his grounded theory with additions, deletions, and reorganizations in his model [43]. Similarly, our process guided us on managing such operations in our taxonomy: adding, deleting, reorganizing, and renaming its high-level elements.

For each interview, we coded relevant excerpts as:

- providing **support** to our theory, when the interviewee employs the terms of the taxonomy to discuss reality or possibilities in a precise and confident way; or
- displaying **confusion** about our theory, when the interviewee employs a term from the taxonomy in a different way than we would expect, or there is difficulty in selecting a term to describe its reality or possibilities.

For each excerpt coded as **confusion**, we defined an action to handle the situation or justify our choice of not taking any action. In consequence, the taxonomy evolved with new versions. We did not wait for interviews to be over to apply such actions; we interleaved collection, analysis, and actions, following GT principles. In this way, we conducted the last interviews based on the latest taxonomy version.

The procedures of our resonance analysis combine aspects of directed content analysis (coding by using predefined categories) and summative content analysis (counting codes and assessing the contexts for **confusions**) [44]. We present the results of the resonance analysis in Section 4.

3.4 6C analysis

Glaser proposed 18 theoretical coding families to guide researchers in labeling data at the conceptual level [45]. These families were intended to sensitize these professionals to the many ways through which a concept could be examined. The "bread and butter" coding family for sociologists are what he labels "The Six C's: causes, contexts, contingencies, consequences, covariances, and conditions" [46]. Other GT studies on software engineering also use this practice [35], [47], [48].

The same author states that theoretical codes are vital because they potentiate a theory's explanatory power and increase its completeness and relevance, resulting in a grounded theory with a greater scope and parsimony [49]. Hernandez explains that "without theoretical codes, the substantive codes become mere themes to describe (rather than explain) a substantive area; the descriptive thematic approach is characteristic of qualitative research methods such as phenomenology or ethnography but not Classic GT" [50]. Indeed, in this paper, we present not just a "theory for analyzing," but a "theory for explaining" [10].

As it happened with Hoda et al. [47], when comparing theoretical coding families, it became evident that the 6C coding family is the most suited for our research goals. We understand that our research questions can be answered in the 6C format since, for a given *context* and certain *conditions*, there are *causes* that lead organizations to take actions expecting specific *consequences* – although *contingencies* emerge as well. Moreover, there may be a *covariance* of such causes, consequences, and contingencies in specific contexts and conditions.

Thus, we transcribed each interview, highlighted key points from the interviews (the excerpts with potential theoretical interest), extracted codes from the key points, and associated them with 6C labels. Usually, theoretical coding associates codes with the core category of the study. In our case, the four organizational structures of our taxonomy are our core categories. As the coding process advanced, we merged and abstracted different codes, as it is common to open coding too. Hence, a code may have different sources (interviews). We provide the key points and the extracted codes as supplementary material (Files 2 and 3).

Although the original 6C labels provide a robust analysis framework, there is no reason to force relevant codes to fit into them. Therefore, as we perceived a need during analysis, and as agreed in review sessions, we adapted the labels – such flexibility agrees with Glaser's ideas [50]. Thus, the employed definitions of the labels used during our 6C analysis were:

- **Characterization**: identifying structures' characteristics, i.e., aspects that enable relating companies to structures.
- **Conditions**: environmental conditions that are necessary to implement a structure (i.e., prerequisites).
- **Causes**: reasons/motivations/opportunities that led the organization to adopt a particular structure and not another.
- **Avoidance reasons**: reasons/motivations that led the organization not to adopt a particular structure.
- **Consequences**: outcomes that happen or are expected to happen after an organization adopts a structure,

including unexpected issues.
- **Contingencies**: strategies to overcome a structure's drawbacks.

We did not analyze context and covariance interview-by-interview. The context of our interviews is the one presented in Section 3.1. We performed the covariance analysis after the last interview. We used **characterization** codes to link interviews and their codes to structures. Nevertheless, as they largely overlap with our taxonomy's **core categories** [28], we do not explicitly report them here.

It is troublesome to differentiate facts from interviewees' opinions. We mitigated this issue by reporting only codes supported by at least three interviews. The rationale is that an idea supported by three persons, thus in at least two companies, is worthy of consideration to some degree. We call the **conditions**, **causes**, **avoidance reasons**, **consequences**, and **contingencies** supported by at least three interviews as **strong codes**. We present the results of the 6C analysis in Section 5.

3.5 Review process

The first author of this paper conducted the initial analysis of each interview. Periodically, other two authors joined for review sessions, in which we held discussions until reaching consensus. While the first author is also a software developer, it is relevant to note that one of the reviewer authors is also an experienced infrastructure professional. Receiving insight from this type of professional for a DevOps-related study was a recommendation given by an interviewee on the first research phase, aiming to soften possible biases given the first author's metier. We held 14 review sessions, from January to October 2021, with most of the sessions taking around two hours.

3.6 Theoretical saturation

At each analysis snapshot (an interview analysis or a review session), we tracked a few metrics to identify theoretical saturation [9]. For the 6C analysis, we defined the following metrics: number of codes, number of **strong codes**, and **conceptual density**.

The number of codes indicates the total "amount of learning" we had in each interview. Although we can always learn something new from new people, we expect diminishing returns in this metric as we approach theoretical saturation. **Strong codes** are supported by at least three interviews, excluding characterization codes. We also expected this metric to initially grow with new interviews and then stabilize when reaching saturation.

Each code has a **support number**, that is, the number of sources supporting that code. The **conceptual density** is the sum of **support numbers** divided by the total number of codes. The idea is that a high density indicates that codes are supported by multiple interviews, pointing to robust results. The expectation was that the value of this metric should only grow.

