A grounded theory of organizational structures for development and infrastructure professionals in software-producing organizations

Leonardo Alexandre Ferreira Leite

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Doctor of Science

Program: Computer Science
Advisor: Prof. Paulo Meirelles
Coadvisor: Prof. Fabio Kon

São Paulo
May, 2022

A grounded theory of organizational structures for development and infrastructure professionals in software-producing organizations

Leonardo Alexandre Ferreira Leite

This version of the thesis includes the corrections and modifications suggested by the Examining Committee during the defense of the original version of the work, which took place on May 31, 2022.

A copy of the original version is available at the Institute of Mathematics and Statistics of the University of São Paulo.

Examining Committee:

Prof. Fabio Kon (advisor) – IME-USP
Prof. Guilherme Travassos – UFRJ
Prof. Breno de França – Unicamp
Prof. Jessica Diaz – UPM (Spain)
Prof. Carolyn Seaman – UMBC (USA)

Acknowledgements

First, I would like to express how extremely grateful I'm to my advisors, Prof. Paulo Meirelles and Prof. Fabio Kon, for all the support and trust in this research effort. It has been a great pleasure working with them. Also, their intense and direct involvement in this project's activities was vital for its success. In this sense, I would like to extend my sincere thanks to a few other scholars who have collaborated with us over these years: Dr. Dejan Milojicic, Prof. Carla Rocha, Prof. Gustavo Pinto, Dr. Claudia Melo, and (super) Nelson Lago. Their involvement was inspiring and motivating. I would also like to recognize the Professors who dedicated themselves to examining and providing valuable feedback on our work during the qualifying examination and the final defense (Prof. Rodrigo Bonifácio, Prof. Guilherme Travassos, Prof. Breno França, Prof. Jessica Diaz, and Prof. Carolyn Seaman).

Second, I would like to say that earning this Ph.D. title resulted from a long journey. Thus, I remember with affection the Professors who were part of my academic journey, especially Prof. Lucia Filgueiras, Prof. João José Neto, and Prof. Marco Aurélio Gerosa. Likewise, I would also like to acknowledge some of my IME classmates (Melissa, Tallys, Dylon, Siqueira, Tatiane, Marcelo, Giuliano, Paula, Haydée, Erika, Jean, Vanessa, Fatemeh, Luana, Isaque, and Tássio) for their companionship and the moments of small talk in the lab, studying for tests, and eating dinner at the University restaurant. Those were special moments, and I'm sorry that the pandemic limited our time together.

With regards to my job at Serpro (Brazilian Federal Service for Data Processing), I would like to thank my leaders and colleagues Roberta Monteiro and Daniel Cesar, who supported my academic activities by providing me the necessary work flexibility. I would also like to recognize them and my other colleagues for their interest in my research, especially Marcio Frayze and Julianno Martins (I promoted my research on "p de Podcast," their podcast on software architecture).

Third, I would also like to express my deepest gratitude to my family for their support and incentive throughout my doctoral years, especially Lari (my wife), Bertília (my mother),

Marco (my father), and Julia (my sister). Particularly, I would like to express my most profound appreciation and love to my wife, Lari, to whom I am forever thankful, especially considering the enormous amount of time I dedicated to this research. I would also like to mention the logistical support provided by Iranice (my mother-in-law). I also take the opportunity to tenderly remember Jaíra (my grandmother), who left us just before the beginning of this research, and to lovely welcome Dante, my son, who will be born very soon.

I would be remiss in not mentioning all the family members, friends, and colleagues who (in person or virtually) attended my thesis defense. In particular, I'd like to acknowledge the family people and close friends who helped me practice for my defense presentation: Bertília (my mother), Leo (my step-father), Koga, Gui, and Tássio. And also, the other people who supported me in this preparation (e.g., Rafael Fernandes, Lara Vacaro, Prof. Alfredo Goldman, and Prof. Afonso Ferreira), thanks to them too. Special thank to Julia (my sister) for the support during the presentation.

Last, I would like to mention the 75 amazing professionals who honored me with the chance to discuss their work environments and the colleagues who referred those interviewees to me. Their spirit of collaboration was the raw material for this thesis. To each interviewee, I say: "a piece of your soul lives on in these pages! Thank you so much again and again!!!" I hope this same spirit of collaboration continues to foster the evolution of the communities involved in the great endeavor of software production.

Resumo

Leonardo Alexandre Ferreira Leite. **Uma teoria fundamentada sobre estruturas organizacionais para profissionais de desenvolvimento e de infraestrutura em organizações produtoras de software**. Tese (Doutorado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

DevOps e entrega contínua têm impactado as estruturas organizacionais dos grupos de desenvolvimento e infraestrutura em organizações produtoras de software. Nossa pesquisa visa revelar as diferentes opções adotadas pela indústria de software para organizar tais grupos, entender por que diferentes organizações adotam estruturas distintas e descobrir como as organizações lidam com as desvantagens de cada estrutura. Ao entrevistar 68 qualificados profissionais de TI, cuidadosamente selecionados, e analisar essas conversas por meio de um processo de Teoria Fundamentada (Grounded Theory), identificamos condições, causas, razões para evitar, consequências e contingências relacionadas a cada estrutura descoberta (departamentos segregados, departamentos colaborativos, departamentos mediados por API e departamento único). Esta tese, então, propõe uma teoria para explicar estruturas organizacionais para profissionais de desenvolvimento e infraestrutura. Essa teoria pode apoiar profissionais e pesquisadores na compreensão e discussão do fenômeno DevOps e seus problemas relacionados, e também fornece informações valiosas para a tomada de decisões dos profissionais.

Palavras-chave: DevOps. estruturas organizacionais. entrega contínua. times de software. engenharia de software. Teoria Fundamentada.

Abstract

Leonardo Alexandre Ferreira Leite. **A grounded theory of organizational structures for development and infrastructure professionals in software-producing organizations**. Thesis (Doctorate). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

DevOps and continuous delivery have greatly 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. By interviewing 68 carefully-selected, skilled IT professionals and 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, collaborating departments, API-mediated departments, and single department). We, then, 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; it also provides valuable input to practitioners' decision-making.

Keywords: DevOps. organizational structures. continuous delivery. software teams. software engineering. Grounded Theory.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In this thesis, we offer a theory to explain organizational structures for development and infrastructure professionals. We describe, in the taxonomy format, the structures currently adopted by software-producing organizations. Moreover, we explain why different organizations have different structures and how companies handle each structure's drawbacks. In other words, we present conditions, causes, reasons to avoid, consequences, and contingencies related to each structure.

In this chapter, we initially explain the underlying problem motivating this research. We then present our research questions and how we organized our research in three phases around such questions. While introducing these research phases, we concisely indicate the employed research methods. We, then, highlight the articles and contributions produced by our research. Finally, we delineate how we arranged the chapters of this thesis.

1.1 Problem outline

To remain competitive, many software-producing corporations seek to speed up their release processes [Sch+16b; HM11]. Organizations may adopt continuous delivery practices in their quest to accelerate time-to-market and improve customer satisfaction [Che15]. However, continuous delivery also comes with challenges, including profound impacts on various aspects of software engineering [Sch+16a]. 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 an organization's structure [Che15]. The lack of clear roles and obligations may induce lengthy negotiations and eventual stress between IT departments [NSP16]. Therefore, organizations moving toward continuous delivery have not only to upgrade their software tooling arsenal but also to find ways to better shape and integrate their IT teams.

Given such recent transformations, there is a need for better understanding the organizational structures the software industry adopts for development and infrastructure employees. By organizational structure, we mean the differentiation (division of labor) and

integration (interaction) [Oli12] of operations activities (application deployment, infrastructure setup, and service operation in run-time) between development and infrastructure groups. An acclaimed structure toward continuous delivery, advocated by Amazon, is organizing cross-functional teams to break the existing silos among development and infrastructure professionals [Gra06]. However, Davis and Daniels alert that part of the community erroneously thinks of silos and cross-functional teams as the only two options of existing structures [DD16].

Empirical software engineering studies have focused on identifying the different organizational structures adopted in the industry to arrange development and infrastructure professionals [NSP16; Sha+17; Lop+21; MB20]. However, the current literature does not reveal why different companies adopt different structures or how companies deal with the drawbacks of each structure.

This lack of research is inconvenient due to at least two crucial reasons: (1) organizations wishing to adopt continuous delivery can be disoriented regarding how to design their human resources structure toward this goal; (2) once a structure is chosen, the organization might be unaware of the consequences of this choice. For example, adopting cross-functional teams can solve many problems but also brings new challenges, such as the difficulty in guaranteeing that each team in the organization masters the required technical skills. The lack of a consolidated view of how the industry handles the outlined problem is also unfortunate. 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.

1.2 Research questions

To mitigate the gap presented in the problem outline, in this thesis we address the following **research questions**:

RQ1: *What are the organizational structures adopted by software-producing organizations 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?*

RQ2: *Why do different software organizations adopt different organizational structures regarding development and infrastructure groups?*

RQ2.1: *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, henceforward, we refer to them simply as “organizations” or “companies.”

DevOps and continuous delivery are contextual factors that have impacted the software industry and are, therefore, part of the motivation for our research. Nevertheless, we clarify that the definitions and procedures of our research are not dependent on the notion of DevOps. We seek to understand the software production landscape regardless of whether companies claim to have adopted DevOps or not.

1.3 Research phases

Our research questions, addressing the problem outlined above, were not a fruit of inspiration solely. Our initial interest in this research endeavor was the DevOps topic. Thus, initially, we conducted a comprehensive literature review on DevOps [Lei+19], which was the first phase of this doctoral research. We wanted mainly to understand the DevOps field, with its concepts and challenges, and find research opportunities before working on an original contribution. Such a literature review employed some techniques of Systematic Literature Reviews (SLR) that we selected based on the literature recommendations [KC07; BB06] and in our own previous experience in conducting an SLR [Lei+13]. We present some of the results of our review in Chapter 2 to introduce some basic concepts to this thesis and to delineate better the problem guiding this doctoral research. Figure 1.1 summarizes the phases of this doctoral research, which we explain in this section.

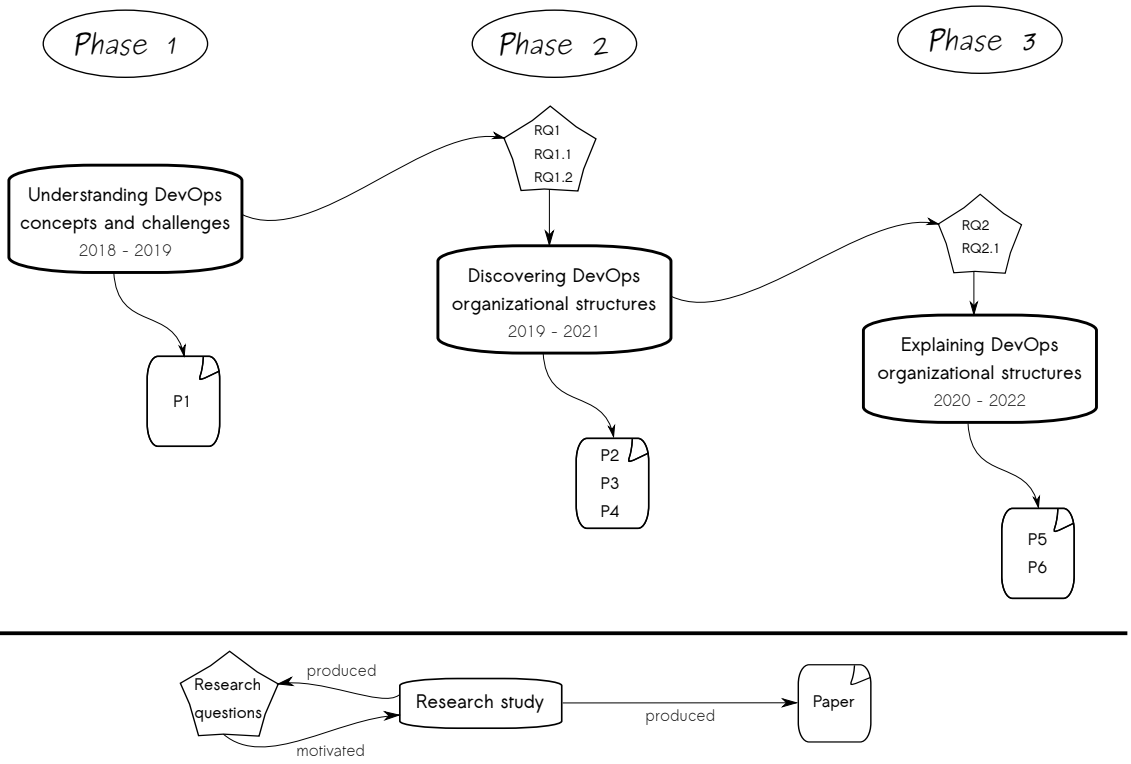

Figure 1.1: *Research phases of this doctoral research*

In particular, one of the “DevOps challenges” found in our literature review concerned

the challenge of splitting roles across teams in the DevOps paradigm. The most basic options would be *(i)* having developers and operations as different but cooperating groups or *(ii)* having in each team both developers and infrastructure specialists. However, how the industry is indeed tackling this issue, which options of organizing teams are available and being adopted, and which results come from different options were all questions not adequately addressed by the academic literature at that time.

In particular, in our DevOps review, we found no work using an empirical process to reveal the existing structures across the IT industry. Based on this need, we defined **RQ1** and its sub-questions. The main idea was to elaborate a taxonomy to describe the existing structures (**RQ1.1**) based on empirical data, thus initiating a *theory to describe* [Gre06]. To answer these questions, we conducted semi-structured interviews with 37 IT professionals, each one belonging to a different organization. We analyzed these interviews by following Grounded Theory [GS99], a methodology well-suited for generating theories. We explain this process in detail in Chapter 3.

To describe the structures, we linked to each of them **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. The structures (the elements of the highest level in our taxonomy) we found, as a product of our research Phase 2, were:

- **Siloed departments**, with highly bureaucratized interaction between development and infrastructure groups.
- **Classical DevOps**, focusing on facilitated communication and collaboration between development and infrastructure groups.
- **Platform teams**, in which the infrastructure team provides highly-automated infrastructure services to assist developers.
- **Cross-functional team**, in which a team takes responsibility for both software development and infrastructure management.

At the same time, at this point, we were eager to assess if a given structure could be somehow "better" than another. The dimension chosen for comparison was delivery performance (explained in Section 2.2), thus leading us to the **RQ1.2** formulation. About this research question, a notable result was the evidence that **platform teams** seemed to lead to better outcomes in terms of delivery performance. We present the results of Phase 2 in Chapter 4.

After having these preliminary results, it was clear to us that there are distinct organizational structures in the industry to organize development and infrastructure professionals. This situation led us to formulate **RQ2**: would there be legitimate reasons for different organizations to adopt different structures? Why do not all organizations adopt the "best" structure? Our intuition was that there was no "best" structure but different structures that were more suitable for different contexts. In this case, probably each structure would have its drawbacks. And if so, how would organizations handle such drawbacks (**RQ2.1**)? Pursuing such answers would lead us to extend our theory to be not only a *theory to describe* but also a *theory to explain* [Gre06] the examined phenomenon (the existence of

distinct structures to organize development and infrastructure professionals).

The search for such explanations was the core of our research Phase 3, which also followed a Grounded Theory process. In this phase, we conducted semi-structured interviews with 31 IT professionals working in 25 different companies, summing up 68 semi-structured interviews throughout our research. The research procedures in this phase are also presented in our chapter devoted to the research approach. Chapter 3 provides a complete view of our investigation methods.

The explanations we offer about the structures, as the product of Phase 3, are provided in the format of conditions, causes, reasons to avoid, consequences, and contingencies related to each discovered structure (this format corresponds to one of the theoretical coding families provided by the Grounded Theory methodology).

Phase 3 had also an additional goal: to interact with practitioners to somehow assess our taxonomy and improve it when necessary. During this process, we altered the taxonomy by adding, removing, and renaming its high-level elements (structures and supplementary properties). In particular, the structures were renamed as follows:

- **Siloed departments** to **segregated departments**.
- **Classical DevOps** to **collaborating departments**.
- **Platform teams** to **API-mediated departments**.
- **Cross-functional team** to **single department**.

Therefore, as a final product of our research endeavor, we offer a *theory to explain* the organizational structures of development and infrastructure professionals in the context of contemporary software production. We present such final results in Chapter 5.

1.4 Produced articles

Our literature review, conducted in the first research phase, is published in the ACM Computing Surveys journal as the article "A Survey of DevOps Concepts and Challenges."

Paper 1 - published

Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, Paulo Meirelles
A Survey of DevOps Concepts and Challenges [Lei+19]
ACM Computing Surveys, 52(6):127:1–127:35, 2019
Official publication on https://dl.acm.org/doi/abs/10.1145/3359981.
Version freely accessible on
http://ime.usp.br/~leofl/devops/2019-12-04/devops-survey-published.html.

In addition to the doctoral candidate and his advisors, other two researchers contributed to the survey, namely Prof. Carla Rocha (University of Brasília - UnB) and Dr. Dejan Milojicic (senior researcher at HP Labs in Palo Alto). The survey has being highly cited by the academic community, reaching, in April 2022, 156 citations according to Google Scholar and 4,346 downloads from the ACM Digital Library.

While building the initial version of our taxonomy, after interviewing 27 IT practitioners, we published a brief overview of our preliminary results. The main goal, at that stage, was to gather internationally qualified feedback on our ongoing research. Such overview is presented in the paper "Building a Theory of Software Teams Organization in a Continuous Delivery Context," published as an extended abstract for the ICSE 2020 Poster Track session.

Paper 2 - published

Leonardo Leite, Gustavo Pinto, Fabio Kon, Paulo Meirelles
Building a Theory of Software Teams Organization in a Continuous Delivery Context [Lei+20b]
IEEE/ACM 42nd International Conference on Software Engineering (ICSE 2020) – Poster Track
Official publication on https://ieeexplore.ieee.org/abstract/document/9270342.
Extended abstract, poster, and video available on
http://ime.usp.br/~leofl/devops/2020-07-06/icse-poster-track.html.

Considering the literature already studied in our DevOps Survey, the existence of the three first organizational structures (siloed departments, classical DevOps, and cross-functional teams) was somehow expected. On the other hand, the emergence of "platform teams" as a distinctive structure was remarkable, mainly because we found evidence that such a structure presents better delivery performance results. To promote this emergent and relevant concept, we produced a paper for an ICSE workshop, entitled "Platform Teams: An Organizational Structure for Continuous Delivery."

Paper 3 - published

Leonardo Leite, Gustavo Pinto, Fabio Kon, Paulo Meirelles
Platform Teams: An Organizational Structure for Continuous Delivery [Lei+20a]
6th International Workshop on Rapid Continuous Software Engineering (RCoSE 2020), held in conjunction with the IEEE/ACM 42nd International Conference on Software Engineering (ICSE 2020)
Official publication on https://dl.acm.org/doi/abs/10.1145/3387940.3391455.
Version freely accessible on
http://ime.usp.br/~leofl/devops/2020-03-17/platform-teams.html.

Afterward, having conducted and analyzed 37 interviews, we further evolved our taxonomy. We published the full description of this version, including **core** and **supplementary properties**, in the Information and Software Technology journal. The paper is entitled "The Organization of Software Teams in Quest for Continuous Delivery: a Grounded Theory Approach," and we present its results in Chapter 4.

Paper 4 - published

Leonardo Leite, Gustavo Pinto, Fabio Kon, Paulo Meirelles
The Organization of Software Teams in Quest for Continuous Delivery: a Grounded Theory Approach [Lei+21]
Information and Software Technology, 139, 2021
Official publication on
https://www.sciencedirect.com/science/article/abs/pii/S0950584921001324.
Version freely accessible on https://arxiv.org/abs/2008.08652.

Besides the doctoral candidate and his advisors, Prof. Gustavo Pinto (Federal University of Pará - UFPA) also contributed to the initial formulation of our taxonomy, having coauthored Papers 2, 3, and 4.

We described our initial plans to address **RQ2**, applying the Case Study methodology [Yin09], in an article accepted in a thesis workshop (WTDSoft). The article is entitled "Understanding context and forces for choosing organizational structures for continuous delivery." The content of this paper provided input for fruitful discussions with the qualifying examination board (Profs. Guilherme Travassos and Rodrigo Bonifácio). After such discussions, we concluded that the conduction of case studies was not a proper path to take.

Paper 5 - published

Leonardo Leite, Fabio Kon, Paulo Meirelles
Understanding context and forces for choosing organizational structures for continuous delivery [LKM20]
10^{th} CBSoft's Workshop on Theses and Dissertations (WTDSoft 2020)
Available on https://sol.sbc.org.br/index.php/cbsoft_estendido/article/view/14609.

The final results of our research focusing on answering **RQ2**, grounded on data from seven brainstorming sessions and 37 semi-structured interviews, are in the paper entitled "A theory of organizational structures for development and infrastructure professionals," submitted to the IEEE Transactions on Software Engineering journal. Chapter 5 reports

its results.

Paper 6 - submitted (under review)

Leonardo Leite, Nelson Lago, Claudia Melo, Fabio Kon, Paulo Meirelles
A theory of organizational structures for development and infrastructure professionals
Submitted to the IEEE Transactions on Software Engineering (TSE) journal.
Preprint available on
https://www.techrxiv.org/articles/preprint/A_theory_of_organizational_structures_for_development_and_infrastructure_professionals/19210347.

In addition to the doctoral candidate and his advisors, Nelson Lago and Dr. Claudia Melo also contributed to the final development of our theory. Mr. Lago is currently the Technical Manager of the USP FLOSS Competence Center, working both as a researcher and IT operations manager. Dr. Melo, former professor at the University of Brasília (UnB) and head of technology for Latin America at ThoughtWorks, is now a Director of Software Engineering/Tech Org Design at Loft.

1.5 Contributions

Our first contribution is to provide an empirically derived taxonomy to describe the structures currently adopted by the software industry to organize development and infrastructure professionals (answering **RQ1**). We provide a detailed presentation of each structure with **core** and **supplementary properties** (answering **RQ1.1**). Although there are other taxonomies in the literature, our taxonomy is valuable too because it is empirically derived (many of the others are not). It is also worthy because empirical studies in the sociological field (and, thus, in software engineering, too) are rarely definitive [Sir+11; Pin86; And83]: multiple studies must be carried out independently so that the community can reach a consensus.

Our taxonomy brings the following key benefits: *(i)* it helps practitioners to differentiate **collaborating departments** from a **single department**, which were traditionally blended under the term DevOps [HM11; FJT16], and *(ii)* it highlights the **API-mediated departments** as a promising and emergent alternative for organizations – our taxonomy was the first empirically derived taxonomy to present such a structure (other researchers working concurrently with us also found equivalent structures [Lop+21]). In this context, a valuable contribution we offer is providing empirical evidence that **API-mediated departments** seem to lead to better outcomes in terms of delivery performance (answering **RQ1.2**).

Our second significant contribution is to provide an explanatory dimension for each of the structures presented in our taxonomy. We explicitly associated conditions, causes,

reasons to avoid, consequences, and contingencies related to each structure. In this way, practitioners can understand why different organizations adopt different structures (answering **RQ2**). They can also learn how organizations handle the drawbacks of each structure (answering **RQ2.1**) – or, at least, be aware of such drawbacks. Other researchers presenting similar taxonomies did not attribute such level of explanation to their structures.

In summary, this doctoral thesis contributes to the area by presenting a systematically-derived theory of organizational structures, based on recent field observations and employing a well-accepted methodology.

These contributions have implications for multiple stakeholders. By increasing the awareness of organizational structures in the community, practitioners can make more informed decisions on structure selection and drawback handling. Moreover, we hope our theory can lead scholars to a deeper comprehension of the software production phenomenon so they can update software engineering classes considering the reality of the industry abstracted under our theory. We support the validity of our findings in Chapter 6. We also provide more details on the relevance of the referred contributions in our conclusions (Chapter 7) by stating their implications for practice and research.

1.6 Thesis structure

After this introduction, Chapter 2 presents some basic concepts for this thesis, including DevOps and delivery performance, which we learned in our research Phase 1. Chapter 2 also discusses works related to organizational structures in the general literature and in the software engineering and DevOps contexts. Chapter 3 details our research approach, evidencing its base methodology, Grounded Theory. We present our results and related discussions in Chapters 4 and 5, respectively, related to Phase 2 and 3. After that, we discuss our quality criteria and the limitations of this research in Chapter 6. Finally, Chapter 7 draws our conclusions.

Chapter 2

Background

This chapter presents some of the fundamental concepts underlying our research effort, especially DevOps and delivery performance. In particular, we expand the research problem outlined in the introduction considering the literature on DevOps.

We also present the literature on organizational structures. We highlight how these studies deal with the balance of specialization vs. integration of teams, which highly relates to our proposed taxonomy for organizing development and infrastructure professionals. To provide a broad perspective for the reader, we also introduce other views on organizational structures, especially the view of complex systems. We then debate on how part of this literature dismisses the concern with structures in favor of tackling cultural issues, concluding that such a view is controversial. Finally, we also explore the "specific literature" on organizational structures in the context of software production, that is our related work.

2.1 DevOps

Seeking to understand DevOps, we initiated our research by conducting a literature review searching for papers with the keyword "DevOps". In this review, we thoroughly examined 50 scientific papers, which were carefully selected after we initially found almost 200 articles somehow related to DevOps. We noted that publications of papers on DevOps grew by a factor of four from 2014 to 2015, reaching a peak in 2016 (see Figure 2.1). We consider this high number of publications in the years preceding our study as an indication of how relevant such a theme has become to academia.

By analyzing the 50 selected papers, we: devised conceptual maps organizing DevOps concepts into four categories: process, people, delivery, and runtime (see Figures 2.2, 2.3, 2.4, 2.5, and 2.6); linked these concepts to the engineer's and manager's perspectives; presented DevOps tools, classifying them and associating them to DevOps concepts (see Table 2.1); listed significant DevOps implications for engineers, managers, and researchers; and detected four fundamental DevOps challenges. Our literature review was published in the ACM Computing Surveys journal [Lei+19].

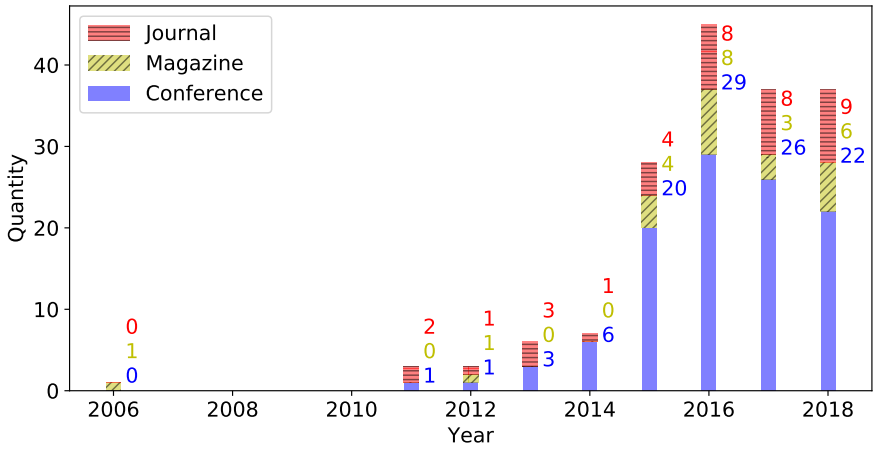

Figure 2.1: *Publications about DevOps found in our survey by source types and publication year*

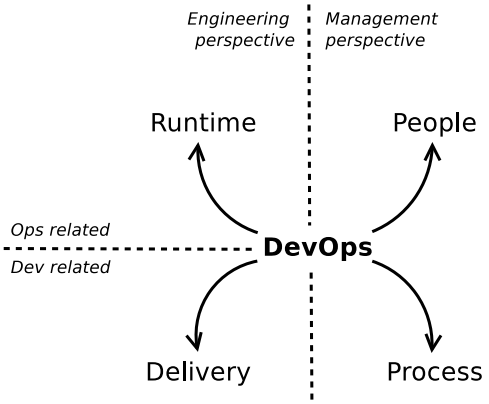

Figure 2.2: *DevOps overall conceptual map*

We now present some topics related to DevOps, most of them taken from our survey, relevant to the rest of this thesis. Thus, we discuss the history of DevOps, our DevOps definition, and the found DevOps challenges.

2.1.1 History

DevOps is an evolution of the agile movement [Chr16]. Agile Software Development advocates small release iterations with customer reviews. It assumes that a team can release software frequently in a production-like environment. However, pioneer Agile literature puts little emphasis on deployment-specific practices. No Extreme Programming (XP) practice, for example, is about deployment [BA04]. The same sparse attention to deployment is evident in traditional software engineering processes [KK03] and books [Pre05; Som11]. Consequently, the transition to production tends to be a stressful process in organizations, containing manual, error-prone activities, and, even, last-minute corrections [HF10]. DevOps proposes a complementary set of agile practices to enable the iterative delivery of software in short cycles effectively.

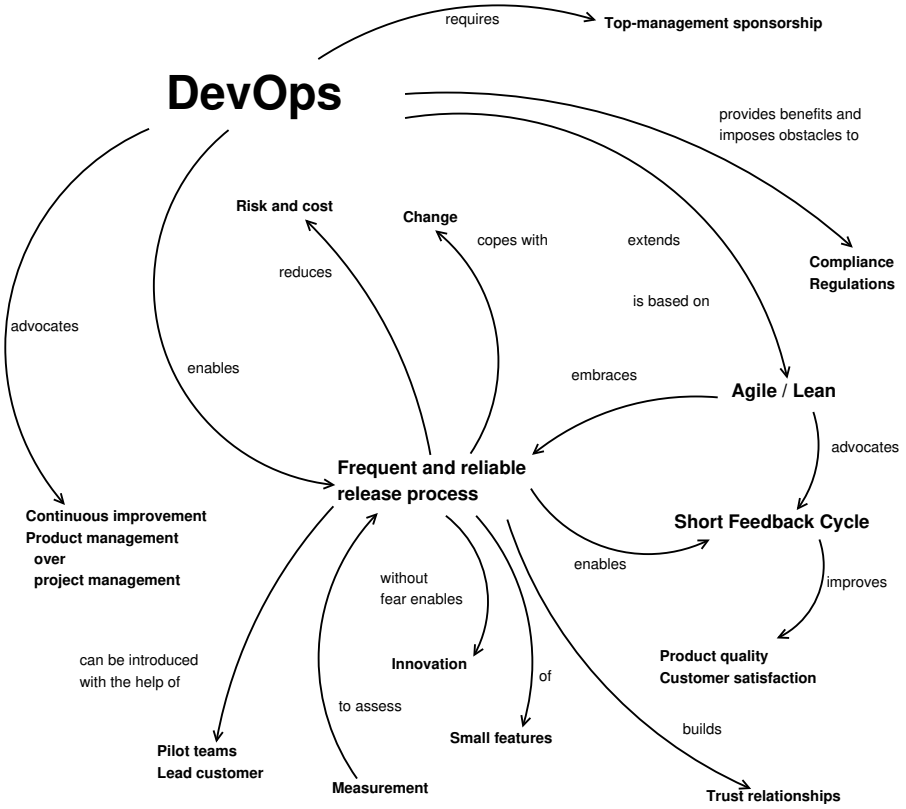

Figure 2.3: *DevOps conceptual map: process*

From an organizational perspective, the DevOps movement promotes closer collaboration between developers and operators. The existence of distinct silos for operations and development is still common: operations staff are responsible for managing software modifications in production and for service levels [Woo16]; development teams, on the other hand, are accountable for continuously developing new features to meet business requirements. Each one of these departments has its independent processes, tools, and knowledge bases. The interface between them in the pre-DevOps era was usually a ticket system: development teams demanded the deployment of new software versions, and the operations staff manually managed those tickets.

In such an arrangement, development teams continuously seek to push new versions into production, while operations staff attempt to block these changes to maintain software stability and other non-functional concerns. Theoretically, this structure provides higher stability for software in production. However, in practice, it also results in long delays between code updates and deployment, as well as ineffective problem-solving processes, in which organizational silos blame each other for production problems.

Conflicts between developers and operators, significant deployment times, and the need for frequent and reliable releases led to inefficient execution of agile processes. In this context, developers and operators began collaborating within enterprises to address this gap. This movement was coined “DevOps” in 2008 [Deb08].

In the *Continuous Delivery* book [HF10], Humble and Farley advocate for an *automated deployment pipeline*, in which any software version committed to the repository must be a

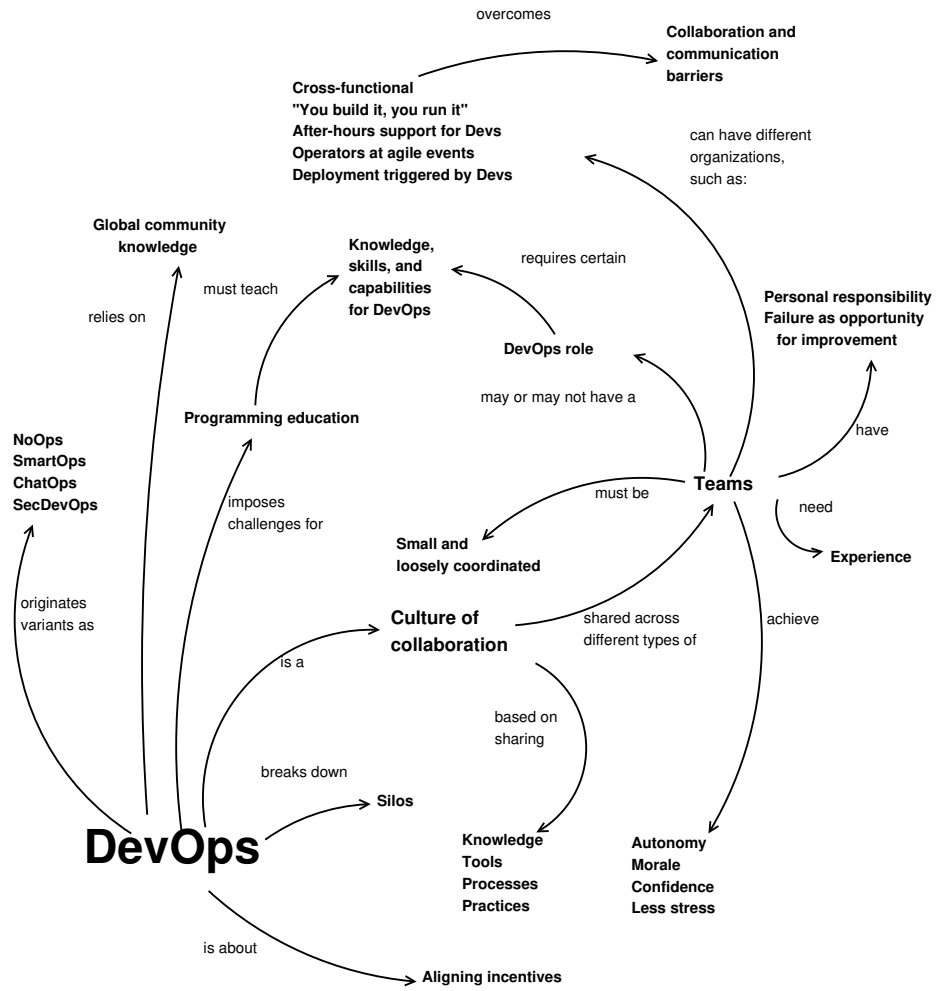

Figure 2.4: *DevOps conceptual map: people*

production-candidate version. After passing through stages, such as compilation and automated tests, the software is sent to production by the press of a button. This process is called *Continuous Delivery*. A variant is *continuous deployment* [Hum10], which automatically sends to production every version that passes through the pipeline. Many authors closely relate DevOps to continuous delivery and deployment [SBZ16; WAL15b; YK16; WAL15a]. Yasar, for example, calls the deployment pipeline a "DevOps platform" [YK16].

Besides automating the delivery process, DevOps initiatives have also focused on using automated runtime monitoring for improving software runtime properties, such as performance, scalability, availability, and resilience [Chr16; Bas+16; BHJ16; Roc13]. From this perspective, "Site Reliability Engineering" [Bey+16] emerged as a new term related to DevOps work at runtime.

2.1.2 Definitions

The first impact when researching DevOps is to perceive that even after a decade of discussions, this term still lacks a widely accepted definition. By consolidating the most

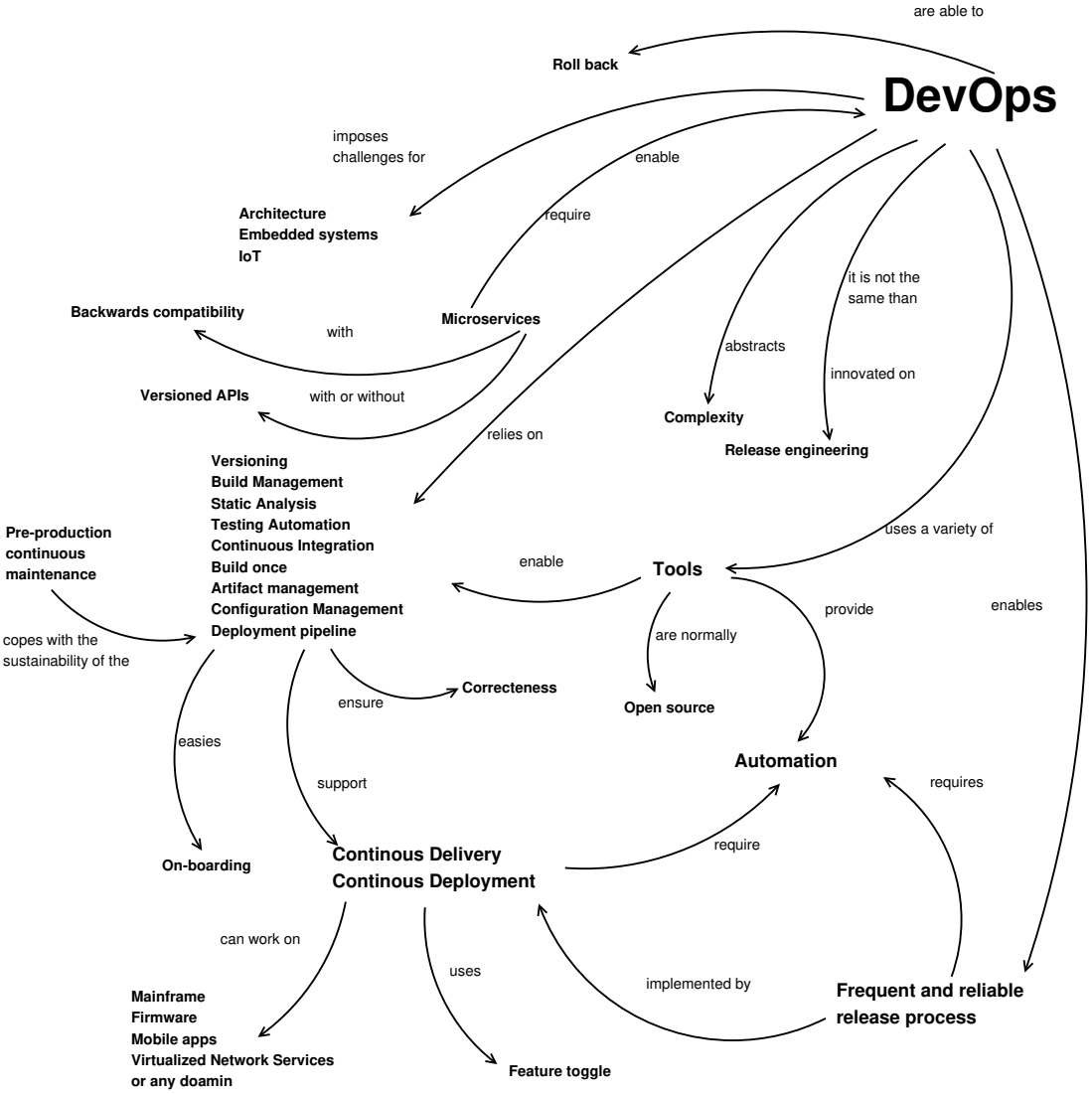

Figure 2.5: *DevOps conceptual map: delivery*

cited definitions of DevOps, we crafted our definition, similar to the one proposed by Dyck *et al.* [DPL15]:

> *DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability.* [Lei+19]

There is a current trend in the industry of adopting DevOps as a role, the "DevOps engineer" [HCM17]. But our DevOps definition is according to the original spirit of the DevOps movement, which was about breaking down the silos ("*DevOps is a collaborative and multidisciplinary effort within an organization*"). This spirit was portrayed, for example, by the novel *The Phoenix Project* [KBS18], authored by a prominent figure of the DevOps movement. Our definition is also in accordance with a shared view in the literature that closely relates DevOps to continuous delivery ("*to automate continuous delivery of new software versions*"), evidenced by the title of the seminal paper on DevOps "Why enterprises must adopt DevOps to enable continuous delivery" [HM11]. Finally,

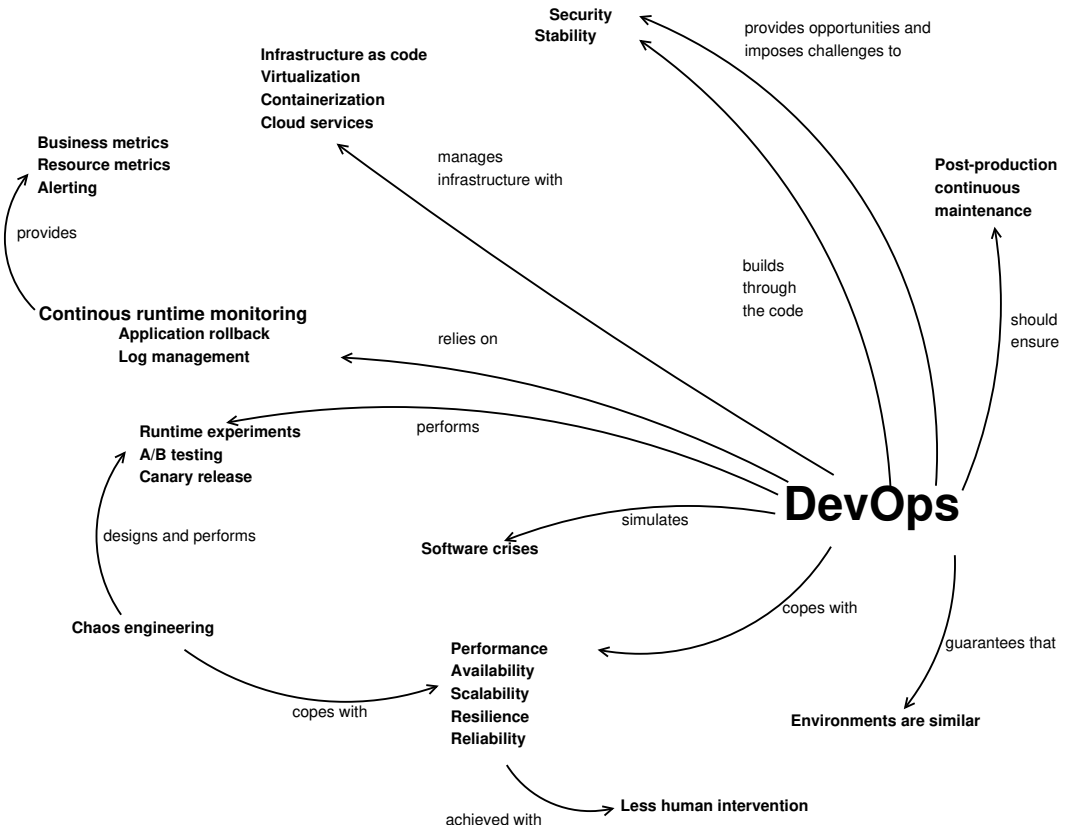

Figure 2.6: *DevOps conceptual map: runtime*

our definition is also aligned with more recent developments that face the fact that it is not enough to deliver faster, but also that reliability is paramount ("*while guaranteeing their correctness and reliability*"), such as the discussion presented in the *Site Reliability Engineering* book [Bey+16].

Other definitions in the DevOps context are relevant to this thesis. First, **infrastructure** refers to all of the hardware, software, networks, facilities, etc. that are required to build and operate IT services, according to the ITIL glossary [Fou19]. 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 [Ful09]. 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. We adopted this careful definition to include in the infrastructure work not only physical machines but also virtualized machines and even deployment platforms, such as Kubernetes[1].

Second, we are considering **development teams** (also called **product teams**) as responsible for developing business services. On the other hand, the **infrastructure staff** uniformly provides computational resources for diverse applications. This second definition aligns with Woods' view about **infrastructure engineers**: the ones providing the infrastructure services the system relies on [Woo16].