As observable in Figure 2, along with the interviews, the metrics followed the expected trends, indicating enough theoretical saturation. In detail, we note that the decreases in the number of codes (L1) are related to the review sessions,

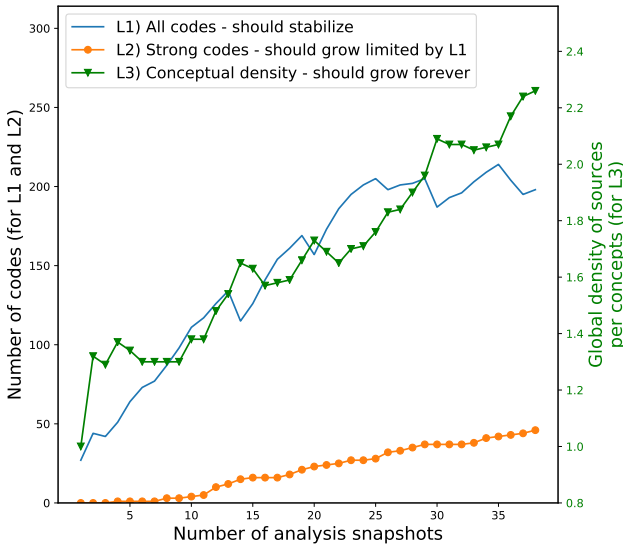

Fig. 2. Metrics of theoretical saturation for the 6C analysis

in which we deleted codes we judged to be inadequate, besides merging other ones, forming more abstract codes.

For the resonance analysis, we defined two metrics: **support level** and **taxonomy changes**. The **support level** is the difference between the number of **support** codes and the number of **confusion** codes. A high **support level** indicates a good resonance of the taxonomy with participants. This metric should ideally only grow.

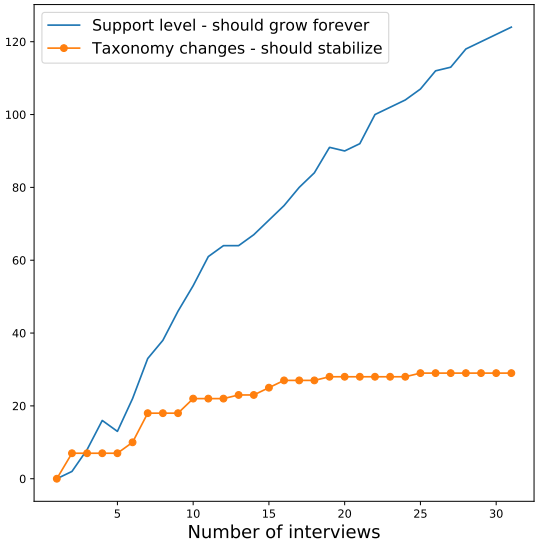

Fig. 3. Metrics of theoretical saturation for the resonance analysis

Taxonomy changes is the number of changes in the high-level view of our taxonomy: addition, renaming, and removal of elements. We derived these changes from observed **confusions** in the interviews or even neutral comments pointing to potential improvements. We expected an initial increase in the number of changes (after all, updating the taxonomy based on the practitioners' views was in

our process), followed by stabilization, indicating that the previous modifications subsequently caused fewer **confusions** among the interviewees. As observable in Figure 3, the metrics again followed the expected trends, indicating enough theoretical saturation.

We clarify that the metrics used here were intuitively conceived by us, given the lack of metrics to detect theoretical saturation in Grounded Theory studies [47], [48], [51].

4 Identifying Organizational Structures

Based on the first 37 semi-structured interviews and following GT guidelines, we elaborated an initial version of our taxonomy, with each structure having **core** and **supplementary properties**. **Core properties** are expected to be found in corporations with a given structure. **Supplementary properties** refine the explanation of a structure, but their association with organizations is not compulsory. Such properties are presented in detail elsewhere [28].

After having this initial version and 31 more semi-structured interviews, we conducted our resonance analysis process, in which we coded fragments of these 31 conversations as **support** or **confusion** fragments. For each excerpt coded as **confusion**, we defined an action to handle the situation or justified our choice of not taking any action. In total, we recorded 35 actions, 26 of them classified as *change the taxonomy*, five as *improve interview presentation*, and four as *improve report*. Actions changed the high-level elements of our taxonomy (organizational structures and **supplementary properties**) with additions, removals, and renamings. We considered interview presentation improvements in how we explained the taxonomy to the subsequent interviewees. We incorporated report improvements in descriptions presented in our online digest of organizational structures[2]. We provide the list of **supports** and **confusions**, plus a summary of actions as supplementary material (Files 4 and 5).

One example of a fragment (I44) coded as **confusion** that led to a taxonomy change: "*I think it's an in-house open-source platform, because although we built it on top of cloud services, AWS in particular, our ecosystem is mostly open-source, based on Kubernetes and its ecosystem.*" Before this comment, we considered that open-source platforms were installed and ran in physical infrastructure only. Then we noted the following memo [9] during analysis: "*Good point. . . maybe we have to expand the scope of the in-house platform (...) when this use of the cloud is merely the use of virtual machines; the company is still building/installing/managing something on its own.*"

Therefore, to address this **confusion**, we took the action of renaming the **supplementary property** "in-house open-source platform" to "in-house-administered open-source platform." Adding the word "administered" suggests what matters is the company administering the open-source platform regardless of whether it is installed in a physical server or in a virtual machine provided by a cloud vendor.

Another notable change in the taxonomy was renaming "Platform team" to "API-mediated departments." We had **confusions** in four interviews (I40, I42, I50, I52) due to the polymorphic meanings of the terms "platform" and "platform team." Moreover, the current name reflects the structure better, since the platform team is just one of its interplaying teams.

We also explicitly adopted the term "dev & infra departments" to rename the structures. This aligns better with the fact that the structures reflect the division of operational activities between development and infrastructure groups, as portrayed by Table 1. We avoid the term operators or even operations staff, which would refer to professionals who accept and operate new and changed software in production and are responsible for its service levels [52]. In fact, our research seeks to understand how these activities were redistributed between development and infrastructure professionals with the DevOps and continuous delivery advent.

Figure 4 presents the consolidated version of our taxonomy, as evolved from the refinements. The figure shows the taxonomy's high-level elements: its structures and their associated **supplementary properties**. We provide all the versions of the high-level view of our taxonomy as supplementary material (File 6).