[1] https://cloud.google.com/google/kubernetes

Category	*Examples*	*Actors*	*Goals*	*Concepts*
Knowledge sharing	Rocket Chat GitLab wiki Redmine Trello	Everyone	Human collaboration	Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Source code management	Git SVN CVS ClearCase	Dev Ops	Human collaboration Continuous delivery	Versioning Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Build process	Maven Gradle Rake JUnit Sonar	Dev	Continuous delivery	Release engineering Continuous delivery Automation Testing automation, Correctness Static analysis
Continuous Integration	Jenkins GitLab CI Travis Nexus	Dev Ops	Continuous delivery	Frequent and reliable releases Release engineering Continuous integration Deployment pipeline Continuous delivery, Automation Artifact management
Deployment automation	Chef, Puppet Docker Heroku Open Stack Rancher Flyway	Dev Ops	Continuous delivery Reliability	Frequent and reliable releases Release engineering Configuration management Continuous delivery Infrastructure as code Virtualization, Containerization Cloud services, Automation
Monitoring & Logging	Nagios Zabbix Prometheus Logstash Graylog	Ops Dev	Reliability	You built it, you run it After-hours support for Devs Continuous runtime monitoring Performance, Availability, Scalability Resilience, Reliability, Automation Metrics, Alerting, Experiments Log management, Security

Table 2.1: *Major DevOps tools related to actors and DevOps concepts*

Third, in this work, we use the term **operations** to refer to the operations activities, such as application deployment, infrastructure setup, and service operation in run-time. We avoid the term operators or even operations staff. In some companies, there are workers dedicated to such tasks. As put by Woods: operations staff accept and operate new and changed software in production and are responsible for its service levels [Woo16]. However, in the pre-DevOps context, such tasks were usually attributed to the professionals in charge of providing the infrastructure. Our research seeks to understand how these activities were redistributed between development and infrastructure professionals with the DevOps advent. We, therefore, prefer to use the term infrastructure professionals instead. Still, in other industries, it is common to apply the term operator to menial workers who control

machines, so we would have engineers projecting machines and operators operating them. If the "machine" built by software engineers is the software system, operators could be the system users. This use could be adequate to situations in which users are employees of the client (who demanded the system), as it would be the case for supermarket cashiers. Clearly, this use is not related to the "ops" in DevOps.

2.1.3 Challenges

The four DevOps challenges we reported in our survey were:

1. How to re-design systems toward continuous delivery, especially considering the relation between DevOps practices and microservices architecture [BHJ16; Ebe+16].
2. How to organize the relations between development and infrastructure professionals to adopt DevOps, especially considering that mutual support between devs and ops is a very different thing from cross-functional teams [NSP16; Gra06].
3. How to assess the quality of DevOps practices in organizations, considering survey and system data and the pitfalls of any metric when used to promote or punish professionals [FK18; Bro18].
4. How to qualify engineers for DevOps practice, discussing the laborious endeavor of professors preparing practical DevOps classes and evaluating students' assignments [Chr16; Bai+18].

We selected the second listed challenge as our guide to this doctoral research[2]. Based on this initial elaboration, we derived our research questions. Chapter 2.4 presents more works related to the problem of structuring development and infrastructure professionals within software-producing organizations. Subsequently, we reproduce the second challenge as published in the DevOps survey.

How to deploy DevOps in an organization

A hard question not yet fully answered by the literature is how – perhaps whether – an organization should be restructured to adopt DevOps. We believe the literature is incomplete and even contradictory regarding this subject. We discuss a few studies that have evaluated this matter.

The seminal paper of Humble and Molesky about DevOps presents three scenarios. First, preserving the structures of development and operations departments, DevOps is led by human collaboration between developers and operators, with operators attending agile ceremonies and developers contributing to incident solving [HM11; Jaa18]. Alternatively, product teams, also called cross-functional teams, effectively incorporate operators. Finally, in a third scenario, one product team is composed of only operators offering support services (continuous integration, monitoring services) to the entire organization.

[2] Based on our experience on Software Engineering and research, we considered the second challenge to be more manageable from an academic perspective, being more a sociological inquiry than an engineering endeavor.

Similarly, Nybom et al. also present three distinct scenarios to the adoption of DevOps [NSP16]: *i)* collaboration between development and operations departments; *ii)* cross-functional teams; and *iii)* creation of "DevOps teams." These approaches are summarized in Table 2.2 and discussed in detail in the following paragraphs.

Collaborating departments
Development and operations departments collaborate closely.
It implies overlapping of developers and operators responsibilities.
Downside: new responsibilities can be unclear for employees.
Cross-functional team
The product team is responsible for deploying and operating (*You built it, you run it*).
Recommended by Amazon and Facebook.
Downside: requires more skilled engineers.
DevOps team
Acts as a bridge between developers and operators.
It is better accepted when it is a temporary strategy for cultural transformation.
Downside: risk of creating a third silo.

Table 2.2: *DevOps adoption approaches*

In the first approach, developers' responsibilities overlap with operators' [NSP16; CSA15; SBZ16; DMC16]. A high risk for adopting this strategy occurs when new responsibilities do not become evident in the organization [NSP16; CSA15], which could lead to frictions between developers and operators [Che15]. This is especially true when delivery automation is not prioritized [NSP16].

Cross-functional teams resemble the "whole team" practice from XP [BA04], and seems to prevail in literature [FFB13; Gra06; Cuk13; BHJ16]. This structure appeals to "T-shaped" professionals, who have expertise in few fields and basic skills in multiple correlated areas [Deb11; Gue91]. In this model, at least one team member must master operations skills. From this perspective, the so-called "DevOps engineer" [HCM17], also called the "full stack engineer" [HCM17], is known as a developer with operations skills.

One could wonder whether it makes sense to talk about collaborations between developers and operators in a cross-functional team context. Adopting cross-functional teams may just be a matter of moving operators to product teams. However, this is not trivial, considering that large organizations usually have only a small pool of operators for several development teams [NSP16]. Therefore, companies must hire people with both development and operations skills or train some of their developers in operations skills. Nevertheless, pressuring developers to learn operations can be ineffective, as they are already overwhelmed with other skills they need to learn. Transforming development teams into product teams has still another major cultural shift, as developers become responsible for 24/7 service support [SBZ16; Deb11]. This is true even when services should be designed to remain available without human interaction [Ham07]. An extra challenge for cross-functional teams is guaranteeing that specialists interact with their peer groups to share novelties in the field [Deb11]. Spotify, for example, achieves this through inter-team "guilds," which are communities of common interest within the organization [Kni14].

Assigning a "DevOps team" as a bridge between developers and operators [NSP16] has become a trend [Vel+14]. However, it is not clear what precisely "DevOps teams" are, and how they differ from regular operations teams [NSP16; Vel+14]. This approach is criticized by Humble, who considers that potentially creating a new silo is not the best policy to handle the DevOps principle of breaking down walls between silos [Hum12]. Skelton and Pais present some variations, called "DevOps topologies," and also DevOps anti-patterns [SP13]. They argue that, although the DevOps team silo is an anti-pattern, it is acceptable when it is temporary and focused on sharing development practices to operations staff and operations concerns to developers. Siqueira et al. report how a DevOps team with senior developers and rotating roles among members was a key decision to construct and maintain a continuous delivery process [Siq+18].

There is yet the strategy adopted by Google, called Site Reliability Engineering (SRE) [Bey+16], which involves the evolution of the operations engineer role. In addition to product teams, Google has SRE teams responsible for product reliability engineering, which includes performance, availability, monitoring, and emergency response. SRE teams have an upper limit of using 50 percent of their time on operational tasks because they must allocate at least 50 percent of their time increasing product reliability through engineering and development activities.

One should also question the impact of deployment automation on operators' roles. Chen reported that, after continuous delivery adoption, operations engineers only needed to click a button to release new versions [Che15]. One should question the role of an engineer in such a context. Moreover, if software updates are adequately broken down into small increments, and delivery automation turns deployment in a "non-event" [HM11], practices such as "development and operations teams should celebrate successful releases together" [HM11] can be perceived as contradictory.

Therefore, if operations provide support services (continuous integration, monitoring) [HM11] and product reliability is appropriately designed, product teams can easily deploy and operate their services. Thus, one can even replace operations with third-party cloud services, as done by Cukier [Cuk13], and achieve an organizational structure known as "NoOps". Some blog posts promote NoOps concepts, but the literature has not yet covered this subject. One possible definition is: "NoOps (no operations) is the concept that an IT environment can become so automated and abstracted from the underlying infrastructure that there is no need for a dedicated team to manage software in-house"[3] [Rou15]. Cross-functional teams can embrace NoOps, with a particular focus on employing cloud technology that automates much of the operational work, rather than reallocating operators or teaching developers all the operational work.

2.2 Delivery performance

Delivery performance combines three metrics: frequency of deployment, time from commit to production, and mean time to recovery [For+20]. These metrics have become

[3] Although, according to this very definition, it would be more suitable for NoOps to mean "no operators" rather than "no operations."

very popular for measuring IT-performance and DevOps adoption [Sal+21]. Moreover, delivery performance correlates to the organizational capability of achieving both commercial goals (profitability, productivity, and market share) and noncommercial goals (effectiveness, efficiency, and customer satisfaction) [FHK18]. We used this construct as an indication of how successful an organization has been in adopting continuous delivery.

Based on a survey with 27,000 responses, Forsgren *et al.* [FHK18] 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 hour to reach production, and incidents repaired in less than 1 hour. *Medium performers* deployed 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 took 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 scale:* 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 scale:* 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 scale:* 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.

During our interviews of Phase 2, we asked interviewees about these metrics in their organizations, so we could classify their contexts as being of high performance or not. Here, context relates to the primary deployable units [Sha+19] with which the practitioner worked in recent times.

Forsgren et al. initially considered also change failure rate to compose IT performance [Vel+14]. However, statistical tests considered it less significant than the other three presented metrics. Nonetheless, researchers still used this fourth metric [FHK18; Sal+21]. Therefore, although we did not employ this metric for high-performance classification, we considered it to broaden our perception of practitioners' situations. The scale for this metric is:

- *Change failure rate scale:* 0% - 1%; 1% - 15%; 15% - 30%; 30% - 50%; More than 50%.

Thus, for the interviews we conducted (described in Section 3.5), by assessing the presented metrics for services maintained by the participants, we perceived how con-

tinuously they could deliver new software versions while guaranteeing reliability. That is, how successful they have been in adopting continuous delivery, a crucial target of DevOps.

2.3 The structures of organizations

Based on several studies [SF95; Don99; Hal84], Oliveira summarizes the basic elements of organizational structures as differentiation (division of labor) and integration (coordination) [Oli12]. Such definition aligns well with how Conway's Law compares graphs abstracting system structures (subsystems' interconnections) and the structures within an organization (communication paths among groups of people) [Con68]. Figure 2.7 illustrates this concept. Therefore, "organizational structure" (or just "structure" in this thesis) can be understood as "differentiation and integration patterns".

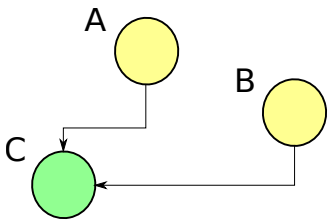

Figure 2.7: *According to Conway's Law, the represented graph may refer, for example, to* (i) *services A and B using the underlying infrastructure C, or* (ii) *development teams A and B demanding the infrastructure group C.*

Organizations and their structures have been extensively studied in the management and administration fields. The studies of organizational structures deal with the disposition of groups (e.g., departments) in a firm and the existing relations among them [Oli12]. The roots of thinking on how the division of labor affects the productive forces of work in industrial settings can be traced back to Adam Smith [Smi14]. Later, Frederick Taylor reinforced the notion of maximum division of labor as the path to attain productivity gains [Oli12; Win14]. In this way, Taylor founded a management school advocating rational administrative procedures, clear channels of authority, centralized command and control, and interchangeability of employees [YH98]. Taylorism also relates to the Weberian bureaucratic theory: managers must avoid favoritism, nepotism, and intuition [YH98]. Taylorism also led to the emergence of Fordism, which focused on the mass production of standardized goods through the assembly-line [Jes92]. This view of classical management laid the basis for profoundly segregating development and operations in the software industry.

Despite the outstanding popularity of Tayloristic approaches until the 60s, the use of top-down enforced rules and standards is effective only in stable environments (e.g., assemble lines) [YH98]. Moreover, Taylorism worked with the assumption that global optimization comes from local optimizations [YH98], which is disputed[4]. Also, some

[4] Theory of constraints alerts that local optimizations lead to global optimizations only when applied to the bottlenecks of the process [GC14].

industries felt tensions in segregating departments. During the Boeing 777 construction, for example, the formerly isolated departments of design and manufacture were reconciled in the "working together" system, which promoted the free flow of information between departments and even suppliers [Sab96]. The "working together" aptitude aligns with the DevOps movement's impetus in making development and infrastructure staffs collaborate among them [AH09].

Yeatts and Hyten claim that in uncertain or unstable environments, employees need to make unique decisions quickly, in such a way that rules and standards are of little value [YH98]. In such a context, they advocate that better performance is achieved with more informal and decentralized organizational designs. They, then, propose the adoption of self-managed work teams (SMWT), which relates to the cross-functional teams strategy adopted by Amazon [Gra06]. Such a position defies the view of maximum division of labor as the supreme efficiency enabler. Yeatts and Hyten define a self-managed work team as "a group of employees responsible for managing and performing technical tasks that result in a product or service being delivered to an internal or external customer." These teams would have the following properties:

- Having typically from 5 to 15 employees.
- Being responsible for all the technical aspects of work.
- Rotating management and technical responsibilities among themselves periodically.
- Not being short-term purpose groups.
- Providing group members with autonomy, responsibility, meaningful tasks, and the opportunity to pursue different skills (what can satisfy in different degrees diverse employees).

By examining the sectors of manufacturing, public service, and health care, Yeatts and Hyten found that SMWTs promote higher performance at less cost, and produce more innovation and creativity. The rationale is that decisions are more effective because team members taking decisions are the most knowledgeable persons about the work. However, the authors acknowledge that it is hard to prove such propositions through a quantitative analysis due to the difficulty in isolating factors. For example, some models predict self-reported performance well but not actual performance.

Besides maximum division of labor versus SMWTs, it is also possible for each employee to report to both a product manager and a functional manager. This structure is called matrix; it provides dual authority, likely to cause confusion and conflict [Gal08]. Fayol corroborates this vision, stating that duality of command would be a perpetual source of conflicts, very grave sometimes [Fay13]. Galbraith also presents a more sophisticated structure, the "front-back structure," aiming at the achievement of global scale [Gal08]. In this structure, part of the organization (the front half) focuses on customers, while the other (the back half) focuses on products, providing services to the front half.

Another alternative to Fordism and maximum division of labor is the Toyota Production System (TPS) [Ohn88]. In this system, mass production is restrained by a *just-in-time* philosophy. Quality becomes a shared responsibility among all employees, not only a specialization: the production line must be stopped to fix problems. This philosophy has

influenced the ideas of lean software development, which focus on principles derived from TPS, such as reducing waste and bottlenecks [PP06].

It is also possible to analyze an organizational structure around other dimensions beyond the division of labor and integration. Researchers have proposed several classifications for organizational structures over the years. Mintzberg, for example, proposes a structure classification around three dimensions: prime coordinating mechanism, key part of the organization, and type of decentralization [Lun12]. In this way, his proposed structures are:

Simple structure, with direct supervision, strategic apex as key part, and both vertical and horizontal centralization.

Machine bureaucracy, with standardization of work processes, technostructure as key part, and limited horizontal decentralization.

Professional bureaucracy, with standardization of skills, operating core as key part, and both vertical and horizontal decentralization.

Divisionalized form, with standardization of outputs, middle line as key part, and limited vertical decentralization.

Adhocracy, with mutual adjustment, support staff as key part, and selective decentralization.

There are also other organizational studies that are more concerned with the dynamics of organizational changes than with the understanding of structures at a certain point in time [Sen+99; KC12; MR04]. Such studies are anchored in systems thinking (or complex thinking), related to complex systems, which is a dynamic view of organizations based on biological metaphors [Mor11]. Complex systems consider the interplay between growth processes and limiting processes [Sen+99]. Senge et al., for example, identify growth processes ("because it matters," "because my colleagues take it seriously," and "because it works"), and strategies to limit these processes (e.g., "don't push so hard for growth") [Sen+99]. Researchers also represent complex systems as networks of factors that mutually affect each other through reinforcing and balancing feedback loops [Pen+18]. Moreover, complex systems encompass leverage points, in which small changes in one aspect can provoke significant system-wide changes [Pen+18]. In other words, while classical management focuses on the division of labor among working groups, complex thinking focuses on the integration of such groups ("how things connect are more important than what they are" [Sin20]).

Assuming that organizations are complex systems has implications. Sosa et al. found a strong tendency for design interfaces and team interactions to be aligned [SER04], corroborating Conway's Law. Nonetheless, in complex systems, they perceived such a match is not perfect. Another implication is that central processes of organizations must focus on change, such as Toyota's *katas*, which are work patterns or routines focusing on continuous improvement and adaptation [Rot09]. In our work, complex thinking is not central. Nonetheless, we consider organizations as heterogeneously encompassing different structures, and such structures to vary over time ("transitions"). Also, the causes and conditions we study for each structure are implied in the forces operating such

transitions.

Authors tackling complex systems have also published books to support managers in carrying organizational changes in their companies [Sen+99; KC12; MR04; Pfl14]. They, for example, list elements necessary to successful transformations (e.g., support from the CEO and reducing barriers within the organization [Sen+99]). Kotter and Cohen provide eight steps toward change: "increase urgency," "build the guiding team," "get the vision right," "communicate for buy-in," "empower action," "create short-term wins," "don't let up," and "make change stick." They also reveal core patterns associated with successful change: see, feel[5], and change. Manns and Rising defend that change is best introduced bottom-up with support at appropriate points from management, both local and at a higher level [MR04]. Such proposition resonates with our empirical findings (see Section 5.4). They also see change and innovation as a process, rather than an event, composed of stages: knowledge, persuasion, decision, implementation, and confirmation.

A significant part of the refereed business books [Sen+99; KC12; MR04] sees actions over people's culture, feelings, and emotions as more important than actions over structures. Senge et al. state that "it is not enough to change strategies, *structures*, and systems, unless the thinking that produced those strategies, structures, and systems also changes" [Sen+99] (at least one of our interviewees thought in this way too – see Section 4.3.1). Kotter and Cohen reported: "the central issue is never strategy, *structure*, culture, or systems (although these elements can be very important). The core of the matter is always about changing the behavior of people" [KC12], which is contradictory since they define culture as a group of norms of *behavior* and shared values. Manns and Rising relate culture and people to the speed up or slow down of innovation-decision processes [MR04].

Nonetheless, this centrality on culture is disputable. Consider Pflaeging' view [Pfl14]:

> *"Culture is not a success factor, but an effect of success or failure. (...) That is why it also cannot be influenced directly. (...) Culture is observable but not controllable. (...) A company cannot choose its culture, it has exactly the culture it deserves. Culture is something like the restless memory of an organization. (...) It endows something like a common "style" which all can rely on. (...) Culture, in this sense, is autonomous. Culture neither is a barrier to change, nor does it encourage change. It can provide hints to what an organization must learn. To change culture is impossible, but culture observation is valuable. (...) Culture allows us to indirectly observe the quality of change efforts: change trickles through into the culture."*

Such a view is somewhat corroborated by other stands, including historical materialism, for which the mode of production (and we can include its structures) of a society conditions the superstructure of this society (which includes culture) [Mar20]. Forsgren et al., for example, find out that practices of lean product development and lean management promote a performance-oriented culture [FHK18]. Therefore, we consider it relevant to study the forms of production of a society, regardless of their interplay with culture and psychological factors (that do exist and are also relevant).

Pflaeging also has a proposal for structuring the production in complex organizations

[5] Other theorists also question the pure rational model of decision making (i.e., the *homo economicus*) [MS93].

toward high-performance [Pfl14]. He advocates a drastic change in the management approach to a non-management approach, organizing the company in a value-creating network of self-managed teams that interact as peers. He also proposes broad participation of employees in decision-making, roles rather than positions for employees, emergent leadership, and assessment of team performance rather than individual performance[6]. Davis and Daniels call this stand holacracy, in which decisions are made by self-organizing teams, not by a traditional management hierarchy [DD16]. Davis and Daniels also alert to the risks of holacracy, in which career development is less clear, and people without the necessary management skills end up unofficially filling power vacuums.

In this section, the reader was able to perceive that the general literature on organizations and their structures is vast and rich. However, one could also note that studies in this literature are not consensual about many dimensions. This legitimates studies in specific areas, such as in software production. Thus, in the next section, we explore the specific literature in the context of software production. Finally, we hope the issues discussed in this section have provided the reader with a broader perspective to interpret the propositions of this thesis.

2.4 Related work

Organizations and their structures have also been studied 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 [SB98]. 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 [HR06]. Nagappan et al. provide eight metrics to quantify organizational complexity, seeking to associate such metrics with software quality [NMB08]. However, these earlier works focus on development activities, without considering infrastructure or operations concerns.

In Section 2.1.3, we already discussed the quandary brought by the DevOps movement regarding how to structure the working groups of development and infrastructure. Researchers and practitioners have proposed taxonomies [Ral19] for the interactions between development and infrastructure professionals [SP13; SP19; NSP16; MBK18; Sha+17; Lop+21; MB20; Kim+16]. 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. [Sha+17] 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., we created a taxonomy of structures to organize development and infrastructure professionals, which mostly coincides with that of Shahin

[6] While advocating cross-functional teams, Donnellon also favors team evaluation over individual evaluation, toward a more collaborative way of working [Don93].

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 [SP19; BSK20]. Thus, we identified the following structures:

- **Segregated departments**, with highly bureaucratized cooperation between development and infrastructure groups.
- **Collaborating departments**, focusing on facilitated communication and collaboration between development and infrastructure groups.
- **API-mediated departments**, in which the infrastructure team (the **platform team**) provides highly-automated infrastructure services to assist developers.
- **Single departments**, in which teams take responsibility for both software development and infrastructure management.

We provide a complete description of each structure in Section 4.3, presenting them as a grounded theory [GS99] in the taxonomy form (i.e., as a classification system) [Ral19]. We observed that professionals may be organized differently for different deployable units [Sha+19] 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 [For+20].

Lopez-Fernandez et al. developed a taxonomy concurrently with ours [Lop+21]. 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. Lopez-Fernandez et al. [Lop+21] acknowledge the common points among the works of Shahin et al. [Sha+17], ours, and theirs, highlighting different insights brought by them and stressing how these models can be appreciated in conjunction, providing data and methods triangulation. More on the differences between our taxonomy and those of other researchers are in Section 4.5.

However, the above-related works present taxonomies to *describe* structures. They do not *explain* why different companies adopt different organizational structures, which is also a research goal of us. Nonetheless, Erich et al. [EAD17] 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 targets 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, as we do in Section 5.4.

Chapter 3

Research design

This thesis presents a theory based on data collected from semi-structured interviews conducted in real-world organizations. By analyzing the data extracted from the interviews, we built a theory integrating concepts organized around a taxonomy. In this chapter, we: disclose our understanding of theory and taxonomy; explain our base methodology – Grounded Theory (GT) [GS99]; and present the steps of our research design. We also make explicit how we conducted the literature review throughout our research process.

3.1 Theories and taxonomies

We present our theory by structuring its concepts around a taxonomy. Broadly speaking, a theory is a system of ideas for explaining a phenomenon [Ral19], or knowledge accumulated in a systematic manner [Gre06]. 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[1] [Ral19]. 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 [Ral19]. If a taxonomy provides an 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 [Ral19]. While applying "taxonomy" to denote a theoretical formulation, we employ "classification" in a broader sense, including classification proposals without theoretical formulation.

We can categorize our study as a field study [SF18], a category of knowledge-seeking studies, in opposition to solution-seeking studies [SF18]. 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, i.e., a guide for reasoning, not a straightforward guide for action. This aligns with Ralph, who says a theory should explain, not prescribe [Ral19].

[1] The scientific literature as a whole does not employ the concepts of taxonomy, typology, and classification in a unified and consistent manner [DG94; McK82]. In our research, we follow Ralph's considerations [Ral19], provided in the software engineering context.

One could see a knowledge-seeking theory as of little practical relevance. However, Gregor defies such a view. He states that accumulated knowledge in theories enlightens professional practice [Gre06] – "nothing is so practical as a good theory" [Lew45]. Gregor still ensures that all types of theories can have social or political implications: "the mere act of classifying people or things into groups and giving them names (e.g., black versus white) can give rise to stereotypical thinking and have political and social consequences." We provide more practical implications of our theory in the conclusion of this thesis (Chapter 7).

Gregor also provides a taxonomy of theory types in information systems research [Gre06]. The types are: I - analysis (description), II - explanation, III - prediction, IV - explanation and prediction, and V - design and action. In this framework, we fit our theory in type II, a *theory to explain* a specific phenomenon. We also fit our effort of Phase 2 as initially building only a *theory do describe* (type I) some existing patterns in the world.

Still, Glaser and Strauss classify theories into substantive and formal theories [GS99]. Substantive theories are grounded in research on one particular substantive area (e.g., work, juvenile delinquency, medical education, mental health), and they might apply to that specific area only. A formal theory is more abstract than a substantive one, it is developed for a formal, or conceptual, area of sociological inquiry (e.g., stigma, deviant behavior, formal organization, socialization, status congruence, authority and power, reward systems, or social mobility). An example of a substantive theory is: "How doctors and nurses give medical attention to dying patients according to the patient's social value." A corresponding formal theory could be "How professional services are distributed according to the social value of clients." Our theory is a substantive one, i.e., we constraint our theoretical formulations to the context in which the involved researchers are experts, the software field. In particular, Glaser and Strauss claim that substantive theories can have important general implications and relevance, laying the foundations for developing subsequent formal theories.

For our research context, building a formal theory would mean fully integrating our findings to more general theories on the study of organizations, usually published in the fields of administration and management. Although we do not covet such an accomplishment, we do not ignore these other works. In Section 2.3, we provided an overview of this general literature about structures in organizations. In this way, the reader can, at least, appreciate our work considering a broader picture.

3.2 Grounded Theory

To build a theory, one must initially embrace some scientific approach geared to theory production. The rise of modern science was sustained by the mathematical and experimental pillars, being experiments strongly conjoined with statistical methods [Rad09]. In the mid-1970s, even sociological sciences, in which strict experimental control is often unfeasible, focused on experimental work [Rad09]. On the other hand, by the time, qualitative research was being seen as lacking scientific rigor, and emphasis was on theory

verification [Hod21]. However, especially in social sciences, there was a gap: how to produce theories to be tested by the established scientific methods?

Grounded Theory (GT) is a methodology originated in social sciences to support researchers to build theories based typically on qualitative data in a disciplined manner. With its systematic and rigorous procedures, the introduction of GT re-established the value of qualitative research in sociology, arguing for theory development as a fundamental research objective [Hod21]. The term "grounded" means a grounded theory is backed (grounded) by data. Generating a theory from data means that most hypotheses and concepts are systematically worked out in relation to the data during the course of the research – generating a theory involves a research process [GS99].

Software engineering encompasses several social and human concerns, as it is, for example, the issue of organizational structures in corporations. In this way, the use of GT in the software engineering field has become commonplace [SRF16; Lop+21; MB20; Jov+17; WNA15; HN17; San+16; FJT16; LPB19]. A grounded theory must fit the data, have relevance to the field, and be modifiable [GS99]. Its primary advantage is to encourage deep immersion in the data, which may protect researchers from missing instances or oversimplifying and over-rationalizing processes [Ral19]. GT is also adequate for our purposes since it is well-suited for questions like "what is going on here?" [SRF16]. In our case, we want to know what is going on in software-producing organizations.

Since there are multiple GT variants, it is important to state which one to adopt. In our work, we base our approach on the seminal book *The Discovery of Grounded Theory* from Glaser and Strauss [GS99], which describes what is known as the "classical Grounded Theory" [SRF16]. Historically, this is the first GT variant, and it is epistemologically[2] more influenced by the views of objectivism[3] and positivism[4]; in this case, theory is discovered from data [SRF16]. While a theory itself must emerge from data, classical GT do not disallow pre-established research questions[5]. There are other GT variants, including ones based on interpretivism[6] and constructivism[7], in which knowledge and truth are created, not discovered [Sch98]. However, debates among scholars make no variant universally endorsed [SRF16; Cha08]. More recentlty, Hoda proposed a new GT variant, the Socio-Technical Grounded Theory (STGT), tailored for use in software engineering and information systems research [Hod21]. However, STGT was not available when we elaborated our research design.

---

[2] Epistemology refers to what can be treated as knowledge and how that knowledge is gained, in a research context [Hod21].

[3] For objectivism, there is a single, correct description of reality [SRF16].

[4] At least in our context, positivism refers to a definite, assured, and certain nature of reality that can be studied through direct observations, and it excludes traditional metaphysics and theology [Hod21].

[5] Examples of questions guiding sociological inquiries in the GT context: "Does the incest taboo exist in all societies?", "Are almost all nurses women?" [GS99].

[6] Interpretivism assumes no universal truth or reality exists, systems exhibit emergent behaviors not reducible to their component parts, and that typical quantitative methods are insufficient for understanding social phenomena [SRF16].

[7] For constructivism, reality is socially constructed and researchers' and participants' presence and interactions construct the reality that is studied [Hod21].

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 [GS99], sufficing that the researcher annotates concepts and adheres to the following rules: *(i)* when noting a concept, compare its occurrence with 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.

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[8] and our previous works on DevOps and software engineering [Siq+18; CK18; Wen+20]. Moreover, we sharpened our sensitivity by conducting our survey on the DevOps literature [Lei+19]. This acquired theoretical sensitivity explains our capacity to pose the research questions of this doctoral research.

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.

Ideally, the researcher should continue the analysis until *theoretical saturation* is achieved, which means that new data no longer meaningly impact the theory. Nonetheless, 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 [GS99].

3.3 Overview of the steps in our research process

To better structure the presentation of our research steps, here we borrow the nomenclature defined by Hoda for the Socio-Technical Grounded Theory (STGT) [Hod21]:

Basic stage: encompasses Basic Data Collection and Analysis (DCA), which involves lean literature review, study preparation and piloting, and iterations of basic data collection and analysis. Initial theoretical concepts start to emerge.

Advanced stage: encompasses theory development, involving targeted literature review if necessary, and selecting the emergent or the structured mode of theory development.

[8] The doctoral candidate is also a professional software developer.

Emergent mode: enables the emergence of theory through the iterative targeted data collection and analysis, ending with theoretical saturation and resulting in a mature and integrated theory.

Structured mode: enables the structured development of theory through structured data collection and analysis, ending with theoretical saturation and resulting in a mature and structured theory.

Lean literature review (LLR): a lightweight and high-level review performed early in the study, during the basic stage, to identify research gaps and motivate the need for a study.

Targeted literature review (TLR): an in-depth review of literature targeting relevance to the emerging concepts, performed periodically during the advanced stage.

With this frame, Figure 3.1 presents the steps of our research. In Phase 1, we conducted an initial literature review to identify research gaps. This was a "lean review" regarding the goal of this thesis (understanding the organization of development and infrastructure professionals in the software industry), but a broad and solid review regarding the topic of the review itself, which was "DevOps". We presented the results of this review, published in a journal (Paper 1), in Section 2.1. The primary output of Phase 1 was the research questions of this thesis.

In Phase 2, the basic stage continued with brainstorming sessions with DevOps specialists. We then fine-tuned our research questions and gathered input to build our research protocol. This stage also yielded some relevant concepts, such as **platform teams**. We explain these brainstorming sessions in Section 3.4.

With a well-defined protocol, we then initiated the advanced stage, geared to theory generation. We iteratively applied GT recommended analysis selection and procedures, especially open coding, to produce concepts in our theoretical framework. By using such concepts and comparing interview data (using the comparison sheet), we integrated such concepts into a taxonomy. We then gathered some feedback from the study participants. In this way, we produced our taxonomy, published in a journal (Paper 4). We expound how we planned to conduct interviews and to analyze data, including the review process, for Phase 2 in Sections 3.5, 3.6, and 3.7. Chapter 4 unfolds the results of the planned actions: the process of participants selection, the achievement of saturation, the produced taxonomy, and the gathered feedback.

In Phase 3, we continued our theory development, now in structured mode and using a theoretical coding template (6C) recommended in the context of GT. We also performed a structured analysis of resonance, labeling interview excerpts as showing **support** or **confusion** about our taxonomy. After another round of participants' feedback, we finished our theory formulation (refined based on resonance analysis and structured in the 6C template), which we submitted to publication in a journal (Paper 6). We expound how we planned to conduct interviews and to analyze data, including the review process, for Phase 3 in Sections 3.8, 3.9, 3.10, and 3.11. Chapter 5 unfolds the results of the planned actions: the process of participants selection, the achievement of saturation, the consolidated theory, and the gathered feedback.

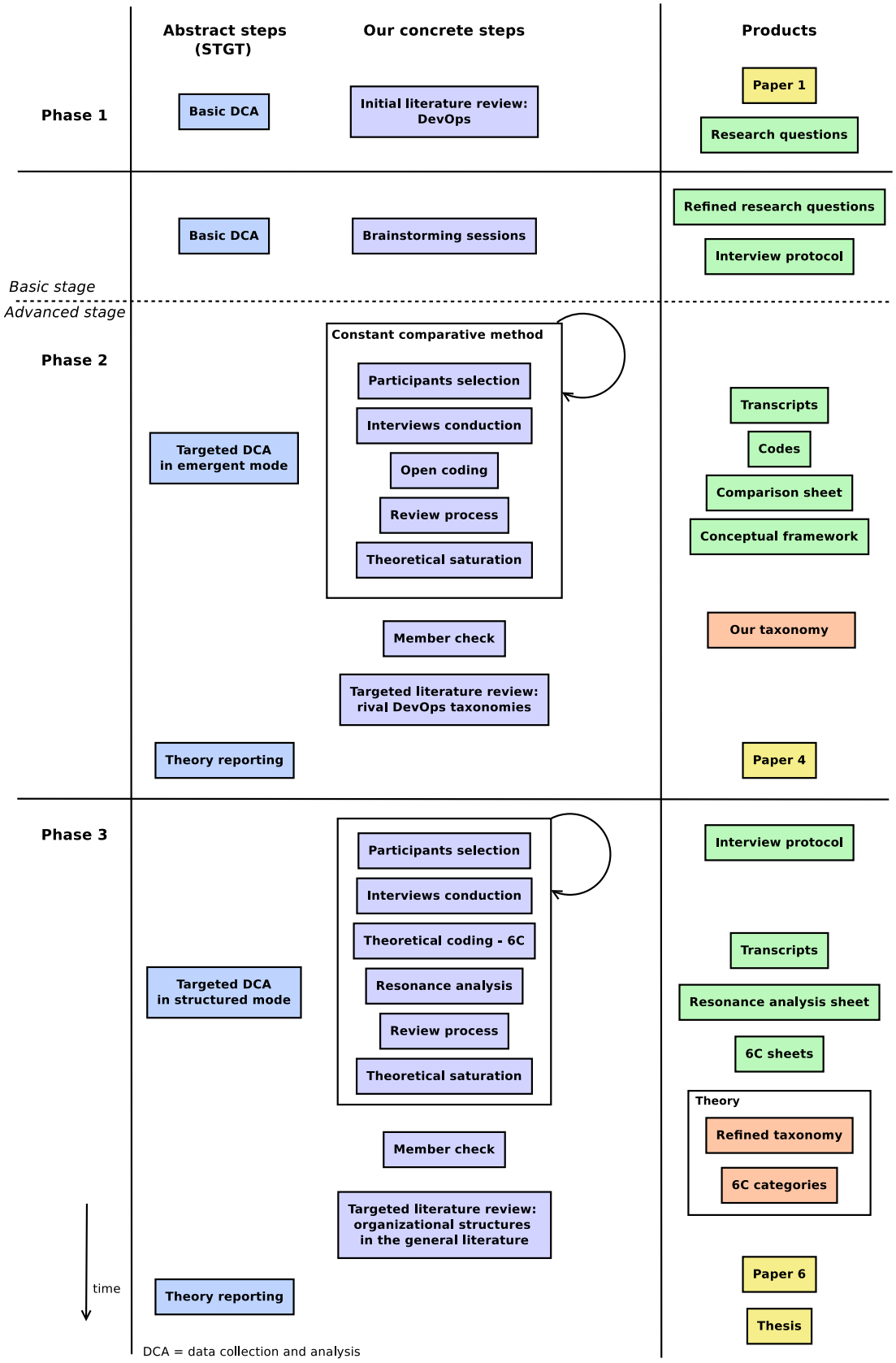

Figure 3.1: *Steps and products of our research*

Finally, in Section 3.12, we clarify the timing and role of our literature reviews.

3.4 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 (such as a senior consultant and a serial entrepreneur we know).

The base script for these sessions aimed to elicit feedback on our research questions and spark discussion about concerns raised by our survey on the DevOps literature [Lei+19]. The conversations were essential for us to fine-tune our research questions and research approach. These sessions also helped us to target better 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 presented in this text and the interview script, which we finished just after the brainstorming sessions.

We did not apply the analysis procedures detailed in Section 3.6 for these preliminary conversations, considering they were not intended to provide answers for 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.

3.5 Conducting the interviews of Phase 2

Since our initial goal was 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 [Ada10], 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 (Appendix A) to guide the process based on our previous experience with interviews [CK18], on other relevant works [HN17; Ada10], and on the guidelines offered by a website for journalists, called ijnet[9].

The interview protocol contained the questions that guided the interviews, which mainly were derived from the brainstorming sessions and our survey of the DevOps litera-

[9] https://ijnet.org/en

ture [Lei+19], and hence also grounded in data[10]. The themes addressed by the interview questions of Phase 2 included: (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 2.2); (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 was 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.

3.6 Analyzing the interviews of Phase 2

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: *transcripts* and *codes*. We also created two global artifacts: a *comparison sheet* and a *conceptual framework*. Finally, by analyzing, comparing, and using all these artifacts, we elaborated our *taxonomy*.

Transcripts. We listened to each audio record and transcribed it in the language of the interview (Portuguese or English). We did not transcribe the full interview. Instead, we excluded minor details and meaningless noise [GA08]. 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."*

Example of part of the conversation not transcribed:

> *"Actually if you haven't read The Phoenix Project [KBS18], read The Goal [GC14] first. You'll see what I mean… because you ill get the foundations first and then the way it is being implemented in the third millennium."*

This was helpful advice that, indeed, we followed. Nevertheless, in this case, it was enough to take note of the suggested material.

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

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

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

Even for interviews in Portuguese, we produced the codes in English (at least starting from the seventh interview). Our supplementary material presents more examples of the coding we performed[11].

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 concepts and their relations form the unified *conceptual framework*, which comprises the shared understanding among researchers about the analyzed data [MH13]. Our conceptual framework is not a representation of our theory or taxonomy, 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. In such visual representation, as illustrated by Figure 3.2, rectangles represent *concepts* abstracted from data, while rounded boxes represent *properties* of these concepts (i.e., an attribute or a characteristic of a concept). Figure 3.3 provides as example a fragment of our conceptual framework. We provide all the versions of our conceptual framework in the supplementary material (see Section 6.2).

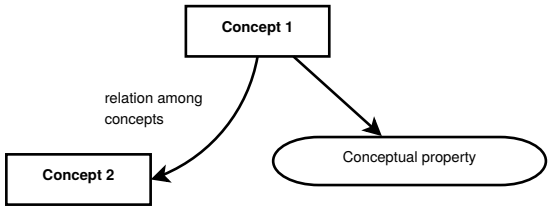

Figure 3.2: *Conceptual framework nomenclature*

As we evolved the conceptual framework and filled our comparison sheet, we developed our taxonomy by classifying each interview by its organizational structure and

[11] The supplementary material to this thesis provides extra material reporting our thorough chain of evidence and enabling the replication of our procedures. See Section 6.2.

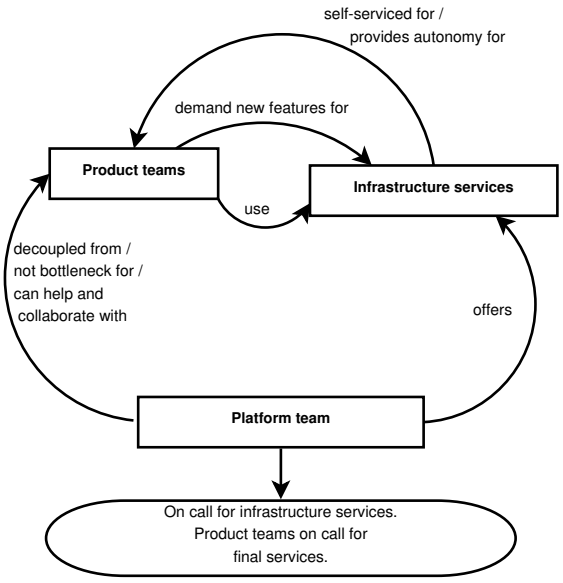

Figure 3.3: *A fragment of our conceptual framework*

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 taxonomy. For example, after the emergence of a **supplementary property**, we checked whether we could classify previous interviews with the new property. We present the taxonomy emerged from this process in Section 4.3.

3.7 Review process of Phase 2

After some interviews, the doctoral candidate developed the first version of the coding lists, the comparison sheet, the conceptual framework, and the taxonomy. Based on the transcriptions, other two researchers thoroughly reviewed these artifacts, triggering discussions that affected their evolution. When making a decision that could impact our taxonomy, we involved a fourth researcher. 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. We went through this internal review process to reduce the bias of a single researcher performing analysis with preconceived ideas.

3.8 Conducting the interviews of Phase 3

In the conducted semi-structured interviews of Phase 3, after presenting our taxonomy to the interviewee, we approached: which organizational structure was adopted in the context of the interviewee according to their opinion; why that structure was chosen over the others; whether a different structure would be more suitable for the interviewee's

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 [Ada10]. We provide our complete script with the rationale of each question in Appendix B.

3.9 Resonance analysis

The interviews of Phase 2 provided us with enough data to build the first versions of our taxonomy of organizational structures. In such conversations, we employed second-level questions [Yin09] to avoid exposing the emerging structures to interviewees. After this, we performed a brief and limited member check [Gub81] 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 [Ral19]) and serves as a grounded theory (being applicable by practitioners [GS99]). To answer **RQ2** and **RQ2.1**, we set up a favorable context to apply and refine our taxonomy – we had to use its concepts during interviews of Phase 3.

We refer to the taxonomy refinement process as *resonance analysis*, where "resonance" alludes to the degree to which findings are understandable to participants [Ral19]. 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 [Sir+11; Pin86; And83]. Steinmacher et al., for example, gathered qualitative data to refine their grounded theory with additions, deletions, and reorganizations in their model [Ste+16]. 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 justified 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**) [HS05]. We present the results of the resonance analysis in Section 5.3.

3.10 6C analysis

Glaser proposed 18 theoretical coding families to guide researchers in labeling data at the conceptual level [Gla78]. 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" [Sal15]. Some GT studies on software engineering use this practice [HNM11; vv13; Jov+17].

Glaser 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 [Gla05]. 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" [Her09]. Indeed, in this doctoral research, we present not just a "theory for analyzing," but a "theory for explaining" [Gre06].

As it happened with Hoda et al. [HNM11], 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* may 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 (see Section 6.2).

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 [Her09]. 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[12].

We did not analyze context and covariance interview-by-interview. The context of our interviews corresponds to our sample (see Section 5.1). 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**, 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 5.4.

3.11 Review process of Phase 3

The doctoral candidate conducted the initial analysis of each interview. Periodically, other two researchers joined for review sessions in which we held discussions until reaching a consensus. While the doctoral candidate is also a software developer, it is relevant to note that one of the review researchers 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 of Phase 2, aiming to soften possible biases given the doctoral candidate's metier. We held 14 review sessions, from January to October 2021, with most of them taking around two hours.

3.12 Our approach to the related literature

A key component of Grounded Theory is limiting the exposure to the literature at the beginning of the research [SRF16]. This advice diverges from recommendations of other methodologies, such as Case Study [Yin09]. The goal is to prevent researchers from testing existing theories or thinking in terms of established concepts [SRF16]. As put by Glaser and Strauss: "covering all the literature before commencing research increases the probability of brutally destroying one's potentialities as a theorist" [GS99].

Nonetheless, it is not feasible to acquire theoretical sensitivity and formulate opportune research questions with no literature exposure at all. Therefore, we had to manage literature

[12] One of the meanings of "contingency" in the Cambridge Dictionary is "an arrangement for dealing with something that might possibly happen or cause problems in the future" (https://dictionary.cambridge.org/dictionary/english/contingency).

exposure in this doctoral research carefully. We started, in Phase 1, with a literature review on DevOps (Chapter 2) in which we studied a few papers unfolding our research questions concerning organizational structures. After reading this bare minimum to formulate our research questions, we did not look for more readings on organizational structures. The main idea was to avoid making our open coding process biased with concepts elaborated in other works. Such care makes the findings in our research much more significant, such when we discovered the concept of **platform teams**. However, with the results of our DevOps survey, we already had some relevant concepts at hand. As much as possible, we tried not to consider these concepts during our open coding process. An example of a promising concept found in our survey was the "DevOps team," which one could expect to emerge as a structure of our taxonomy. The fact that this did not happen is evidence that we, fortunately, did not drive our analysis by the initial concepts yielded by our DevOps survey.