5 Explaining Organizational Structures

Now we present, in Tables 3, 4, 5, and 6, the 46 discovered **strong codes** grouped by organizational structure. Each **strong code** is preceded by its identifier (e.g., SC02) and the amount of supporting interviews.

TABLE 3
Strong codes for segregated dev & infra departments

	Consequences
SC01 (5)	Devs lack autonomy and depend on ops
SC02 (4)	Low delivery performance (queues and delays)
SC03 (3)	Friction and blaming games between devs and infra

TABLE 4
Strong codes for collaborating dev & infra departments

	Conditions
SC04 (5)	Enough infra people to align with dev teams
SC05 (3)	Top management support
	Causes
SC06 (4)	In a non-large company / with few products, it is easier to be collaborative
SC07 (3)	Trying to avoid the delivery bottleneck
SC08 (3)	Bottom-up initiative with later top-management support
	Consequences
SC09 (6)	Growing interaction inter-areas (e.g., knowledge sharing)
SC10 (5)	Precarious collaboration (ops overloaded)
SC11 (4)	Discomfort/frustration/friction/inefficiency with blurred responsibilities (people don't know what to do or what to expect from others)
SC12 (3)	Waiting (hand-offs), infra still a bottleneck
SC13 (3)	Automation supports collaboration
	Contingencies
SC14 (3)	Giving more autonomy to devs (in staging or even production)

During the coding process, we linked a few codes to **supplementary properties** of our taxonomy. The **strong codes**

2. http://ime.usp.br/~leofl/devops/2021-06-20/structures-digest.html

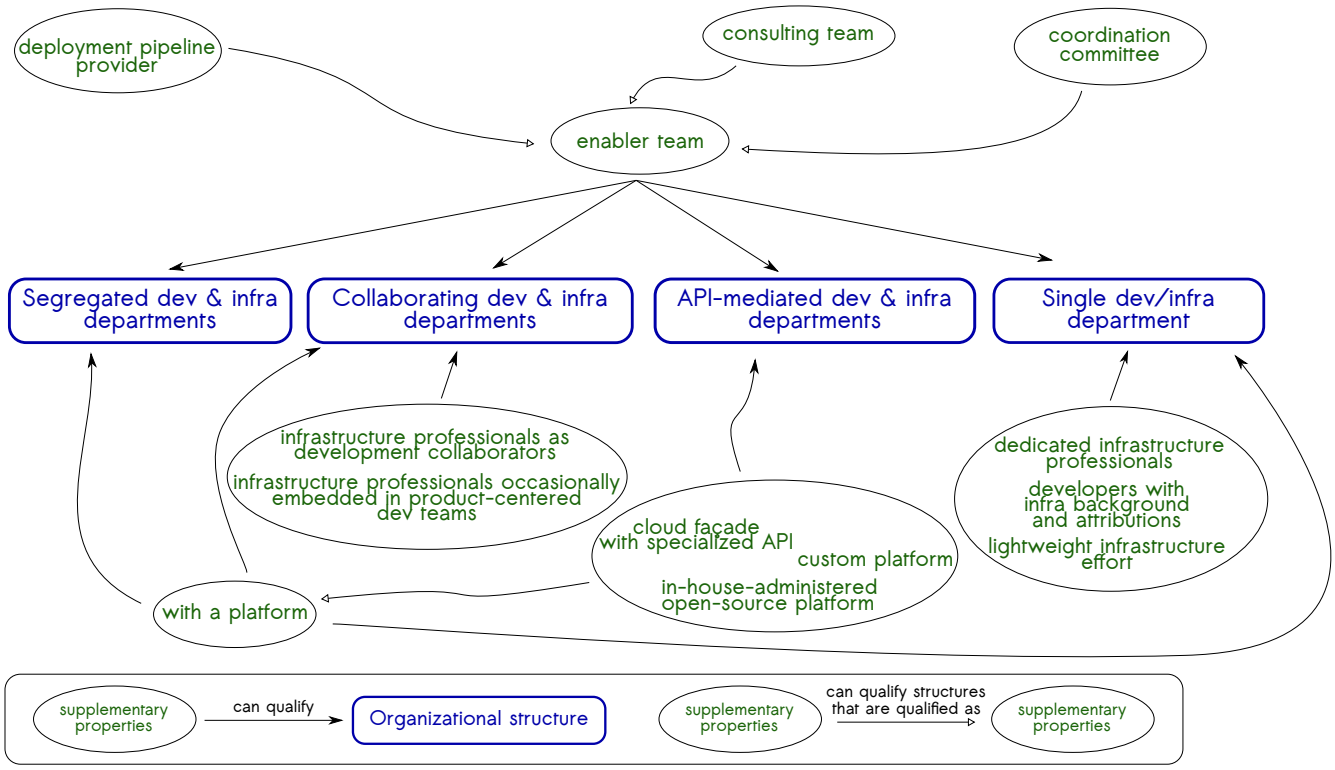

Fig. 4. The high-level view of our taxonomy after the refinement process

TABLE 5
Strong codes for single dev & infra departments

	Conditions
SC15 (3)	Enough ops for each dev team
	Causes
SC16 (10)	Startup scenario (small, young, weak infra scalability requirements, business focus, use of cloud services to limit costs)
SC17 (6)	Cloud services decrease the need of infra & ops staff
SC18 (3)	Delivery velocity, agility, critical project
	Avoidance reasons
SC19 (4)	Not suitable for applying corporate governance & standards
SC20 (3)	More costs: duplication of infra work among teams, high salaries for infra professionals, underused infra professionals
	Consequences
SC21 (9)	No [infra] defaults across teams: freedom, but possibly leading to duplication of efforts and high maintenance costs
	Contingencies
SC22 (3)	Improve infra skills in-house, inclusive with tech talks

having such attribution are SC15 (associated with "dedicated infrastructure professionals") and SC22 (associated with "developers with infra background and attributions").

We also report, in Table 7, the total number of codes (not only **strong codes**) found per **code class** (6C label vs. organizational structure). One example of a code that is not a **strong code**, an avoidance reason for **single departments** found in I55 and I58, is "specialized knowledge brings scale gains." We list all the codes in the supplementary material (File 3), where each code is linked to its supporting interviews.

To illustrate, we present here some supporting statements from the interviews (for one condition, one consequence, one avoidance reason, one cause, and one contingency):

SC04 (condition): "*We tried this approach [collaborating departments] for a while, and it didn't work. (...) The infra team was small, and it had to be in several teams at the same time.*" (I49).

SC11 (consequence): "*If there is a problem, it may not be clear who owns the problem, and this creates inefficiency. This division hurts ownership.*" (I38).

SC19 (avoidance reason): "*They are afraid of being too open and starting to lose control over the best practices and care with security, availability, attacks.*" (I68).

SC28 (cause): "*The platform team was ideal because it was how we managed to move people within the company without generating a very big impact to other areas to set up a platform.*" (I41).

SC46 (contingency): "*The biggest discussion was about to what extent devs have to understand the infra. I radically thought devs should know some minimum. But the tech leads thought differently, that devs didn't have to know anything.*" (I53).

In the supplementary material, we discuss the above-cited codes in more depth (File 8). A list with one supporting excerpt for each strong code is also available in the supplementary material (File 9).

5.1 Covariance analysis

Covariance, part of the 6C analysis, means the occurrence of one code correlates with the occurrence of another code [45], [47]. Covariance analysis reveals codes commonly supported by multiple interviews, providing more insight

TABLE 6
Strong codes for API-mediated dev & infra departments

	Conditions
SC23 (8)	Medium to large sized company
SC24 (5)	Top-down initiatives/sponsorship
SC25 (4)	Upfront investment
SC26 (3)	Requires coding skills from infra people
	Causes
SC27 (8)	Delivery bottleneck in infra management
SC28 (4)	Compatible with existing rigid structures (low impact on organogram) / Only a few people needed to form a platform team
SC29 (4)	Fosters continuous delivery
SC30 (4)	A hero or visionary (hero culture)
SC31 (4)	Emerged as best solution; other initiatives not so fruitful
SC32 (3)	Multiple products / multiple dev teams / multiple clients (requires high delivery performance)
	Consequences
SC33 (8)	Interaction (devs x platform team) to: support devs, make things work, and demand new capabilities from the platform
SC34 (7)	The platform provides common mechanisms (e.g., scaling, billing, observability, monitoring)
SC35 (4)	Promotes continuous delivery, agility, and faster changes
SC36 (4)	Devs responsible for infra architecture / concerns (e.g., non-functional requirements)
SC37 (4)	Platform team provides consulting and documentation to devs
SC38 (4)	Adding devs do not require adding [proportionally] more infra people
SC39 (3)	Eliminated previous bottleneck
SC40 (3)	Small platform team (excellence center)
SC41 (3)	High costs when using public clouds
SC42 (3)	Devs skills are too focused on corporate needs, lacking base infra knowledge (bad for devs themselves, not for the company)
SC43 (3)	The cost of managing the platform (even using open-source software) is high
SC44 (3)	Risk: platform is magic to devs; neglect quality because they trust too much in the platform, any problem they blame the platform and do not know what to do, even for simple problems or when the problem is in the application itself
SC45 (3)	Devs possibly unable to understand the infra or to contribute to the platform
	Contingencies
SC46 (3)	Decide how much devs must be exposed to the infra internals (some places more, some places less)

into theory building. We define a **strong covariance group** as a group of at least three codes related to a set of at least three sources. For example, codes A, B, and C are in the same **strong covariance group** when all of them are supported by interviews 1, 2, and 3.

Following this definition, we found: 17 **strong groups** with 3 codes linked to a 3 source-set; 1 **strong group** with 4 codes linked to a 3 source-set; 1 **strong group** with 5 codes linked to a 3 source-set; and 2 **strong groups** with 3 codes linked to a 4 source-set. Table 8 describes the last four **strong covariance groups**, which are the strongest ones since they have more codes or more sources.

6 Discussion

In this section, we first explain how the presented results relate to the posed research questions. Then, we indicate some practical implications of our findings and further interpretations of the results.

TABLE 7
Amount of codes found per code class

	Segregated	*Collaborating*	*API-mediated*	*Single*
Characteristics	5	4	6	2
Conditions	0	4	6	3
Causes	5	9	25	8
Avoidance reasons	1	1	3	7
Consequences	7	16	32	13
Contingencies	1	9	13	18

TABLE 8
Strong covariance groups

Group reference	*Strong codes*	*Supporting interviews*
G1	SC27, SC33, SC34, SC37	I57, I59, I62
G2	SC10, SC24, SC27, SC30, SC31	I40, I41, I48
G3	SC27, SC33, SC34	I48, I57, I59, I62
G4	SC10, SC24, SC27	I40, I41, I48, I49

RQ1 is about why different organizations adopt different structures for development and infrastructure professionals. **Conditions** and **causes** in the 6C analysis point in this direction. For example, a startup, still struggling to validate its value proposition, is not in a moment to be so cautious about infrastructure requirements. Therefore, such a startup scenario (SC16) usually leads to the adoption of **single departments**, with developers taking care of infrastructure concerns. After scalability and other infrastructure concerns gain relevance for the company, and the company increases its portfolio with multiple products to multiple clients (SC32), **API-mediated departments** is seen as a path to overcome existing delivery bottlenecks (SC27), and this promise seems to be fulfilled (SC39). **Collaborative departments** requires certain parity in the ratio of development to infrastructure people (SC15) to increase its success possibility. Therefore, having a low number of infrastructure professionals and a hierarchy culture (SC28), hampering direct contact between departments on a daily basis, are forces pushing to **API-mediated departments**.