At the end of Phase 2, we started to look for similar works (i.e., other works proposing taxonomies for organizing development and infrastructure professionals in software-producing companies). We then elaborated a detailed comparison between our taxonomy and these other studies, which we present in Section 4.5.

While conducting Phase 3, we continued to track similar works that could appear. Nevertheless, we also broadened the scope of our target literature review at this stage. We started to look for works discussing more generally on organizational structures, not only in the software production context. Such works, the "general literature," are usually published in the management and administration fields. The objective of these readings was to provide perspective on our substantive theory. We did not aim to integrate such general theories with ours. Such integration would lead us to the work of formulating a formal theory [GS99], which is out of the scope of this doctoral research. However, ignoring this research landscape would not be suitable. Therefore, we present this general literature in Section 2.3.

Chapter 4

A taxonomy for describing organizational structures

In this chapter, we present the outputs of our research Phase 2. First, we explain our approach to the participant selection and the resulting sample (Section 4.1). Then, we explain our saturation criteria to stop selecting new participants (Section 4.2). After that, and most importantly, we present the first versions of our taxonomy for organizing development and infrastructure professionals (Section 4.3). We also present the result of our member check process (Section 4.4) and the comparison between our taxonomy and other similar works (Section 4.5).

4.1 Sample

We sent out 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 has implemented 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 [Str19], we anonymized all the interviewees and their organizations. 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[1], five of which took place at the interviewee's company, and 28 were held online. The sessions took from 24 to 107 minutes (median of 48 minutes).

We employed several strategies to foster diversity and enhance comparison possibilities in our sample, as recommended by GT guidelines [SRF16]. We aimed to include a broad range of organization and interviewee profiles, as perceivable in Table 4.1. For instance,

[1] These in-person interviews were before the COVID-19 pandemic.

we selected organizations of different sizes[2]: 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.

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: PhD	

Table 4.1: *Number and description of participants and organizations*

Table 4.1 describes the participants, presenting only an aggregated profile of participants to cope with anonymization [Str19; San+16]. 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 hiring the consultancy companies (and not to the consultants' employers). The interviewees worked in the following business domains: IoT, finance, defense, public administration, justice, real estate, maps, education, Internet, big data, research, insurance, cloud, games, e-commerce, telecommunication, fashion, international relations, mobility, office automation, software consulting, inventory management, vehicular automation, team management, and support to software development. Five of the interviewed companies were unicorns (startups with a valuation over U$1 billion), and two of them were tech giants (among the top 4 IT companies in the world).

We also selected participants with theoretical purposes in mind, thus applying theoretical sampling [GS99]. We interviewed participants who work in scenarios where it

[2] We employed 200 and 1,000 as size thresholds because they are used in information publicly available on LinkedIn.

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 (see Section 6.2) and in this thesis. The excerpts are formatted in italics and within quotes. Excerpts and other accounts refer to interviews using tokens in the format "I*N*". 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 [GS99].

4.2 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, by counting the number of concepts, conceptual properties, and links. Consider Figure 4.1-(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.

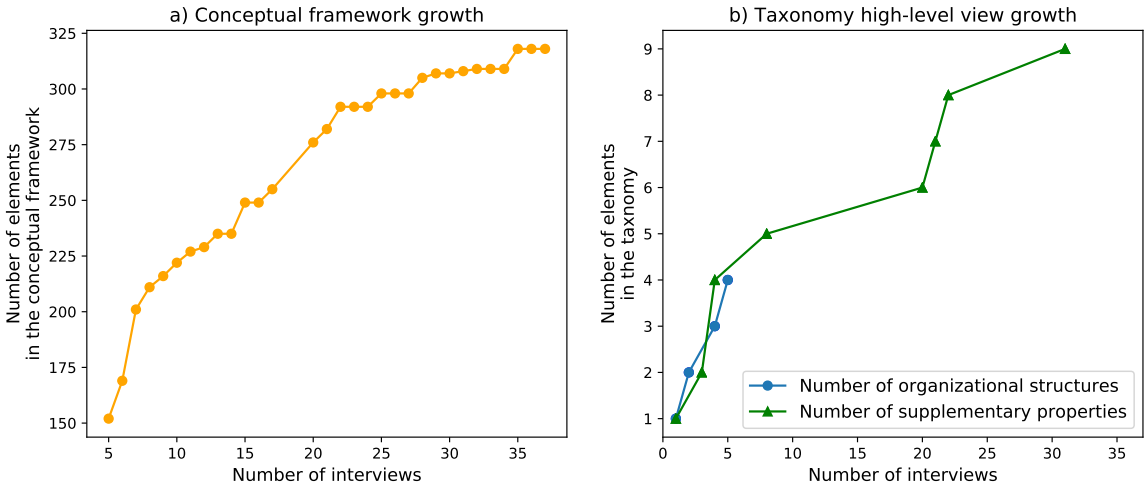

Figure 4.1: *Evidence of theoretical saturation*

In addition to measuring the size of the conceptual framework, an intermediary artifact, we also measured the growth of the taxonomy itself. Figure 4.1-(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 a lot less to shaping our theory.

By conjoining these two observations of Figure 4.1, we claim to have reached enough of saturation for our purposes.

4.3 The taxonomy of organizational structures

In this section, we answer our research questions **RQ1**, **RQ1.1**, and **RQ1.2** 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 between development and infrastructure may vary for each deployable unit [Sha+19], 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 a transition from one structure to another.

Figure 4.2 presents the high-level view of the first version of our taxonomy. We evolved this first version until a fourth version (Figure 4.3), which was published [Lei+21] (we clarify such an evolution in Section 5.3). This diagram encompasses 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 4.2 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 (see Section 6.2) indicates how much of our findings are backed by the 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 [Yin09; Ral19; Gub81]. Figure 4.4 features a word cloud built from the chain of evidence, thus providing an overview of the topics discussed during the interviews.

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

Interview	*Organizational structure*	*Supplementary properties*	*Delivery performance*	*Organization size*
I1	Cross-functional	with dedicated infra professionals	High	[200, 1000]
I2	Classical DevOps		High	[200, 1000]
I3	Cross-functional	without infra background	Not-high	< 200
I4	Platform team	cloud façade with enabler team	High	> 1,000
I5	Siloed departments		Not-high	> 1,000
I6	Classical DevOps		Not-high	[200, 1000]
I7	Siloed departments *to* Classical DevOps		Not-high	[200, 1000]
I8	Siloed departments *to* Platform team	with a customized private platform	Not-high	> 1,000
I9	Platform team	cloud façade with enabler team	High	[200, 1000]
I10	Siloed departments		Not-high	< 200
I11	Classical DevOps	with enabler team	Not-high	[200, 1000]
I12	Platform team	with a customized private platform with enabler team	High	[200, 1000]
I13	Siloed departments		Not-high	> 1,000
I14	Classical DevOps *to* Platform team	cloud façade	Not-high	[200, 1000]
I15	Siloed departments *to* Classical DevOps		Not-high	[200, 1000]
I16	Cross-functional *to* platform team	with dedicated infra professionals with a customized private platform with enabler team	Not-high	> 1,000
I17	Classical DevOps	with enabler team	High	[200, 1000]
I18	Classical DevOps	with enabler team	Not-high	> 1,000
I19	Siloed departments		Not-high	< 200
I20	Siloed departments *to* Platform team	with an in-house open source platform	High	> 1,000
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	> 1,000
I23	Siloed departments		Not-high	< 200
I24	Siloed departments *to* cross-functional	with dedicated infra professionals	High	[200, 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	> 1,000
I27	Cross-functional	with dedicated infra professionals	Not-high	< 200
I28	Cross-functional	with dedicated infra professionals	Not-high	[200, 1000]
I29	Classical DevOps		High	< 200
I30	Siloed departments	with enabler team with a platform with an in-house open source platform	Not-high	> 1,000
I31	Classical DevOps	infra as development collaborator with enabler team	Not-high	> 1,000
I32	Cross-funcional	without infra background	Not-high	< 200
I33	Platform team	cloud façade	High	> 1,000
I34	Classical DevOps		Not-high	< 200
I35	Cross-functional	with dedicated infra professionals	Not-high	< 200
I36	Classical DevOps		Not-high	> 1,000
I37	Siloed departments		Not-high	> 1,000

Table 4.2: *Interview's classification. In the second column, "to" indicates transitioning from one structure to another.*

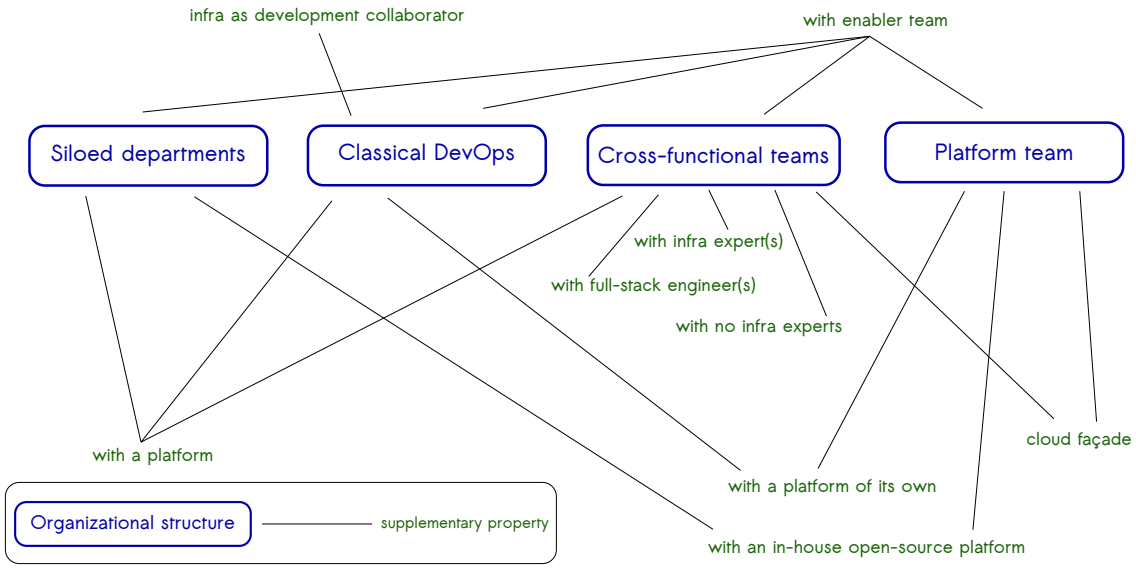

Figure 4.2: *High-level view of the first version of our taxonomy: the discovered organizational structures and their supplementary properties*

4.3.1 Siloed departments

With **siloed departments**, developers and the infrastructure staff are segregated from each other, with little direct communication among them. The core problem of this structure, as vividly portrayed in the novel "The Phoenix Project" [KBS18] and witnessed by interviewee I19, are the frictions between silos since developers want to deliver as much as possible, whereas operations target stability, blocking deliveries. The DevOps movement was born in 2008 [Deb08] to handle such problems. We found seven organizations adhering to this structure, and six others transitioning out of this structure.

Core properties

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, a problematic pattern exposed by Goldratt and Cox over 30 years ago [GC14]. Participant I26 told us about conflicts involving many departments: *"there is a big war there... the security, governance, and audit groups must still be convinced that the tool [Docker / Kubernetes] is good. All the world uses it... but they do not understand... I get sick with this."*

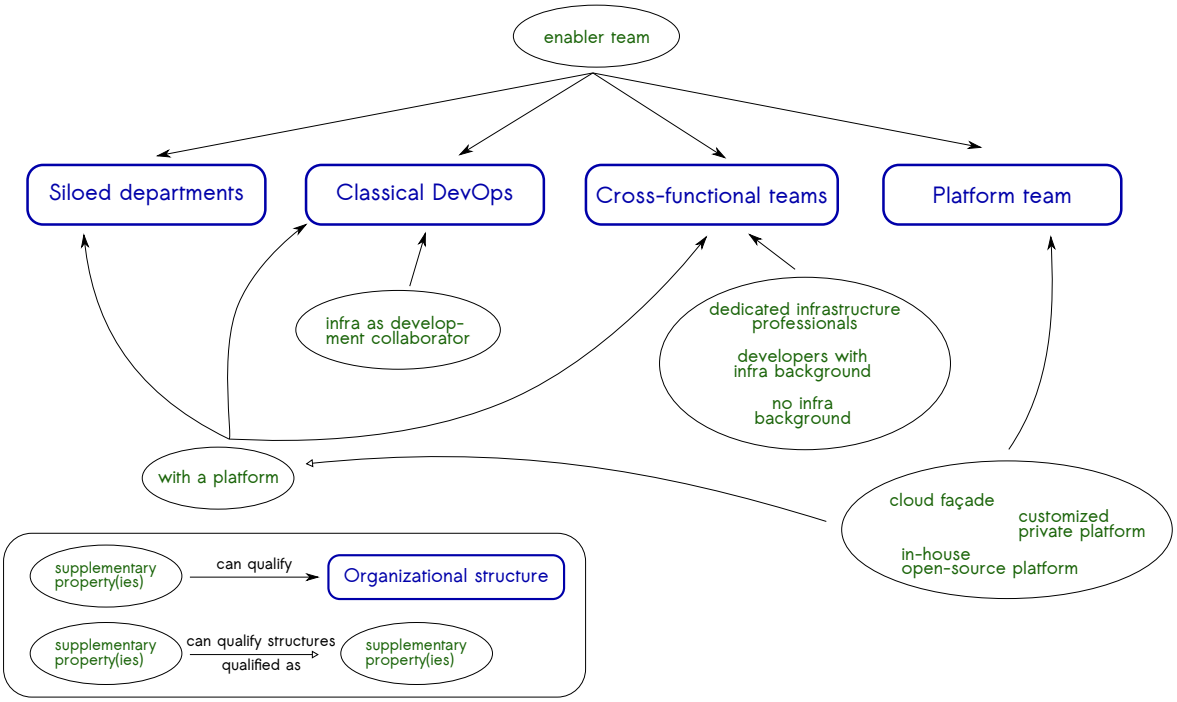

Figure 4.3: *High-level view of the fourth version of our taxonomy, published in the IST journal*

Figure 4.4: *Word cloud built from the chain of evidence of Phase 2*

→ **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 between 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, thus fulfilling Humble's forethought

about DevOps teams [Hum12]. Moreover, developers "infiltrated" infrastructure teams to skip enterprise workflows. As I15 mentioned: *"When this group was established for working with DevOps, the easy part was the outside guy [external consultant], who knew Azure and all the tools. This was not the problem, the problem was cultural, because all the hierarchies, powers, and necessities were maintained."* Interviewee I30 claimed that *"it's a matter of silos of power... I have power, I'm important, I can deny you something... so you must have a network, be friend of people, friend of the [ruling] party... to the friends everything, to the enemies: the death."*

→ **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 4.3 shows that only two of 13 **siloed organizations** presented high delivery performance, and these two were already transitioning to other structures. This finding resonates with interviewee I5's thought *"Will developers have an agile mindset? If not, it is worthless."*

However, we observed cases in which low delivery performance was not a problem. In I13, applications are short-lived research experiments, not requiring frequent updates. In I10, regular releases of content (new phases of a game) usually do not require code changes since *i)* designers produce content with well-established support tools and *ii)* correction patches are hardly necessary due to comprehensive testing. Network isolation policies may also hinder frequent deployment (B1, I7).

→ We observed a **lack of proper test automation** in many siloed 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, I37) and label a peer as "incompetent" because he is *"taking too much time with tests"* (I23). We notice that some of the observed scenarios were more challenging for test automation, such as games.

Key characteristics of siloed departments: limited collaboration among departments and barriers for continuous deployment.

4.3.2 Classical DevOps

The **classical DevOps** structure focuses on collaboration between developers and the infrastructure team. This mindset does not imply the absence of conflicts among these groups, but *"what matters is the quality of conflicts and how to deal with them"* (I34). We named this structure "Classical DevOps" because we understand that a collaborative culture is the core DevOps concern [LPB19; DD16], concurring to our DevOps definition: "*a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability*" [Lei+19]. We classified ten organizations into this structure. We also observed three organizations transitioning to this structure and three transitioning out of this structure.

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

Table 4.3: *Organizational structures and delivery performance observed in our interviews*

Core properties

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

→ 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 between developers and the infrastructure team (I2, I6, I17, I22, I31, I36), even before the product deployment in some cases. Participant I6 mentioned that developers more concerned with architectural and NFR issues, the "guardians," have close contact with the infrastructure team. 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). In short, development and infrastructure teams are *"partner teams that are helping each other"* (I36).

→ Roles remain well-defined, and despite the collaboration on some activities, there are usually **no conflicts over who is responsible for each task**[3].

[3] We later removed this **core property** in Phase 3 (see Section 5.3).

→ Developers feel relieved when they can rely on the infrastructure team since *"in the product team you don't have time to worry about so many things"* (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 other words, *"operations people are the first ones to get f***ed"* (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** (I17). 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). In I22, usually, only development leaders are called after-hours.

→ Some observed organizations **rebranded their infrastructure teams as DevOps or SRE** [Bey+16] teams (I2, I6, I14, I17). Although there is some debate about the difference between DevOps/SRE departments and traditional operations departments [Vel+14], in our observations, these DevOps/SRE teams are just infrastructure teams, but more inclined to adopt automation and to cooperate with developers when compared to infrastructure teams in the **siloed structure**. This need for embracing automation imposes some pressure on some infrastructure professionals to learn new skills, especially in large organizations (I36). Participant I29, a consultant, declared that *"many infrastructure professionals are lazy for embracing cloud computing"*.

→ Humble expects a culture of collaboration between developers and the infrastructure staff to **prescind from a "DevOps team"** [Hum12]. We understand this criticism applies to DevOps teams with dedicated members, like 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 [Kni14].

→ 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 4.3). One possible reason is the lack of proper test automation (I22, I36)[SB20], which is a hard goal to achieve since nearly all developers must master it.

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. Such strategy seeks to lessen the negative impact of any deployment problem. 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. Nonetheless, this is a controversial theme: while some developers dislike release windows because it blocks deliveries (I24), others understand it as necessary (I31, I36).

Supplementary properties

For **classical DevOps** organizations, we found **infra as development collaborator** as the only supplementary property: 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.

> **Key characteristic of classical DevOps:** intense collaboration between developers and the infrastructure team.

4.3.3 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*" [Gra06] and with the "autonomous squads" at Spotify [Kni14]. This gives more freedom to the team, along with a great deal of responsibility. As interviewee I1 described: *"it's like each team is a different mini-company, having the freedom to manage its own budget and infrastructure."* That said, we found seven organizations with this structure, two organizations transitioning to this structure, and one transitioning out of it.

Core properties

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). In a scenario with too many products, organization I35 promotes alignment among the single **cross-function team** and the customer needs by making developers knowledgeable about the business.

→ **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 coding features in their spare time avoids such idleness (I35).

→ **Most of the cross-functional teams we interviewed were in small organizations** (Table 4.4), likely because there is no sense in creating multiple teams in too small organizations.

Organizational structure	*Organization size*	*Number of interviews*
Siloed departments	< 200	3
Siloed departments	> 1,000	4
Classical DevOps	< 200	2
Classical DevOps	[200, 1000]	4
Classical DevOps	> 1,000	4
Cross-functional	< 200	5
Cross-functional	[200, 1000]	2
Platform team	[200, 1000]	2
Platform team	> 1,000	2
Siloed departments *to* Classical DevOps	[200, 1000]	2
Siloed departments *to* Classical DevOps	> 1,000	1
Siloed departments *to* Cross-functional	[200, 1000]	1
Siloed departments *to* Platform team	> 1,000	2
Classical DevOps *to* Cross-functional	< 200	1
Classical DevOps *to* Platform team	[200, 1000]	1
Cross-functional *to* Platform team	> 1,000	1

Table 4.4: *Organizational structures and organization size observed in our interviews*

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 involved technologies: front-end, back-end, and infrastructure; so I'm 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. Indeed, we heard of problems with database connections in a microservice context elsewhere (I27).

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 (considering the complexity of IoT deployment requires dedicated staff). Interviewee I19 argued that such an arrangement could not sustain growth in his company in the past: *"as we grew up, we saw we couldn't do everything if we wanted our product to be improved quickly... we can't do all the maintenance, monitoring, etc. So we really need an external team to manage that."*

Key characteristic of cross-functional teams: self-sufficient teams in charge for both development and infrastructure management.

4.3.4 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.

Core properties

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 is related to an infrastructure service (I8, I9, I12, I33). This situation differs from the **classical DevOps** scenario, in which usually the infrastructure staff is the first to take care of any incident, summoning developers only if needed.

→ 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 among data-centers (I4, I8, I16, I33). As participant I33 told us, *"you don't 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 between 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**, as it would be the case for **cross-functional teams**. Participant I33 expressed wanting to understand better 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, differently from a **siloed structure** ("throw over the wall" aptitude).

→ **The platform may not be enough to deal with particular requirements,** which happens when a given project is so different, regarding non-functional concerns, from other projects within the organization. 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 [Mor16] 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 4.3). 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. Therefore, we claim that having a **platform team** is a promising way to achieve high delivery performance. As stated by B5, *"the idea is to facilitate the life of development teams: they get these [monitoring] dashboards for free, without effort."*

→ In Table 4.4, no organization with **platform teams** 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 Microsoft 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); i.e., it *"enhances the usability of the [cloud] infrastructure"* (I16).

We also observed a small company with a "platform mindset" but without the resources to build a platform of its own (I14). So the infrastructure team prepares Terraform[4] templates encompassing good practices, so developers would need only to provide a few data fields for deployment.

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[5], an open-source platform used for managing the lifecycle of Docker containers.

In-house open-source platform. The platform is an open-source software deployed on-premise. Organization I20 uses Rancher[6], a graphical interface for developers to interact with Kubernetes.

[4] https://terraform.io

[5] https://kubernetes.io

[6] https://rancher.com

Key characteristic of platform teams: infrastructure team provides a platform with highly-automated infrastructure services to empower product teams.

4.3.5 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 organizational structures, as depicted in Figure 4.3.

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 do not follow 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 platform 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**.