RQ2 inquires about the drawbacks and mitigations for each structure. **Avoidance reasons**, **consequences**, and **contingencies** in the 6C analysis provide us answers. Although planned to increase direct cooperation (SC09), **collaborative departments** may present side effects since responsibilities become blurred (SC11). Some interviewees expressed concerns about governance and technological standardization. We found that multiple **single departments** in mid-sized or large companies can be an obstacle to such standardization (SC19). If this is a concern[3], the company may prefer to adopt **API-mediated departments**. A peculiar disadvantage of **API-mediated departments** is that programmers are less likely to be aware of the infrastructure (SC42), which may not be a disadvantage for the company, but possibly for the developers' careers. In this context, we witnessed

3. We interviewed one very large company (I38) adopting **single departments** that were not concerned with standardization: fostering an inner "free market" of solutions was an innovation strategy.

discussions about to which degree to expose infrastructure details to developers (SC46) and strategies to communicate how to use the platform, such as personalized consulting and mass communication (SC37). These discussions also emerged from situations with programmers overly relying on the platform, ignoring the bare minimum of infra management they should master (SC44).

In this way, the association of concepts of our taxonomy to 6C codes provides explanations about the structures' phenomenon. Such a new explanatory dimension enables scholars to understand structures more deeply and better equip practitioners to discuss them and make decisions.

6.1 Practical implications

We, now, provide some advice to the software industry based on the practical implications of our research:

- If, for any reason, your organization must strongly segregate development and infrastructure professionals, do not demand high delivery performance as an organization goal (SC02).
- If your organization does not have enough infrastructure people to collaborate with developers freely, do not attempt to establish a strategy of collaborating departments (SC04).
- When adopting collaborating departments, warn professionals that it is not feasible to foresee detailed responsibilities expected from each role (SC11). Instead, actions must be taken based on goals aligned among departments.
- If you want highly autonomous teams able to move fast, be ready to give up on enforcing corporate standards to make uniform the infrastructure of these teams (SC21). Also, consider some measures to make every team knowledgeable in infrastructure: hire enough infra people (SC15), embrace cloud automation (SC17), and provide time to the workforce to improve their skills (SC22).
- If you want to achieve high delivery performance by adopting an internal infrastructure platform, be prepared to pay the price in advance (SC25): prepare infra people with development skills (SC26) and enlarge developers' responsibility over operations including non-functional concerns (SC36).
- Define your platform's expected level of abstraction (SC46) and warn people about this expectation.. This measure can soothe conflicts and leverage collaboration among developers and the platform team. Also, warn developers and the platform team about their expected interaction patterns (SC33).

We discuss more concrete implications in the supplementary material. There (File 8), we further discuss five strong codes (SC04, SC28, SC19, SC11, SC46), the same codes for which we provided supporting statements from the interviews (in Section 5). This additional discussion unveils some rich insights behind strong codes, such as bosses fearing losing their positions as an inhibitor of radical changes in structures (SC28), low participation of infrastructure professionals in agile ceremonies as an indication of failure in adopting collaborating departments (SC04), and onboarding time being a reason for the managers' concern with corporate standards for infrastructure (SC19).

6.2 Further interpretations

Unfortunately, the codes of different 6C labels were not evenly distributed across the structures of our taxonomy. In particular, we found more strong codes for **API-mediated departments**, and we had only three **contingency strong codes**. We identified many more **contingencies** (38 in total), but most of them were supported by only one interview. This may point to a lack of structure awareness by the community, so each company tries to handle the problems in different ways. It could also reflect the peculiarities of the organizations, but evaluating the raw data, that does not seem to be the case. For example, "Playground area so that devs can learn infra" (I50) appears to be a **contingency** that could be applied to many more organizations. Another interpretation points to the analysis strategy: we considered a contingency as "a solution to one problem," which splits a common solution into different codes.

The largest **class** of **strong codes** (6C label and structure) we found was **consequences** for **API-mediated departments** (32 **consequences**). This suggests that this structure may have more predictable outcomes than the others. Also, from the eight codes within the **strong covariance groups**, seven are related to **API-mediated departments**. This also puts **API-mediated departments** as a more understandable phenomenon. Interestingly, the presence of a disadvantage of **collaborative departments** (SC10) within **strong covariance groups** (G2, G4) shows a motivation for adopting **API-mediated departments**.

The structure for which we found more codes suggesting failure scenarios was **collaborative departments** (SC10, SC11, SC12). We consider it may be easier for large organizations with **segregated** structures trying to move first to **collaborative departments**. However, having a limited number of infrastructure professionals and not giving them enough budget to interact with developers are recurring factors leading to overload (SC10). Such a situation makes professionals forget the DevOps initiative and revert to the previous siloed style of work. Giving more autonomy to devs (SC14) is an attempt to handle this scenario. Moreover, we found more examples of **collaborative departments** in which the infrastructure remained as a bottleneck in the delivery path (SC12). This reinforces our previous finding that there is no correlation between **collaborative departments** and delivery performance [28].

Finally, an additional benefit of our theory, provided by its building process, is offering a taxonomy with objective terms. This concern, for example, led us to replace "silo" (a metaphor) with "segregated" in our refinement process. Such objectiveness is valuable considering the amount of energy and time practitioners take in discussing again and again what DevOps is[4] [3].

7 Quality Criteria

The quality criteria we pursued are the ones defined by Guba [39] for naturalist inquiries: credibility (how plausi-

4. See, for example, the "Chef Style DevOps Kungfu" talk: https://youtu.be/_DEToXsgrPc.

ble or true the findings are); dependability (methodology applied consistently); confirmability (opportunities for correcting research bias); and transferability (generalizability). Such criteria are widely used in Grounded Theory (GT) studies [7], [35], [53], [54]. For meeting such criteria, we applied the following treatments [53]:

- Providing a chain of evidence (the files of our supplementary material[5]), which contributes to credibility and dependability.
- Collecting interviewees' opinions on the results, i.e., member check (Section 7.1), which contributes to credibility and confirmability.
- Having a diverse selection of participants (see Section 3.1), including the triangulation with development and infrastructure staff within some organizations, which contributes to transferability;
- Triangulating among coauthors through a review process (Section 3.5) and methodology revision by a coauthor experienced in qualitative methods [55], which contribute to confirmability.
- Providing quantified evidence of saturation (Section 3.6), which contributes to dependability (this is an additional quality measure, not standard in other works).
- Reporting only codes confirmed by multiple participants (**strong codes**), which contributes to credibility and transferability.

7.1 Member check

Grounded Theory aims to formulate relevant theories for practitioners, so it is crucial to investigate whether findings make sense to them [24]. However, it is opportune to highlight that the goal of our member check is to assess only the theory resonance [24] with participants and not to validate the theory itself. Participants are not obliged to verify whether the abstractions risen from diverse data are conceptually adequate [56]. Therefore, readers should consider member check together with other quality treatments.

We performed a member check by collecting feedback on our results from the participants using online surveys, which we provide as supplementary material (File 7). For each structure, we prepared one form listing its corresponding **strong codes**. For each **strong code**, the respondent had to tick one option within the following Likert scale: "true", "usually true", "no correlation / I don't know", "usually false", "false". We also left a field for general comments. Because opining on all **strong codes** would take too long, we asked each participant to answer only one form as a strategy to increase the response rate. Nonetheless, they were free to answer all the forms if they so wished.

We sent the feedback requests in three rounds: (i) to Phase 1 interviewees, (ii) to Phase 2 interviewees, and, after some time, (iii) to the ones who did not reply to us and to the 7 participants of the initial brainstorming sessions. We received responses from 39 of these 75 participants. We received 8 responses for the **segregated departments** form, 12 for the **collaborating departments** form, 12 for the **API-mediated departments** form, and 15 for the **single departments** form. From the 564 manifested opinions on strong codes, 365 were favorable (65% of the participants considered them to be true or usually true), while only 83 (15%) were unfavorable (usually false or false). In particular, every strong code received favorable feedback from at least one participant. This result suggests a good resonance of the participants with our theory.

According to a respondent's comment, disagreements related to **single departments** may relate to possible interpretations depending on **single departments** having infrastructure specialists. Indeed, in the forms, we did not associate SC15 and SC22 to their respective **supplementary properties** (see Section 5), which possibly jeopardized respondents' reasoning. This issue was fixed in the second round of feedback requests.

7.2 Generalizability

Although we cannot claim generalizability with statistical confidence, the employed methods and the taken sample provide some relative generalizability to our theory (at least more than case studies [38] usually provide).

We took a diverse sample. Considering semi-structured interviews in the first and second research phases, we interviewed 20 companies with more than 1,000 employees, 17 with between 200 and 1,000 employees, and 17 with less than 200 employees. We interviewed people in 8 countries. The company domains also varied wildly.

In the second research phase, we applied our taxonomy in conversations with professionals working in companies that did not provide data to the classification construction in the first phase. In the context of our resonance analysis, among the interviews in **new** companies, we had 99 codes of **support** and 25 codes of **confusion** (four times more **support** than **confusion**), which contributes to showing the generalizability of our theory. In addition, **confusion** codes started to rarefy at the last interviews since we used **confusions** to improve the theory. In the 6C analysis, we found 15 **characteristics** (of 17 **characteristics** codes) that define the organizational structures in 15 new companies (88% of the **new** interviewed companies), which also contributes to show that our theory applies to these new contexts.

We also were cautious about the diversity of the interviewees' roles. Only five (13%) of the interviewees had an infrastructure role in the first research phase. In the second phase, we were able to improve this situation. From the 31 interviewees in the second phase, 14 of them (nearly half) had an infrastructure role. Since, for the second phase, we looked for people with more in-house experience, so they could answer our "why questions," we ended up also interviewing a larger fraction of managers (30% vs. 61% of managers).

The 6C analysis also provided more evidence for some findings of the first phase. In particular, **API-mediated departments** still seems to be a promising path to achieve high delivery performance in comparison with the other structures. Such results now concur with results of other independent studies [7], [29]. We heard of bottlenecks in the delivery path, especially in the **segregated** and the **collaborating departments** structures, while some interviewees claimed that the **API-mediated** structure removed the

5. Available at http://www.ime.usp.br/~leofl/devops/assets/files/smtse-v2.tar.gz.

bottleneck previously existing. Another reinforced findings is that **single departments** is more associated with small organizations, while **API-mediated departments** is not.

In summary, the points strengthening our theory's generalizability are: *(i)* diversity of professional and company profiles in our sample; *(ii)* applying our taxonomy to scenarios that did not feed its initial version; *(iii)* gathering evidence of support for the theory among **new** companies; and *(iv)* gathering evidence corroborating preliminary findings.

8 Threats to Validity

As usual to taxonomical theories [24], our work does not provide probabilistic predictions in terms of dependent and independent variables. As usual to grounded theories [9], some factors (such as interviewees anonymity and unavoidable subjectivity in analysis) make the research not fully replicable. In particular, anonymity is a trade-off with the potential number of interviewees and goes along with ethical research [57]; also, interviewing the same person again does not necessarily yield the same results.

Given cost considerations, the review process has limitations. Reviewing authors read only some transcriptions excerpts: whenever needed during reviews, we searched for these relevant excerpts to support our discussions. Our research data corresponds to people's views and opinions, which may drift from objective reality. Therefore, an observational research approach would be desirable [24]. However, our research questions are too abstract to grasp only by observation, even meeting observations, without further conversations with the observed people. Therefore, observational research with the same goal as ours would be much more expensive, if at all feasible, especially involving as many organizations as we did.

Responses were dependent on how we presented the taxonomy, which interviewees could misunderstand. A mitigation for this issue was analyzing how to improve the taxonomy presentation for the subsequent sessions.

9 Conclusion

Our research provides a theory on how software-producing companies organize their development and infrastructure workforce according to different organizational structures: **segregated departments**, **collaborative departments**, **API-mediated departments**, and **single department**. Some of these structures have relevant variations represented in our taxonomy (e.g., **single departments** may or may not encompass staff dedicated to infrastructure). We found why different organizations adopt (or do not adopt) different structures and the conditions leading to this choice (e.g., **API-mediated departments** are pursued to address delivery bottlenecks; **single departments** are avoided for its unsuitability in enforcing corporate standards; and **collaborating departments** require proportional infra people per development team). We also found that each structure has drawbacks and how organizations deal with such disadvantages (e.g., **collaborating departments** may lead to conflicts due to blurred responsibilities; tuning the platform abstraction level may prevent developers from over-relying on the platform as magical). A notable result is that we found many **contingencies**, but only a few shared among different organizations, which can be a consequence of the lack of awareness of organizational structure patterns in the software industry.