4.3.6 Transitioning

Organizations are not static. We identified nine of them transitioning from one structure to another. Considering the transition flows in Figure 4.5, 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 empirical 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.* [NSP16], 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 I20 declared that *"it's not yet everybody together."* Similarly, interviewee I21 stated that *"There are two worlds... one was born in the cloud, and it's 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."*

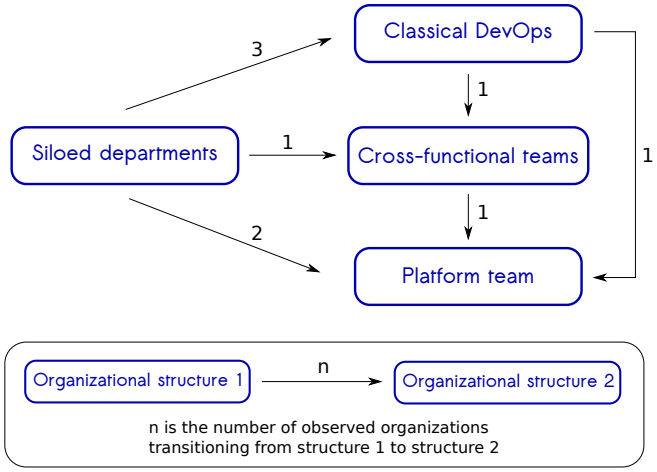

Figure 4.5: *Observed transition flows*

4.3.7 Summary

We close the description of our taxonomy by highlighting the key differences among its organizational structures. Table 4.5 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).

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 in specific situations, not on a daily basis

Table 4.5: *Summary of organizational structures – operations responsibilities and interactions in each structure*

4.4 Member check

Grounded Theory aims to formulate a theory that has relevance for practitioners, so it is crucial to also investigate whether findings make sense to them [Ral19]. Moreover, practitioners can help to identify taxonomy errors, such as inclusion and exclusion errors [Ral19]. Therefore, we collected feedback on our taxonomy from the study participants, using an online survey (available in the supplementary material – see Section 6.2).

We sent to each one of the 37 interviewees the feedback form asking whether the

interviewee agreed with the chosen classification (organizational structure and supplementary properties) for its context using the following Likert scale [Jos+15]: 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 any elements. Finally, we also left a free field for general comments. We report the answers to our open questions in Appendix D. 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 taxonomy.

We also attached to the emails a digest describing our taxonomy. It briefly introduced each organizational structure, each **core property**, and each **supplementary property**. We provide this digest as supplementary material. Figures 4.6, 4.7, 4.8, and 4.9 are the figures we used in the digest to convey the structures to the participants; we exhibited them here with the same captions used in the digest.

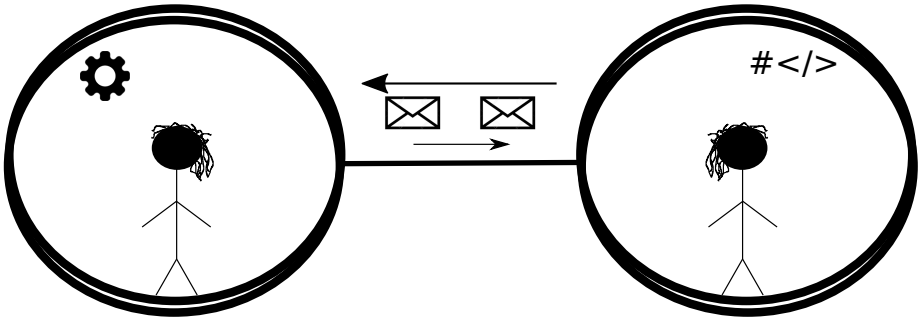

Figure 4.6: *With siloed departments, operators and developers are each one in their own bubbles. They do not interact directly too much and communication flows slowly by bureaucratic means (ticket systems).*

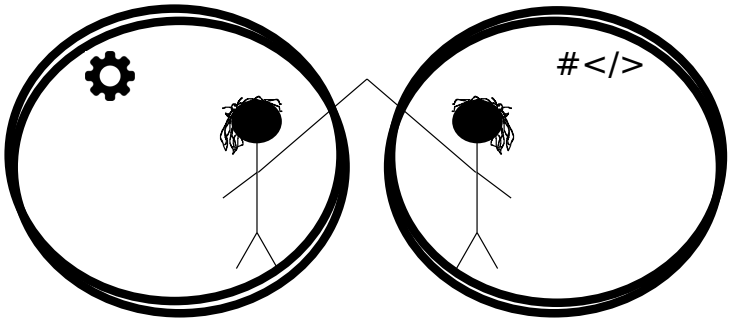

Figure 4.7: *With classical DevOps, operators and developers in different departments seek to work together, even if not easy, by direct contact and close collaboration.*

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

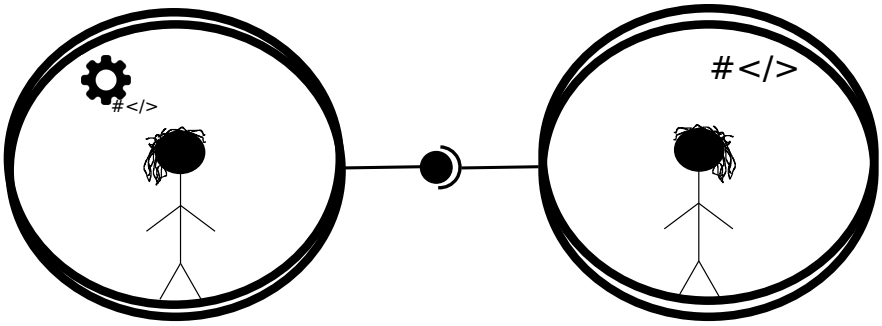

Figure 4.8: *The platform team provides automated services to be used, with little effort, by developers. The platform team and developers are open to hear and support each other.*

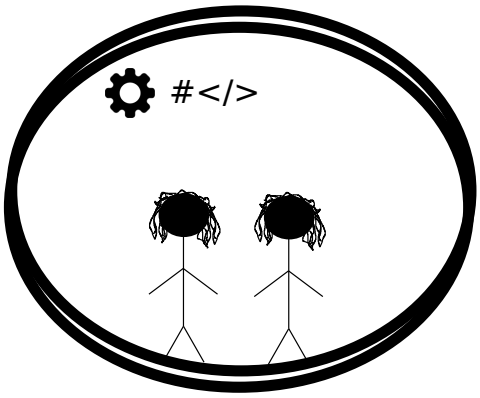

Figure 4.9: *A cross-functional team contains both operators and developers.*

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 express better 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.

4.5 Comparison to other taxonomies

In this section, we discuss the existing literature [HM11; Kim+16; NSP16; MBK18; SP13; SP19; Sha+17] by comparing their classifications of inter-team relations to our taxonomy.

In one of the foundational writings on DevOps [HM11], 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 between 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.

In the The DevOps Handbook [Kim+16], its authors describe three primary types of organizational structures: *functional-oriented*, optimizing for expertise, division of labor, and cost; *market-oriented*, optimizing for responding quickly to customer needs; and *matrix-oriented*, attempting to combine functional and market orientation. In the sequence, they campaign for companies embedding ops in service teams (**cross-functional teams**) or using automated self-service platforms (**platform teams**). The rationale is to avoid hand-offs caused by the segregation of development and operations. However, they acknowledge the existence of admired companies retaining functional orientation of operations. In such cases, mitigations include high-trust culture, slack in the system, encouraging employees to be generalists, keeping team sizes small, and curtailing the scope of each team's domain. In this way, they expect to reduce communication and coordination needs so that operations are not a constraint for product teams.

Nybom et al. present three distinct approaches to DevOps adoption [NSP16]: (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 between 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 between 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 [MBK18], offering 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 [SP13], 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 between the DevOps topologies (T) / anti-patterns (AP) and our taxonomy. *Dev and ops silos* (AP) corresponds to our **siloed departments** structure. *Dev don't need ops* (AP) 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 *rebranded sysadmin* (AP) since there were cultural changes in the

observed cases. *Ops embedded in dev team* (AP)[7] 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* (AP). *Dev and ops collaboration* (T) corresponds to our **classical DevOps** structure. *Ops as infrastructure-as-a-service (platform)* (T) corresponds to our **platform teams**. We consider *SRE team* (T) 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 [Bey+16].

The *Team Topologies* book [SP19], from the same authors of the DevOps topologies, 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. 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 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 between team topologies and 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 influence.

Most of the discussed work so far [NSP16; HM11; MBK18; SP13] 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 even so, it lacks a scientific methodology. In this way, Shahin *et al.* [Sha+17] present a closer work to ours, with a set of structures based on field data and scientific guidelines.

Shahin *et al.* [Sha+17] conducted semi-structured interviews in 19 organizations and surveyed 93 practitioners to empirically investigate how development and operation teams

[7] It is interesting to note that while *Ops embedded in dev team* is an anti-pattern for Skelton and Pais [SP13], it is advocated by other authors [Kim+16; SB20]. Skelton and Pais' rationale is that "the value of Ops is diminished because it's treated as an annoyance for Devs (as Ops is managed by a single Dev team manager with other priorities)."

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 structures.

Shahin *et al.* [Sha+17] 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 4.4: **classical DevOps** was less observed in small organizations, while **cross-functional teams** were not prevalent in large organizations. In other paper, Shahin and Babar recommend adding operation specialists to the teams [SB20], which favors **cross-functional teams** with dedicated infra professionals.

Lopez-Fernandez et al. also developed a taxonomy of DevOps Team Structures [Lop+21]. They employed Grounded Theory considering: industrial workshops[8], observations at organizations, and chiefly 31 semi-structured interviews in software-intensive companies. They analyzed six variables in the interviewed companies: shared ownership, leadership from management, organizational silo, cultural silo, collaboration frequency, and autonomy. Based in such analysis, they derived four patterns of DevOps Team Structures: *Pattern A: Interdepartmental Dev & Ops collaboration*, *Pattern B: Interdepartmental Dev-Ops team*, *Pattern C: Boosted cross-functional DevOps team*, and *Pattern D: Full cross-functional DevOps team*. Pattern A relates to our **classical DevOps** structure, while Pattern D corresponds to **cross-functional teams**. Pattern B is a matrix structure (employee reports to both a product and a functional manager), and Pattern C is a transitory pattern, in which DevOps experts support a team until it reaches Pattern D.

These researchers [Lop+21] also found that horizontal teams usually support product teams that follow the above-described patterns. Pattern A and Pattern B are usually supported by platform teams (in the same sense of our **platform teams**). They observed cases in which the platform is provided by a DevOps chapter, gathering DevOps experts of various product teams. Also, Patterns C and D are supported by a DevOps Center of Excellence (which we classify as an **enabler team**). Moreover, they concluded that companies implementing Patterns C and D (that they considered more mature) achieve better software delivery performance (measured with lead time, mean time to recovery, and delivery frequency). However, they did not consider horizontal teams in their analysis of delivery performance, thus achieving a conclusion that contrasts with ours (we concluded that **platform teams** are a significant driver for high delivery performance).

[8] https://www.youtube.com/watch?v=rDHv3dK_Am8

Finally, we note that Lopez-Fernandez et al. developed their work simultaneously to the production of this doctoral research. They acknowledge that by 2018, little work had empirically analyzed team structures in companies adopting DevOps. They also acknowledge that the work of theirs, ours, and that of Shahin et al. [Sha+17] bring common elements and different insights, concluding that "it is important to take into account that the aforementioned works of research are contemporary and have been carried out in different parts of the world, so that a triangulation of studies could be later analyzed in a systematic study."

Chapter 5

A theory for explaining organizational structures

In this chapter, we present the outputs of our research Phase 3. First, we explain our approach to the participant selection and the resulting sample (Section 5.1). Then, we explain our saturation criteria to stop selecting new participants (Section 5.2). Later, and most importantly, we present the results of the refinement process (Section 5.3) and the results of framing our theory under the 6C format (Section 5.4). We also present the result of our member check process (Section 5.5) and a discussion on the results (Section 5.6), which links our findings to our research questions.

In particular, through the refinement process, we renamed the structures like this: **siloed departments** to **segregated departments**, **classical DevOps** to **collaborating departments**, **platform teams** to **API-mediated departments**, and **cross-functional team** to **single department**. Beginning in this chapter, we employ this evolved terminology.

5.1 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 Phase 2 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 (as discussed in Section 4.5), we also aimed to select organizations of different sizes. Then, to accomplish our selection goal, 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 of revisiting 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 5.2). As a form of triangulation, we also sought to interview a developer and an infrastructure professional from each organization, whenever it was applicable and possible.

Within Phase 3, one can see our sample as following a purposive strategy, in which selection follows some logic or strategy [BR20; Pat14], possibly occurring before analysis. This deviates from the GT prescription of theoretical sample, in which selection criteria are driven by data analysis [GS99]. However, considering our research process as a whole, our selection process for Phase 3 fits theoretical sample since it is dependent on the concepts and findings emerged in Phase 2.

Nonetheless, we note that having access to IT professionals willing to expose organizations' internal affairs cannot be guaranteed in advance. Therefore, although the above strategy guided our selection, we could not strictly stick to it. In this way, Table 5.1 presents the 31 carried interviews in Phase 3, which extend the previous 37 interviews to the total of 68 interviews 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, healthcare, logistics, telecommunications, public administration, development support, hiring, marketplace, travel, manufacturing, finances, mobility, games, defense, networking, semiconductors, IoT, and cloud computing. One interviewed company was a tech giant, while three were unicorns.

Although we did not plan to balance the structures among **new** companies, in the end, our sample was reasonably balanced. From the 17 **new** companies, four presented the **collaborating departments** structure, six showed the **single department** structure, and seven had the **API-mediated** structure. For this count, 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.

Figure 5.1 shows how we interleaved data collection (interviews) with data analysis

Revisit	*Organizational structure*	*Number of employees in the organization*	*Reference codes and roles of interviewees*	*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

Table 5.1: *Description of participants and organizations*

(6C and resonance analysis), as advocated by Grounded Theory.

5.2 Saturation

At each analysis snapshot (an interview analysis or a review session), we tracked a few metrics to identify theoretical saturation [GS99]. 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.

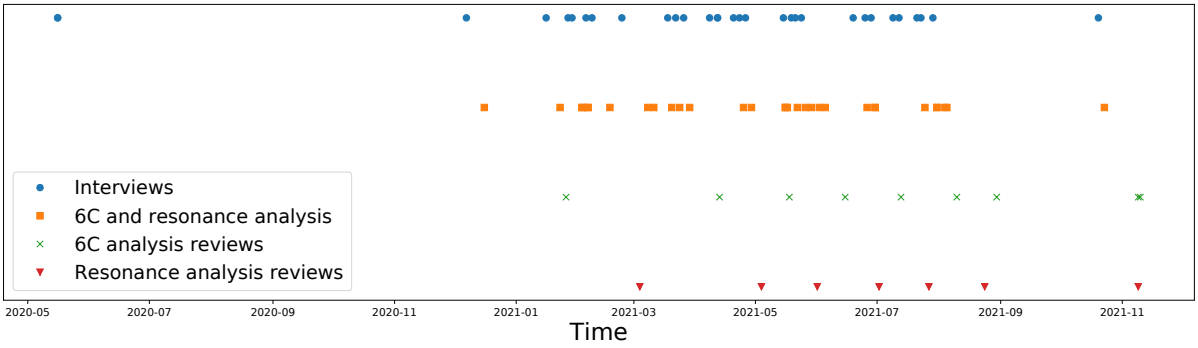

Figure 5.1: *Interleaving of data collection and analysis in Phase 3*

As observable in Figure 5.2, 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, 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.

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 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 5.3, 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 [HNM11; vv13; WNA15].

5.3 Results on refining the structures

After the 37 semi-structured interviews of Phase 2, following GT guidelines, we concluded the elaboration of what we considered our first taxonomy version. Figure 4.2 illustrates it. After that, we gathered feedback from participants through online surveys and started the interviews of Phase 3. To ask feedback, we shared with the participants a taxonomy digest, containing Figure 4.2. Based on this referred feedback and on the first two interviews of Phase 3, we elaborated more on our taxonomy until its fourth version (Figure 4.3), which figured in our journal publication [Lei+21].

After that, while interviewing more 31 participants, we conducted our resonance analysis process, in which we coded key points as **support** or **confusion** fragments. Key

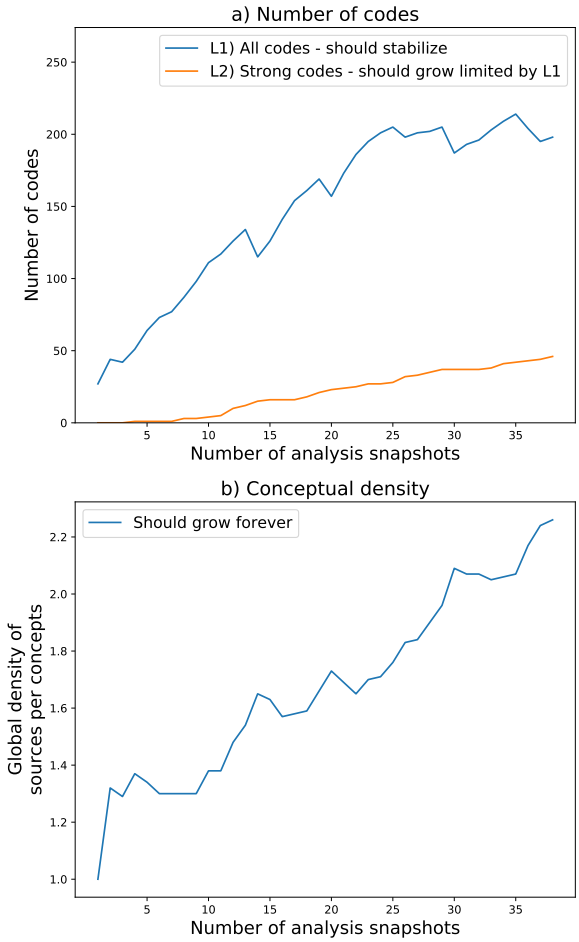

Figure 5.2: *Metrics of theoretical saturation for the 6C analysis*

points are interview excerpts that we considered of theoretical interest for the coding activities. Figure 5.4 features a word cloud built from the key points, thus providing some view of the discussed topics during the interviews. When comparing this word cloud to the word cloud produced for Phase 2 (Figure 4.4), we note that some main themes persist (e.g., development, infrastructure, team, platform, and people), while some disappear or decrease (e.g., DevOps, monitoring, problems, and production), and others emerge (e.g., think and company). Interestingly, we can relate the "DevOps" disappearance to the usage of our taxonomy, whose well-grounded concepts diminish the need to talk in terms of "DevOps." The themes of "monitoring/problems/production" decreased since we had specific questions about them in Phase 2. Finally, questions of Phase 3 were more abstract than questions of Phase 2 (e.g., "How did your company arrive at this structure?"), so participants expressed themselves much more in terms of "I think" when talking about their companies as a whole.

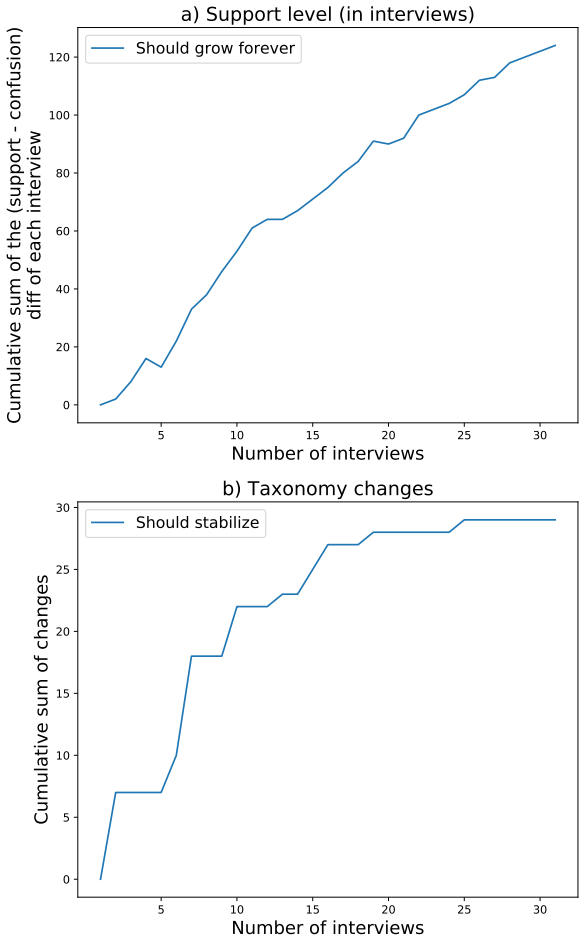

Figure 5.3: *Metrics of theoretical saturation for the resonance analysis*

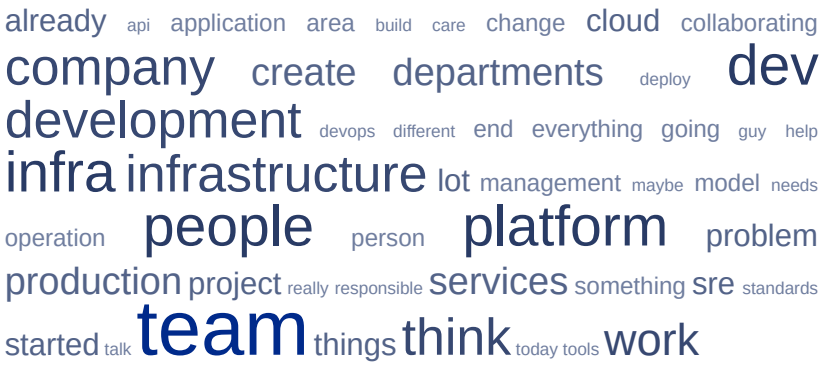

Figure 5.4: *Word cloud built from the key points of Phase 3*

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*. The "change the taxonomy" actions affected 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[1]. Table 5.2 discloses all the 35 taken actions. We provide the list of **supports** and **confusions** as supplementary material (see Section 6.2).

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 [GS99] during analysis:

> "*Good point... maybe we have to expand the scope of the in-house platform and the customized platform even when built in the cloud, 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 that 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.

A 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 ambiguous 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.