This work has implications for practice. By increasing the awareness of organizational structures in the community, practitioners can make more informed decisions on structure selection and drawback handling. Moreover, having an up-to-date view of what "other companies are doing" is always a relevant input for practitioners' decision-making. An example is that practitioners can relate observed problems in their organizations to expected consequences under our theory. Such explicit associations can avoid hours of unfruitful discussions, possibly assuming the problem to be "something that only happens here."

This work also has implications for scholars. Professors can update their understanding of the software production phenomenon based on concepts and relations grounded on the current behavior of the software industry. Thus, professors can update their software engineering classes accordingly. A secondary and methodological contribution of our work for developing new grounded theories is providing an objective technique to detect theoretical saturation, which is rarely seen in the literature.

Finally, grounded theories can always be adapted according to the discovery of new instances of the phenomenon. Therefore, further research on the topic is welcome, especially considering that, usually, software engineering theories (as social theories) are not proven to be true. In particular, observational studies would be desirable to strengthen (or dispute) our theory. Given the existence of other taxonomies in the DevOps context, promising future work is conciliating them in a unified model. Such unification may also demand methodological advances: e.g., how to merge taxonomies and validate such an integration? We also believe many more insights and discussions can be derived from our **strong codes**, be it in academic or practitioners forums. In particular, a relevant implication for researchers is that each **strong code** could be submitted to validation studies or, at least, be a starting point to new studies.

Acknowledgments

We thank the support of the Brazilian Federal Service for Data Processing (Serpro), CNPq proc. 465446/2014-0, CAPES – Finance Code 001, and FAPESP procs. 14/50937-1, 15/24485-9, and 2019/12743-4.

References

[1] "ITIL 4^{th} edition, glossary," 2019, https://purplegriffon.com/downloads/resources/itil4-foundation-glossary-january-2019.pdf, accessed on Nov 2021.

[2] J. Fulmer, "What in the world is infrastructure," *PEI Infrastructure Investor*, vol. 1, no. 4, pp. 30–32, 2009.

[3] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Computing Surveys*, vol. 52, no. 6, pp. 127:1–127:35, 2019.

[4] D. A. A. Tamburri, R. Kazman, and H. Fahimi, "On the relationship between organisational structure patterns and architecture in agile teams," *IEEE Transactions on Software Engineering*, pp. 1–23, 2022, early access.

[5] K. Nybom, J. Smeds, and I. Porres, "On the impact of mixing responsibilities between devs and ops," in *International Conference on Agile Software Development*, ser. XP 2016. Springer International Publishing, 2016, pp. 131–143.

[6] M. Shahin, M. Zahedi, M. A. Babar, and 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*, ser. EASE'17. ACM, 2017, pp. 384–393.

[7] D. Lopez-Fernandez, J. Diaz, J. Garcia-Martin, J. Perez, and A. Gonzalez-Prieto, "Devops team structures: Characterization and implications," *IEEE Transactions on Software Engineering*, 2021, early access.

[8] R. W. Macarthy and J. M. Bass, "An empirical taxonomy of DevOps in practice," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, pp. 221–228.

[9] B. Glaser and A. Strauss, *The discovery of grounded theory: strategies for qualitative research*. Aldine Transaction, 1999.

[10] S. Gregor, "The nature of theory in information systems," *MIS quarterly*, pp. 611–642, 2006.

[11] J. Stoner and R. E. Freedman, *Administração*. Prentice-Hall, 1995.

[12] L. Donaldson, "Teoria da contingência estrutural," in *Handbook de estudos organizacionais*. Atlas, 1999.

[13] R. H. Hall, *Organizações, estruturas e processo*. Prentice-Hall, 1984.

[14] N. Oliveira and N. Takahashi, "Organizational structure, format, shape design and architecture," in *Automated organizations: Development and structure of the modern business firm*. Springer, 2012.

[15] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[16] F. C. Lunenburg, "Organizational structure: Mintzberg's framework," *International journal of scholarly, academic, intellectual diversity*, vol. 14, no. 1, 2012.

[17] D. E. Yeatts and C. Hyten, *High-Performing Self-Managed Work Teams: A Comparison of Theory to Practice*. Sage Publications, 1998.

[18] D. S. Pugh, D. J. Hickson, and C. R. Hinings, "An empirical taxonomy of structures of work organizations," *Administrative Science Quarterly*, vol. 14, no. 1, 1969.

[19] M. E. Sosa, S. D. Eppinger, and C. M. Rowles, "The misalignment of product architecture and organizational structure in complex product development," *Management Science*, vol. 50, no. 12, 2004.

[20] C. B. Seaman and V. R. Basili, "Communication and organization: an empirical study of discussion in inspection meetings," *IEEE Transactions on Soft. Engineering*, vol. 24, no. 7, pp. 559–572, 1998.

[21] J. Herbsleb and J. Roberts, "Collaboration in software engineering projects: A theory of coordination," in *International Conf. on Information Systems 2006 Proceedings*, ser. ICIS 2006, 2006, pp. 553–568.

[22] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, 2008, pp. 521–530.

[23] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Computing Surveys*, vol. 46, no. 1, 2013.

[24] P. Ralph, "Toward methodological guidelines for process theories and taxonomies in software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 712–735, 2019.

[25] M. Skelton and M. Pais, "Devops topologies," 2013, https://web.devopstopologies.com/, accessed on Nov 2021.

[26] ——, *Team Topologies: Organizing business and technology teams for fast flow*. IT Revolution Press, 2019.

[27] A. Mann, M. S. A. Brown, and N. Kersten, "2018 State of DevOps Report," 2018, https://puppet.com/resources/whitepaper/2018-state-of-devops-report, accessed on Jul 2019.