Finally, one more example of how we evolved our taxonomy. After a discussion with I52 about the terms infrastructure and operations, we reflected that the "no infra background" situation could happen because in some **single departments** the operations work is much bigger than the infrastructure work ("lightweight infra effort"). Thus, after revising I3 and considering it to match **collaborating departments**, we dropped the

[1] http://ime.usp.br/ leofl/devops/2021-06-20/structures-digest.html

After interview	*Type*	*Action*
39	Change the taxonomy	"With infra experts" renamed to "dedicated infrastructure professionals"
39	Change the taxonomy	"With full-stack engineers" renamed to "developers with infra background"
39	Change the taxonomy	"With no infra experts" renamed to "no infra background"
39	Change the taxonomy	"With a platform of its own" renamed to "customized private platform"
43	Change the taxonomy	"Classical DevOps" renamed to "Collaborating departments"
43	Change the taxonomy	"Cross-functional teams" renamed to "No infra department"
43	Change the taxonomy	"Infra people occasionally embedded in dev teams" added
44	Change the taxonomy	"No infra department" renamed to "Unified departments"
44	Change the taxonomy	"Platform team" renamed to "API-mediated departments"
44	Change the taxonomy	"Infra people occasionally embedded in dev teams" renamed to "Infra people occasionally embedded in product-centered dev teams"
44	Change the taxonomy	"Developers with infra background" renamed to "developers with infra background and attributions"
44	Change the taxonomy	"No infra background" renamed to "with infra attributions, but no background"
44	Change the taxonomy	"Cloud façade" renamed to "cloud façade with specialized API"
44	Change the taxonomy	"Customized private platform" renamed to "custom platform"
44	Change the taxonomy	"In-house open-source platform" renamed to "in-house-administered open-source platform"
46	Improve interview presentation	When explaining collaborating departments, provide as example "ops take part in agile meetings" and project visible to ops (beyond the ticket)
47	Change the taxonomy	"Siloed departments" renamed to "siloed dev & infra departments"
47	Change the taxonomy	"Collaborating departments" renamed to "Collaborating dev & infra departments"
47	Change the taxonomy	"API-mediated departments" renamed to "API-mediated dev & infra departments"
47	Change the taxonomy	"Unified departments" renamed to "Unified dev & infra departments"
48	Improve interview presentation	When explaining "collaborating departments", explain "infra people embedded in dev teams" too (to avoid confusion with single department)
50	Change the taxonomy	"Unified dev & infra departments" renamed to "Single dev/infra department"
50	Improve report	Reinforce in the API-mediated departments: there is an infra group responsible for the running infrastructure
50	Improve report	Reinforce in the API-mediated departments: devs operate and deploy services *in production*
51	Improve interview presentation	Highlight the difference between "infra people occasionally embedded in dev teams" and "unified departments"
52	Change the taxonomy	"With infra attributions, but no background" removed
52	Change the taxonomy	"Lightweight infra effort" added
54	Change the taxonomy	"Deployment pipeline provider" added
54	Change the taxonomy	"Consulting team" added
54	Improve interview presentation	Emphasize about "single department": there isn't a central infra group
56	Change the taxonomy	"Coordination committee" added
56	Improve interview presentation	Reinforce: diagram boxes are unordered
56	Improve report	Reinforce: while there is some correlation, cloud usage is not necessary in single departments.
57	Improve report	A relevant difference between collaborating departments and single department is the hierarchical structure (in single depart same boss to devs and infra)
62	Change the taxonomy	"Siloed dev & infra departments" renamed to "Segregated dev & infra departments"

Table 5.2: *Summary of the actions took during our resonance analysis*

"no infra background" concept and created the "lightweight infra effort" **supplementary property**.

Figure 5.5 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 (see Section 6.2), while we disclose the versioning history in Table 5.3.

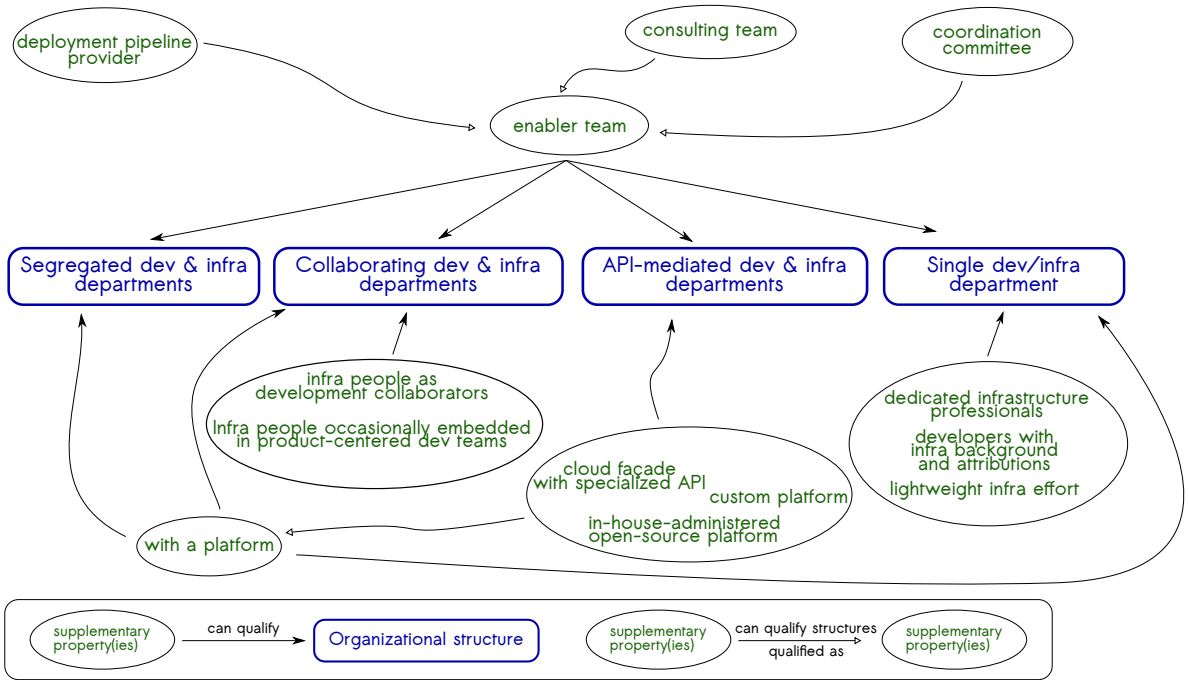

Figure 5.5: *High-level view of the 12th version of our taxonomy, the last one produced by our refinement process*

Version	*Date (yyyy-mm-dd)*	*Last analyzed interview*	*Observation*
v1	2020-04-30	I37	
v2	2020-05-24	I39	
v3	2020-05-29	I39	Submitted to the IST journal
v4	2020-08-30	I39	English review, published in the IST journal
v5	2021-02-04	I43	
v6	2021-03-08	I44	
v6.1	2021-03-08	I44	Just visual reordering
v7	2021-03-20	I47	
v8	2021-05-04	I51	
v9	2021-05-16	I54	
v10	2021-05-17	I53	We analyzed I54 before I53
v11	2021-05-26	I56	
v12	2021-11-09	I68	Submitted to the TSE journal

Table 5.3: *Taxonomy versioning history*

After the resonance analysis process, we also checked the consistency of the taxonomy's **core properties** with our new results. Thus, we decided to drop the property "*no*

conflicts over who is responsible for each task" of **collaborating departments** since we found conflicting evidence regarding this concern. We then updated our taxonomy digest accordingly.

5.4 Results on explaining the structures

Now we present, in Tables 5.4, 5.5, 5.6, and 5.7, 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. These strong codes, alongside the refined taxonomy, comprise our grounded theory. No other taxonomy available in the literature, built to explain the organization of development and infrastructure professionals, provide such a level of theoretical analysis given the explanatory dimension added to our taxonomy by the herein reported strong codes.

	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 5.4: *Strong codes for segregated 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)

Table 5.5: *Strong codes for collaborating dev & infra departments*

During the coding process, we linked a few codes to **supplementary properties** of our taxonomy. The **strong codes** 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 5.8, 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

	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

Table 5.6: *Strong codes for single dev & infra departments*

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 (see Section 6.2). In the "6C analysis sheet", each code is linked to its supporting interviews. In Appendix C, we also provide examples of supporting fragments from the interviews to some strong codes.

5.4.1 Covariance analysis

Covariance, part of the 6C analysis, means the occurrence of one code correlates with the occurrence of another code [HNM11; Gla78]. Covariance analysis reveals codes commonly supported by multiple interviews, providing more insight 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 5.9 describes the last four just-listed **strong covariance groups**, which are the strongest ones since they have more codes or more sources than the first 17 groups.

5.5 Member check

Grounded Theory aims to formulate relevant theories for practitioners, so it is crucial to investigate whether findings make sense to them [Ral19]. However, it is opportune to highlight that the goal of our member check is to assess only the theory resonance [Ral19]

	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., NFR)
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)

Table 5.7: *Strong codes for API-mediated dev & infra departments*

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 [Gla02]. Therefore, readers should consider member check together with other quality treatments (Chapter 6).

We performed a member check by collecting feedback on our results from the participants using online surveys, which we provide as supplementary material (see Section 6.2). 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

	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 5.8: *Amount of codes found per code class*

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

Table 5.9: *Strong covariance groups*

scale [Jos+15]: “true”, “usually true”, “no correlation / I don’t know”, “usually false”, “false”. We also left a field for general comments. We report the answers to this open question in Appendix D. 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 2 interviewees, (ii) to Phase 3 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. 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 5.4), which possibly jeopardized respondents’ reasoning. This issue was fixed for the second round of feedback requests.

5.6 Discussion

This section discusses the relationship between our findings and our research questions **RQ2** and **RQ2.1**, besides drawing some implications of these results.

RQ2 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). **Collaborating 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**.

RQ2.1 inquires about the drawbacks and mitigation measures for each structure. **Avoidance reasons**, **consequences**, and **contingencies** in the 6C analysis provide us answers. Although planned to increase direct cooperation (SC09), **collaborating 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[2], 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 their careers. In this context, we witnessed 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.

We discuss more concrete implications of strong codes to practitioners in Appendix C. There, we further discuss five strong codes (SC04, SC28, SC19, SC11, SC46) – one condition, one cause, one avoidance reason, one consequence, and one contingency. This additional discussion unveils some rich insights behind strong codes, such as: bosses fearing losing

[2] 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.

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).

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 may split a common solution to different problems 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 **collaborating 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 **collaborating departments** (SC10, SC11, SC12). We suppose it may be easier for large organizations with **segregated** structures trying to move first to **collaborating 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 **collaborating 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 **collaborating departments** and delivery performance (Section 4.3).

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 discussing again and again what DevOps is[3] [Lei+19].

[3] See, for example, the "Chef Style DevOps Kungfu" talk.

Chapter 6

Quality criteria and limitations

According to Ralph, good taxonomies should increase cognitive efficiency and assist reasoning by facilitating more inferences [Ral19]. A path to evaluate taxonomies is to conduct case studies [Yin09]. However, although case studies can test theories [Eis89], they are more helpful in demonstrating how a current theory is incomplete or inadequate to explain observed cases [Pin86; And83]. 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 [Sir+11]. On the other hand, Grounded Theory (GT) focuses on generating theories rather than validating preconceived hypotheses. Although researchers can be tempted to try to validate their theory as soon as it is created, Glaser and Strauss caution that verificational approaches hinder theory development too early [GS99].

Though it is inadequate to evaluate our theory as simply valid or not, we can *evaluate our process of generating the theory* [GS99; Ral19]. This can be done by checking adherence to GT prescriptions (i.e., theoretical sampling, open coding, theoretical coding, theoretical sensitivity, and theoretical saturation), which we address in the chapters containing our research design and results. Ralph still cautions that researchers should be careful when selecting evaluation criteria to assess a theory development. Since multiple criteria exist and there is no universally accepted set of criteria, researchers must choose what make sense for the emerging theory [Ral19].

For our research, we chose the quality criteria of Guba for naturalistic inquires [Gub81]. Although the naturalistic paradigm contrasts with the rationalist one [Gub81], in which classical GT is based, the use of Guba's criteria for a social and abstract phenomenon is suitable. In this way, these criteria are widely employed by software engineering studies adopting GT as methodology [Wat14; Jov+17; SB20; Lop+21]. In the following section, we present such criteria and how we meet them. In particular, we devote a whole section about generalizability, one of these criteria, since it requires a more in-depth discussion. Finally, we also present the limitations of this work, considering especially Ralph's recommendations for taxonomy generation [Ral19].

6.1 Quality criteria

The quality criteria we pursued are the ones defined by Guba [Gub81]: credibility (how plausible or true the findings are); dependability (methodology applied consistently); confirmability (opportunities for correcting research bias); and transferability (generalizability). For meeting such criteria, we applied the following treatments [Wat14]:

- Providing a chain of evidence (available in the supplementary material – see Section 6.2), which contributes to credibility and dependability.
- Collecting interviewees' opinions on the results, i.e., member check (Sections 4.4 and 5.5), which contributes to credibility and confirmability.
- Having a selection of participants with varying roles, experience levels, and genders, working in companies with varying sizes, locations, and domains (Sections 4.1 and 5.1). In particular, in Phase 3, we triangulated interviews with development and infrastructure staff within some organizations. These procedures contribute to transferability.
- Triangulating among researchers through a review process (Sections 3.7 and 3.11) and methodology revision by a colleague experienced in qualitative methods [Mel15], which contribute to confirmability.
- Providing quantified evidence of saturation (Sections 4.2 and 5.2), which contributes to dependability (this is an additional quality measure, not standard in other works).
- For Phase 3, reporting only codes confirmed by multiple participants (**strong codes**), which contributes to credibility and transferability.
- In Phase 3, we also conducted a process for refining our taxonomy based on our interactions with participants (Sections 3.9 and 5.3). This process improved confirmability (we evolved the taxonomy) and transferability (we applied our taxonomy to scenarios that did not feed its initial versions).

Table 6.1 offers another view on the trace between quality criteria and their treatments, so the reader can more quickly check the treatments we adopted for each criterion. In this way, one can see we applied at least two treatments for each criterion. The table also indicates the section of this thesis tackling the referred treatment.

6.2 Supplementary material

The supplementary material to this thesis, available online[1], provides the chain of evidence supporting the findings here reported. In this way, readers can fully judge how much data back our theory.

We provide the following files as supplementary material for Phase 2:

[1] http://ime.usp.br/~leofl/devops/assets/files/mst.tar.gz.

Treatment	*Credibility*	*Dependability*	*Confirmability*	*Transferability*	*Section*
Chain of evidence	X	X			Supplementary material
Member check	X		X		4.4 and 5.5
Diverse sample				X	4.1 and 5.1
Researchers triangulation			X		3.7 and 3.11
Evidence of saturation		X			4.2 and 5.2
Reporting stronger evidences only	X			X	3.10
Refinement process in Phase 3			X	X	3.9 and 5.3

Table 6.1: *Our treatments for the adopted quality criteria*

- Chain of evidence (PDF), linking our theory's primary elements (the structures and the supplementary properties) to codes, memos, and transcript excerpts that originated them.
- Conceptual framework history (PDF), which are all the versions of the diagram representing our conceptual framework.
- Feedback survey (PDF), with the questions sent online to the study participants.
- Structures digest (PDF), elaborated to support participants in giving us feedback[2].

We provide the following files as supplementary material for Phase 3:

- Key points (PDF), which are the transcription excerpts used in the 6C and resonance analysis.
- Resonance analysis sheet (XLS), providing the fragments coded as **support** or **confusion**.
- 6C analysis sheet (XLS), with the applied theoretical coding.
- Taxonomy versions (PDF), which are all the versions of the diagram with the high-level view of the taxonomy.
- Feedback survey (PDF), with the questions sent online to the study participants.

For Phase 2, the "chain of evidence" document directly links high-level concepts to supporting interview fragments. For Phase 3, one must find the interviews linked to high-level concepts (theoretical codes) in the "6C analysis sheet", and then search for support in the corresponding interviews in the "key points" document.

[2] An updated version is available on http://ime.usp.br/ leofl/devops/2021-06-20/structures-digest.html.

6.3 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 [Yin09] 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 Phase 3, we applied our taxonomy in conversations with professionals working in companies that did not provide data to the classification construction in Phase 2. 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 Phase 2. In Phase 3, 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 Phase 3, 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% of managers in Phase 2 vs. 61% in Phase 3).

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. 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 bottleneck previously existing. Another reinforced findings are that **single department** 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 versions; *(iii)* gathering evidence of support for the theory among **new** companies; and *(iv)* the discovery of more evidence corroborating preliminary findings.

6.4 Limitations and threats to validity

Despite our best efforts to meet the exposed quality criteria, every work has threats to validity. We now discuss them.

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 [Ral19]. Thus, it is not appropriate to discuss, for a grounded theory, the number of interviewees in terms of statistical sampling.

As usual to grounded theories, some factors (such as interviewees anonymity and unavoidable subjectivity in analysis) make the research not fully replicable. In particular, anonymity goes along with ethical research [Str19]. It is also a crucial trade-off with the potential number of interviewees: many participants might not accept an invitation to participate in a non-anonymous interview. Moreover, anonymity copes with social desirability bias [Dd14]. Still, violating anonymity does not provide reproducibility: interviewing the same person again does not necessarily yield the same results[3].

There is another subtle disadvantage in employing GT, which is a drawback of embracing empirical methodologies. Other approaches, such as the one advocated by Doty and Glick to build typologies[4], claim to free researchers from empirical world limitations. These authors state that "because typologies are based on ideal types of organizations, they may allow researchers to identify types of organizations that are more effective than any organizations currently observed" [DG94]. This statement implies that, while seeking to guide practitioners regarding organizational structures, a grounded theory will miss optimal structures if they do not exist yet in the real world.

Although we relied on previous work [FHK18; For+20] to adopt the delivery performance construct, we needed to adapt the original thresholds due to the reasons exposed in Section 2.2. 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 [For+17]. Therefore, by changing some decisions related to delivery performance, other researchers could reach different conclusions based on the same data. We handled this concern mainly by stating our definitions regarding this topic. Nonetheless, as already stated, GT does not guarantee that two researchers working in parallel with the same data would achieve identical results [GS99]. Still, relying on self-reported metrics may be subject to some bias; nonetheless, collecting such metrics from system data in many organizations would be extremely difficult.

In an ideal process, different researchers would analyze the interviews independently, avoiding some possible agreement bias favored by review discussions (one could even calculate interrater agreement scores [Ban+99]). However, the cost for independent analysis would be too onerous (analyzing one interview takes hours). Regarding the review process,

[3] "One cannot step into the same river twice" [Vla55].

[4] Typologies identify multiple ideal types with no rules to classify instances; each ideal type represents a unique combination of attributes that determine relevant outcomes [DG94].

instead of relying on the transcripts produced by a single researcher, 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 research collaborators took part in a few initial interviews to standardize the transcription procedures and assess how the doctoral candidate conducted these interviews. Beyond this, the researchers who take part in the analysis read some transcription excerpts only. 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 [Ral19]. Rother, for example, argued about how an observational approach was crucial for his research since Toyota people had difficulty in articulating and explaining their unique thinking and routines [Rot09]. However, our research questions are too abstract to grasp only by observation, even meeting observations, without further conversations with the observed people. Our context thus differs from other software engineering situations, such as observing a pair-programming session. 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. Nonetheless, anonymity reduces such a "drift from objective reality" by favoring an open attitude from the participants. We were also careful so as not to ask the research questions directly to participants, which would force our preconceptions onto them [Ral19]. We carefully crafted second-level questions [Yin09], which are more objective than the research questions and, therefore, reduce the gap from report to reality.

We are aware that large companies usually present groups at different maturity levels, and that such groups could be classified, in Phase 2, 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. Moreover, as already stated, in Phase 3 we had more cases of two interviews for the same company.

In Phase 3, 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.

Chapter 7

Conclusion

Our research provides a theory on how software-producing companies organize their development and infrastructure workforce according to different organizational structures. Answering **RQ1** (*What are the organizational structures adopted by software-producing organizations to structure operations activities among development and infrastructure groups?*), such structures are:

1. **Segregated departments**, with seven **core properties**;
2. **Collaborating departments**, with eight **core properties** and two **supplementary properties**;
3. **Single department**, with four **core properties** and three **supplementary properties**;
4. **API-mediated departments**, with nine **core properties** and three **supplementary properties**.

Supplementary properties reveal relevant variations represented in our taxonomy, as, for example, that **single departments** may or may not encompass staff dedicated to infrastructure. Beyond the eight supplementary properties directly related to one structure each, we also found two other supplementary properties applicable to more than one structure. And for one of them (**enabler team**), we identified three subtypes. In this way, our taxonomy presents a total of 13 supplementary properties. The **core properties**, describing each organizational structure, plus the supplementary properties answer **RQ1.1** (*What are the properties of each of these organizational structures?*).

Our research also answers **RQ1.2** (*Are some organizational structures more conducive to continuous delivery than others?*) by finding evidence that **API-mediated departments** are more suitable for achieving continuous delivery given its relationship with delivery performance. An explanation fitting our theoretical analysis is that **API-mediated departments** provide a better balance between specialization (division of labor) and inter-team coordination (integration): workers are more specialized than it is the case for **single departments**, but integration is more fluid than for **segregated** and **collaborating departments**. Also, the **API-mediated** approach reduces the costs of forming autonomous teams by not demanding an infrastructure expert in each product team.

Still about **RQ1.2**, we also observed that **segregated departments** are more common in organizations with less-than-high performance. Additionally, there was no apparent relation between delivery performance and the other structures (**collaborating departments** and **single department**). Nonetheless, we critically state that reaching high delivery performance is not a need for every software-producing organization, as adopting **API-mediated departments** is not adequate for every organization. In particular, a promising topic for future research is investigating whether the organization domain influences the need of high delivery performance.

We, then, found why different organizations adopt (or do not adopt) different structures and the conditions leading to this choice. Such findings answer **RQ2** (*Why do different software organizations adopt different organizational structures regarding development and infrastructure groups?*). For example, a **cause** for **API-mediated departments** is trying to address delivery bottlenecks; an **avoidance reason** for **single departments** is its unsuitability in enforcing corporate standards; and a **condition** for **collaborating departments** is having a certain proportion of infra people per development team.

To answer **RQ2.1** (*How do organizations handle the drawbacks of each organizational structure?*), we identified the drawbacks of each structure and the ways organizations deal with such disadvantages. For example, a **consequence** of **collaborating departments** is possibly leading to conflicts due to blurred responsibilities; tuning the platform abstraction level is a **contingency** to 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. We hope our work can contribute to spreading such awareness.

7.1 Implications

Understanding the world is the primary goal of science [Knu74]. Many fields in physics, biology, and sociology, possibly more than in computer sciences, are dedicated to understanding world phenomena. Discovering regularities in the world and providing better vocabulary to describe such regularities lay the basis for both advancing science itself, through new research, and changing the world based on a more scientific comprehension of it. In this way, we hope our research has contributed to somehow evolving our views on the software production phenomenon. Such comprehension by academics is vital, in the first place, to support the teaching of software engineering to the rising number of computing students. In particular, a new vocabulary has the power to change practitioners' goals, expectations, and actions, besides basing discussions and arguments (consider the "technical debt" concept [Cun92], to take only one example).