[28] L. Leite, G. Pinto, F. Kon, and P. Meirelles, "The organization of software teams in the quest for continuous delivery: A grounded theory approach," *Information and Software Technology*, vol. 139, p. 106672, 2021.

[29] A. Brown, M. Stahnke, and N. Kersten, "2020 State of DevOps Report," 2020, https://www2.circleci.com/2020-state-of-devops-report.html, accessed on Dec 2021.

[30] L. Leite, F. Kon, G. Pinto, and P. Meirelles, "Building a theory of software teams organization in a continuous delivery context," in *42nd International Conference on Software Engineering Companion*, ser. ICSE '20 Companion, 2020, pp. 294–295.

[31] L. Leite, G. Pinto, F. Kon, and P. Meirelles, "Platform teams: An organizational structure for continuous delivery," in *IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20, 2020, pp. 505–511.

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

[33] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," *Journal of Software: Evolution and Process*, vol. 29, no. 6, 2017.

[34] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *2016 IEEE/ACM 38th International Conference on Software Engineering*, ser. ICSE '16, 2016, pp. 120–131.

[35] M. Jovanović, A. Mas, A.-L. Mesquida, and B. Lalić, "Transition of organizational roles in agile transformation process: A grounded theory approach," *Journal of Systems and Software*, vol. 133, 2017.

[36] K.-J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transac. on Software Engineering and Methodology*, vol. 27, no. 3, 2018.

[37] W. C. Adams, "Conducting semi-structured interviews," in *Handbook of Practical Program Evaluation*, 3rd ed. Jossey-Bass, 2010.

[38] R. K. Yin, *Case Study Research, Design and Methods*, 4th ed. Sage Publications, 2009.

[39] E. Guba, "Criteria for assessing the trustworthiness of naturalistic inquiries," *Educational Technology Research and Development (ECTJ)*, vol. 29, pp. 75–91, 1981.

[40] D. G. Sirmon, M. A. Hitt, R. D. Ireland, and B. A. Gilbert, "Resource orchestration to create competitive advantage: Breadth, depth, and life cycle effects," *Journal of management*, vol. 37, no. 5, pp. 1390–1412, 2011.

[41] L. T. Pinfield, "A field evaluation of perspectives on organizational decision making," *Administrative Science Quarterly*, vol. 31, no. 3, pp. 365–388, 1986.

[42] P. A. Anderson, "Decision making by objection and the Cuban missile crisis," *Administrative Science Quarterly*, vol. 28, no. 2, pp. 201–222, 1983.

[43] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. ACM, 2016, pp. 273–284.

[44] H.-F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis," *Qualitative health research*, vol. 15, no. 9, pp. 1277–1288, 2005.

[45] B. Glaser, *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. The Sociology Press, 1978.

[46] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.

[47] R. Hoda, J. Noble, and S. Marshall, "The impact of inadequate customer collaboration on self-organizing agile teams," *Information and software technology*, vol. 53, no. 5, pp. 521–534, 2011.

[48] G. van Waardenburg and H. van Vliet, "When agile meets the enterprise," *Information and Software Technology*, vol. 55, no. 12, pp. 2154–2171, 2013.

[49] B. Glaser, *The grounded theory perspective III: Theoretical coding*. The Sociology Press, 2005, pp. 70.

[50] C. A. Hernandez, "Theoretical coding in Grounded Theory methodology," *Grounded Theory Review*, vol. 8, no. 3, 2009.

[51] M. Waterman, J. Noble, and G. Allan, "How much up-front?: A grounded theory of agile architecture," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, ser. ICSE '15, 2015, pp. 347–357.

[52] E. Woods, "Operational: The forgotten architectural view," *IEEE Software*, vol. 33, no. 3, pp. 20–23, 2016.

[53] M. G. Waterman, "Reconciling agility and architecture: a theory of agile architecture," Ph.D. dissertation, Victoria University of Wellington, 2014.

[54] M. Shahin and 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*, ser. ICSSP '20. ACM, 2020, pp. 175–184.

[55] C. Melo, "Productivity of agile teams: an empirical evaluation of factors and monitoring processes," Ph.D. dissertation, University of São Paulo, 2015.

[56] B. G. Glaser, "Conceptualization: On theory and theorizing using grounded theory," *International Journal of Qualitative Methods*, vol. 1, no. 2, pp. 23–38, 2002.

[57] P. E. Strandberg, "Ethical interviews in software engineering," in *International Symposium on Empirical Software Engineering and Measurement 2019*, ser. ESEM '19, 2019.

Leonardo Leite received the MSc degree and the PhD degree in Computer Science from the University of São Paulo (USP) respectively in 2014 and 2022. Since 2014, he is a software developer at the Brazilian Federal Service for Data Processing (Serpro), in which he has successfully promoted DevOps practices, such as the adoption of automated tests, deployment pipelines, continuous delivery, and monitoring.

Nelson Lago received the Bachelor's degree in music and the MSc degree in Computer Science from the University of São Paulo (USP) in 2000 and 2004. He is currently the Technical Manager of the USP FLOSS Competence Center, and works both as researcher and IT operations manager.

Claudia Melo received the MSc degree and the PhD degree in Computer Science from the University of São Paulo (USP) respectively in 2006 and 2013. Former professor at the University of Brasília (UnB) and head of technology for Latin America at ThoughtWorks. She is now a Director of Software Engineering/Tech Org Design at Loft.

Fabio Kon received a bachelor's degree in Music from the São Paulo State University in 1992 and a Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2000. He is a Full Professor at the University of São Paulo (USP) and carries out research in software engineering, agile methods, distributed systems, data science, and smart cities. He is the coordinator of the Brazilian National S&T Institute on the Future Internet for Smart Cities (https://interscity.org).

Paulo Meireles received the PhD degree in Computer Science from the University of São Paulo (USP) in 2013. He is an adjunct professor at the Federal University of ABC (UFABC). He also works as a collaborating researcher at USP, investigating Software Engineering, with a focus in Free Software and DevOps.