Therefore, 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. Moreover, by increasing the awareness of organizational structures in the community, practitioners can better discuss the current situation of their corporations, besides making more informed decisions on structural changes and drawback handling. 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." Other example of applicability is that our taxonomy 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.

This work also has *implications for scholars and research*. The elements of our taxonomy, provide a common vocabulary to support the formulation of new research questions. For example, researchers can investigate the impact of each taxonomy 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** (as part of **API-mediated departments**). Also, 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 and IT instructors can update their software engineering classes accordingly. A secondary and methodological contribution of our work for developing new grounded theories is providing an objective approach to detect theoretical saturation, which is rarely seen in the literature.

In addition, we observed that test automation is still not adequately practiced in many software companies, as also noted by Olsson *et al.* [OAB12], 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 [BHJ16; Ebe+16]. 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. One more example of research pursuit is envisioning techniques to avoid release windows and, thus, increase delivery performance.

7.2 Future work

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) cannot be proven 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, each **strong code** could be submitted to validation studies or, at least, be a starting point to new investigations.

Finally, the organizational structure agenda applied to software engineering is vast. For example, it is still controversial how to share responsibility on security across a groups of experts, specialists embedded in product teams, and non-specialist developers [Man+19]. Therefore, software engineering research may still provide more light to practitioners on distributing responsibilities linked to specific software attributes (e.g., security, quality, performance, user experience). Alternatively, if it is not possible to clarify the best option for each concern, at least new research could unveil each structural option's benefits, drawbacks, and contingencies.

7.3 Final words

The workforce dedicated to software production has grown over the world. Nevertheless, the demands of clients have heightened too: new features must be shipped quicker and quicker. Furthermore, new features must support more users while dealing with more robust non-functional requirements (for example, society and governments are now demanding more security and data protection from software products than ever before).

In this scenario, it is not uncommon for companies to try to address such demands by simply pressuring more on their employees or expecting a single employee to master a vast set of skills. Correspondingly, it is also not uncommon to experience slow or unavailable software in our daily lives. Solving these issues is not just a matter of demanding more of individual employees. As told a long ago by Adam Smith, a single worker could barely produce 20 pins per day, while just ten employees, when wisely organized, could produce thousands of pins per day in a manufacture [Smi14].

To tackle the contemporary challenges of software production, we expect that advances brought by software engineering research, ours included, can promote more rational ways of organizing the software workforce. In particular, we hope that such progress benefits companies, which will produce more software, workers, which will have better work conditions, and consumers, which will enjoy better software.

Appendix A

Interview protocol for Phase 2

The purpose of this protocol is to guide the interviewer in the preparation, conduction and analysis of semi-structured interviews in the context of our research (organizational structures for infrastructure management in the context of continuous delivery) following academic guidelines, including Grounded Theory.

We also hope that this protocol can support other researchers in conducting semi-structured interviews for software engineering research.

A.1 Purpose of the interviews

By using Grounded Theory, start the development of a theory addressing the following research questions:

- RQ1 Which organizational patterns are organizations currently adopting?
- RQ2 Are there organizational patterns delivering better results than others?
- RQ3 Which forces drive organizations to take different organizational patterns?
- RQ4 Which are the advantages, challenges, and enablers for each organizational pattern?

The questions were designed to address mainly RQ1 and RQ2, and to start to deal with RQ3 and RQ4.

A.2 Inputs for this protocol

- Original research proposal (research questions above)
- Appendix “Research Protocol for constructing a Conceptual Framework and Mapping Regional Software Startup Ecosystems” of “A Conceptual Framework for Software Startup Ecosystems: the case of Israel” (a technical report of our research group)

- "Protocol to be used in the interviews at startups" (a document of our research group)
- Article "The Grounded Theory of Agile Transitions" (ICSE, 2017)
- Interview tips of Ijnet, a website for journalists
- Tips from Daniel Cukier, a former PhD student of our research group who conducted interviews within the startup environment
- Our experience with the brainstorm sessions with specialists
- The chapter "Conducting semi-structured interviews" of the book "Handbook of Practical Program Evaluation" (Adams, 2010).
- Our own experience with the interviews conducted under this protocol (we evolved this protocol with more tips based on this experience).

A.3 Method

Use of semi-structured interviews. According to Adams, "Conducted conversationally with one respondent at a time, the semi-structured interviews (SSI) employs a blend of closed - and open - ended questions, often accompanied by follow - up why or how questions. The dialogue can meander around the topics on the agenda — rather than adhering slavishly to verbatim questions as in a standardized survey — and may delve into totally unforeseen issues."

A.4 Target population

People who have (in the present or recent past) involvement with teams within software developer organizations who use continuous delivery of software or who are going through the process of adopting continuous delivery.

A.5 Guidelines and tips

Before the interview:

- Diversified selection. Organization: size, type, location. Interviewee: gender, position (from developer to marketing).
- Test audio recording before the first interview.
- For online interviews, check the audio recording before each interview.
- Choose a recorder that is as discreet as possible. At this point smartphones are suitable, since people are used to cell phones on the table.

- Avoid scheduling interviews in noisy places.
- When possible, prefer face-to-face interviewing to online interviewing.
- Avoid, if possible, asynchronous text-based interviews.
- Prepare a printed version of the questions of this protocol to assist the interviewer during the interview.
- If you make notes on the same board for different interviewees, use different colors.
- When inviting people, provide a link with a dynamic listing of the interviewers' available days and times. Try to offer as much as time slots as possible.
- Do not send too many invitations at once. You may fill all your slots and, thus, not have time to analyze between interviews.
- Send a LinkedIn connection invitation to the interviewee. It is an opportunity for the interviewee to know better who is interviewing her.

During the interview:

- Always be polite.
- Try to be pleasant. A quick unrelated conversation right before the interview can help. Example: "How long have you lived in the city?" Or "Why did you move to São Paulo?". Easier to do this when the interview is in person.
- Listen. Avoid talking unnecessarily.
- While the interviewee speaks, do not display strong emotions such as enthusiastic agreement, disagreement or surprise.
- Avoid giving opinion during the interview.
- Do not debate (try to contradict) the respondent's answers.
- You can demonstrate knowledge to the interviewee, but humbly, without putting yourself as more expert than the interviewee.
- At times, repeat what the interviewee said to show interest and understanding in the message.
- Respect the silence, let the interviewee reflect as much as necessary.
- On the other hand, prepare extra questions to keep the interviewee talking. Or improvise. Extra questions can also lead to unexpected revelations.
- Be open to explore other unforeseen topics. New questions more interesting than those already established may arise.
- Do not drive the interviewee to say what we want to hear. On the contrary, if possible try to make him speak what we do not want to hear.
- If the interviewee dodges the question, after a while try to return to the question with another approach. But do not insist too much. Especially in the first interviews, the avoidance may be a sign that the question is not very good.

- Do not follow the script mechanically. The order of the questions in the protocol don't need to be followed.
- Questions can be tailored to the interviewee's profile.
- Make hooks based on what the respondent commented.
- Do not be afraid to ask naive questions.
- Ask for clarification when needed. But avoid saying "what do you mean…?" As it seems that you are scolding the interviewee for not communicating properly.
- If the interview is online: if feasible, video chat; otherwise, at least try to greet the interviewee (and say goodbye) by video; and explain to the respondent that the video will be turned off for connectivity reasons.
- It may be impolite to ask the interviewee to turn on the video; The interviewer should just turn on his own video, which suggests to the interviewee to turn on her video too. If in this case the interviewee does not turn on the video, that's fine.
- Avoid using the term "DevOps" (broad and ill-defined in the community). But especially in explaining the motivation of the interview it can be difficult to escape from this.
- Avoid very specific terms related to the organizational patterns we already know of (e.g., SRE).
- Focus on the interviewee's latest experiences on her team. Do not ask for an explanation about the organization as a whole. Consider projects only from the recent past.
- Estimated length of interview: 1 hour.
- To calibrate the time, you may want to "test the script" before the first interview or conduct the first interview with someone you already know, who will be more tolerant of any endurance.
- Try to minimize note-taking to focus on the interviewee. Write down points during the conversation that lead to the next questions. Choose a standardized place or a different sheet for this, so you do not forget to ask these questions.
- For online interviews, a good software for audio recording is OBS. When starting the interview, pay attention to the audio capture bars (mic and desktop audio), so you are sure that audio is being recorded.

After each interview:

- Do the interview analysis as soon as possible. Advantages: 1) the interview is more "fresh" in memory and 2) we have a more agile (iterative) approach, since 10 analyzed interviews are worth more than 40 interviews without analysis.
- Check the audio. If the recording did not work, try to do the analysis on the same day or as early as possible.

A.6 Interview script

Before the interview:

- Collect respondent data. Do not spend interview time on this. Search for data, for example, in Linkedin.
  - Name
  - Linkedin
  - E-mail
  - Gender
  - Graduation course
  - Higher degree (no graduated, graduated, master, PhD)
  - Graduation year
  - Role
  - Years in the company
  - Years in the role
  - Company
  - Company founding year
  - Number of employees in the company
  - Number of IT employees in the company
- Research about the interviewee before the interview. This search goes beyond profile data. It helps to create “intimacy” with the interviewee.
- On the interview eve, remind the interviewee about the interview.
- If the interview is in person: arrive early (about 15 min); if possible try to know the environment of the company.

Starting the interview:

- Mention group researchers.
- Explain the purpose of the research.
- Mention expected benefits for the community (software producing organizations).
- Mention that the interview can also benefit the interviewee by helping her to reflect on her work environment.
- Explain the protocol.
- Explain why the recording and its confidentiality.
- Explain that the publications will treat companies and respondents anonymously.

- Explain that any results will be sent to the respondent firsthand.
- Mention that our group will be open for future collaborations.
- Ask permission to start recording.
- Start recording by saying "OK, it's recording now."
- Start with an easy and open question ("icebreaker"). Helps the interviewee become more comfortable.

During the interview:

- Record the audio.
- Record with the cellphone. But take the portable recorder in case of any problem with the phone.
- Take only a few notes (on the computer) but always remember to keep eye contact with the interviewee.
- If another researcher is present, the second researcher may take further notes.

Main questions:

- Please, tell me about your company, your role within your company, and the project in which are currently working.
  - Rationale: Icebreaker question
- Who is responsible for deploying the solution in production?
  - Rationale: Directly linked to RQ1
  - Rationale: Starting with "relevant but still non threatening questions" (Adams, 2010).
- In a new project, who builds the new environment?
  - Rationale: Directly linked to RQ1
- And about the Cloud? PaaS / serverless? What do you think?
  - Rationale: In our DevOps literature review, we already found that the use of cloud platforms, especially PaaS and serverless, can strongly impact on expected skills of software engineers.
  - Evolution notes: Usually we did not need to ask this directly.
- Who is in charge for non-functional requirements?
  - Integrity, availability, flow rate, response time.
  - Capacity planning, load balancing, overload management, timeout and retrials configuration.
  - Deploys without unavailability.
  - Rationale: Directly linked to RQ1

- Who is responsible for configuring monitoring?
  - Rationale: Directly linked to RQ1
- Who is responsible for tracking the monitoring?
  - Rationale: Directly linked to RQ1
- Who is on-call for incident handling?
  - After-hours frequency
  - Blameless post-mortem?
  - Rationale: Directly linked to RQ1
- Delivery performance:
  - Time from commit to production
    - < 1h / 1 week < t < 1 month / > 1 month
  - Deployment frequency
    - Under demand (many times per day)
    - 1 per week < f < 1 per month
    - f > 1 per month
  - Mean time for repair
    - < 1h / < 1 day / 1 day < t < 1 week / > 1 week
  - Frequency of failures (% of deliveries)
    - 0 - 1% / 1 - 15% / 15 - 30% / 30 - 50% / More than 50
  - Reasons for the above results.
  - Rationale: Directly Connected to RQ2. Provides a link between the organizational structure and its "delivery performance" ("better results" in RQ2). We use "delivery performance" as defined by Forsgren. The questions alternatives follow the results found in the State of DevOps Surveys.
- What would you change in this work system of your team/company?
  - Why is it hard to implement such changes?
  - Why did this problem arise? Why do they remain?
  - Rationale: Directly connected to RQ2. A more qualitative approach to feel the respondent's (in)satisfaction with the results of the organizational pattern in which she is immersed.
  - Rationale: Sub-questions help to answer RQ3, because at this point the interviewee will probably justify the reasons for the existing organizational structure.

  - Rationale: Sub-questions help to answer RQ4 since what could be improved may refer to "advantages, challenges and enablers" of the existing organizational pattern.
  - Rationale: Follows the guideline "rather than asking people to identify what is "bad," asking advice on "how to make things better" and "areas that need improvement" can help minimize defensiveness" (Adams, 2010).
  - Rationale: "The most potentially embarrassing, controversial, or awkward questions should come toward the end" (Adams, 2010). In that case, the most complicated of the main questions is at the end of the main questions.

Specific questions:

- Rationale: These questions should be dynamically chosen according to the "advantages, challenges and enablers" that have so far been revealed in the interview. They serve to deepen the understanding of the "advantages, challenges and enablers" of the existing organizational structure.
- Inter-team communication; problem?
- Little communication? Over communication?
  - Rationale: The classic idea of DevOps calls for closer approximation between devs and ops ("tearing down walls between silos"). But more communication is not necessarily better communication, especially between different departments. Organizational patterns may be directly linked to the need for communication between the organization's professionals.
- Are different teams aligned (committed to the project)?
  - How do this happen?
  - Consequences
  - Rationale: DevOps literature advocates that different teams and departments should be aligned. But little is said about how to achieve such alignment. However, in organizational patterns that maintain separate, collaborating departments (devs and ops), alignment is essential and, therefore, deserves to be questioned.
- Clear definition of responsibilities?
  - If not, any problem?
  - If yes, long hand-offs?
  - Rationale: There are reports of "DevOps deployments" in which devs and ops passed to share responsibilities, especially on deployment automation. However, reports show that conflicts arise from this responsibility sharing as professionals question the responsibilities of those involved in the project ("Am I doing something the other should do?", "Are they doing what I should do?").
- Is there a "DevOps team"? What is it about?

  - Rationale: The literature comments on the existence of DevOps teams in organizations, but little is known what exactly these teams are.

- Are there people with the "DevOps role"? What is it about?
  - Rationale: The literature also comments on the existence of the "DevOps engineer" or "full-stack engineer". However, the very initial idea of DevOps (devs and ops collaborating) seems to be against the idea of a DevOps role.
- If there are no "dedicated ops": how do software engineers improve their skills on infrastructure?
  - Were they hired with such skills?
  - Everyone in the team knows about infrastructure?
  - How people are chosen for "knowing more" about infrastructure.
  - Rationale: The lack of "exclusive ops" is a feature we expect to find in many organizations. It is associated with the so-called cross-functional teams. Advocates argue for higher productivity (less wait time between teams) in this model. But the big challenge is getting a single team to possess a wide range of expertise in different technical topics.
  - Evolution notes: After interview I20, we diminished the focus on this question and the following two ones since they were not contributing so much for the evolution of our taxonomy.
- Is there any kind of incentive from the company about continuous education?
  - community of practices
  - FLOSS
  - internships
  - teams mutation
  - timeshare for studying
  - Rationale: Again, in cross-functional teams, the ability to learn is even more important than in more traditional structures.
- If a team needs knowledge/skills that they do not have, what to do?
  - Study with team mate
  - Search for help in another teams
  - Search for in-home specialist
  - Search for external consultant
  - Rationale: Continuation of the previous question.
- Does each team have their own specialists? Are there shared specialists?
  - Sharing policy

  - Handoffs problems
  - Rationale: The way specialists spread across the organization is totally tied to the issues of organizational patterns.

Extra questions:

- Rationale: Most of these questions further explore the impacts of choices on how experts spread across the organization.
- How does the "DBA role" work in your organization?
- Is there production problems related to database?
  - Availability, connection between application and database
  - Evolution notes: By the end, we asked less this specific question.
- Are there any security experts in the company?
- How do developers become aware about security concerns?
  - E.g., known vulnerabilities
- For your last project, were vulnerabilities found in production?
  - Were they exploited by attackers?
  - Rationale: "The most potentially embarrassing, controversial, or awkward questions should come toward the end (...) along with reminders of confidentiality as needed" (Adams, 2010).
  - Evolution notes: By the end, we asked less this specific question.
- Microservices: do you use? What do you think?
  - Rationale: This question will probably arise naturally if that is the case. But as it is a theme strongly linked to DevOps and cross-functional teams, it is worth to briefly explore it. But it is good to leave more to the end, so it does not improperly take the time of the interview.
  - Evolution notes: As we rose some hypotheses related to microservices, we strengthened the relevance of this question. However, most of the times we did not need to ask it directly.

Some other additional questions we created after I20 to support the elaboration of rising hypotheses:

- Do you have few automated tests? How do you deal with the tradeoff velocity vs quality?
- How is the collaboration among developers and members of the platform team?
- Do you feel the platform provides more benefits for microservices than for monolithic services?

Ending the interview:

- Ask: "About your technology stack, would you have a favorite tool to recommend?" Quick and easy research-related question to give a final “chill out”. I think in general people are excited to talk about tools.
  - Evolution notes: Due to time, in general this question was not asked.
- Ask: "Would you have any questions for me or a final comment?"
- Ask for suggestions of other people to be interviewed (snowballing).
- Ask to take a picture together: "Can I take a selfie with you?"
- After the interview is over, wait a moment: sometimes it is at this time of relaxation that the interviewee remembers something interesting to add. Selfie time can also help in this regard.
- Thank for the interview.
- Offer a business card.
- At the end of the day, send a “thank you” message.
- After the interview, save the audio with metadata: interviewee, date, and location.

After each interview:

- Analyze what went right and what went wrong in the interview:
  - Have the research questions been answered?
  - Were the guidelines and tips followed?
  - Has any new and exciting topic come up?
  - Raise improvement points for upcoming interviews.
  - If necessary, evolve the interview protocol.
  - Do this exercise mainly for the first interview.
- Take notes of references (books, articles, lectures, etc.) that may have been indicated by the interviewee.

Appendix B

Interview questions for Phase 3

B.1 Research goals

Goal 1 - Answer research questions:

- Why do different software organizations adopt different organizational structures regarding development and infrastructure groups?
- How organizations handle the drawbacks of each organizational structure?

Goal 2 - Refine the taxonomy (adding, removing, and renaming taxonomy high-level elements).

B.2 Semi-structured script (for previously visited organizations)

- Explaining our research: we want to understand the division of labor between development and infrastructure groups in the software industry, and how these groups interact with each other.
- Ask permission to record
- Q0 Context: what do you do? / type of software built by the company.
  - Rationale Q0: Mostly ice-breaker.
- Q1 Concerning the operation activities, such as deployment, infrastructure setup, tracking monitoring, and so on… Do you believe there is a clear expectation of how such tasks should be split or shared between developers and infrastructure people? Is there a clear expectation of how these groups should interact?
  - Rationale Q1: Interviewee starts to think about the organizational structure. Input for the decision point (after Q2).
- Explain how we have classified the interviewee's organization under our taxonomy.

- Q2A What do you think about this classification? Is it intelligible?
- If it is applicable, explain the supplementary properties attributed to the interviewee's organization under our taxonomy.
- Q2B (if pertinent) What do you think about this classification? Is it intelligible?
  - Rationale Q2: It provides input to analysis regarding Goal 2.
- (Decision point) Here, we have two variables:
  - clarity = is there clarity for the interviewee about its structure?
  - match = does the interviewee agree with our description of its structure?
- If there is no clarity and no match, ask questions from phase 2 (to help understanding the structure):
  - Who is responsible for deploying the solution in production?
  - In a new project, who builds the new environment?
  - Who is in charge of non-functional requirements?
  - Who is responsible for configuring monitoring?
  - Who is on-call for incident handling?
  - Rationale (decision point): if there is no clarity and no match, the answers for the next questions may not provide good data.
- Q3 Now, I would like to understand how your company reached this structure:
  - Have you considered other alternatives for structuring devs and ops?
  - Have you ever had other structures?
  - Who had defined the structure?
  - Was it a top-down imposition? From whom?
  - Was it a formal decision?
  - Was it a bottom-up suggestion? From whom?
  - How was the decision made?
  - Were decisions based on references, consulting, or known cases?
  - Was the change incremental or abrupt?
  - Rationale Q3: It is related to Goal 1, since it provides mainly the causes, conditions, and consequences for the organization's structure.
- Briefly present our taxonomy (only structure, not supplementary properties).
- Q4A Why do you think that your organization, in its context, adopted this structure and not some other?

- (If pertinent) Present the supplementary properties related to the organization's structure.
- Q4B (If pertinent) Why do you think that your organization, in its context, adopted these supplementary properties and not others?
  - Rationale Q4: It reinforces the point of the previous question, but also provides data to analysis regarding the Goal 2. I.e.: pushes the interviewee to answer again to the same question, but this time discussing it by using the taxonomy. Q4 is also expected to possibly provide contingencies about other structures (Goal 1).
- Q5 Do you think your organization, in its context, should choose another organizational structure? Why? (ask about the supplementary properties as well).
  - Rationale Q5: It encourages the interviewee to talk about causes, consequences and conditions of some other structure (Goal 1). It also provides data regarding Goal 2.
- Q6 How do you handle the disadvantages of the adopted structure? (ask about the supplementary properties too).
  - Rationale Q6: It is mostly expected to provide the contingencies of the organization's structure (Goal 1).

B.3 Semi-structured script (for new organizations)

- Explaining our research: we want to understand the division of labor between development and infrastructure groups in the software industry, and how these groups interact with each other.
- Ask permission to record
- Q0 Context: what do you do? / type of software built by the company.
  - Rationale Q0: Mostly ice-breaker.
- Q1 Concerning operation activities, such as deployment, infrastructure setup, tracking monitoring, and so on… Do you believe there is a clear expectation of how such tasks should be split or shared between developers and infrastructure people? Is there a clear expectation of how these groups should interact?
  - Rationale Q1: Interviewee starts to think about the organizational structure.
- Briefly present our taxonomy (only structure, not supplementary properties).
- Q2A How would you classify your organization? Why?
- If the reasoning behind the choice does not fit what we could expect, then we discuss more on this.
- Explain pertinent supplementary properties.

- Q2B (if pertinent) How would you classify your organization regarding the supplementary properties?
  - Rationale Q2: It provides input to analysis regarding Goals 2 and 3.
- Q3 Now I would like to understand how your company arrived at this structure:
  - Have you considered other alternatives for structuring devs and ops?
  - Have you ever had other structures?
  - Who had defined the structure?
  - Was it a top-down imposition? From whom?
  - Was it a formal decision?
  - Was it a bottom-up suggestion? From whom?
  - How was the decision made?
  - Were decisions based on references, consulting, known cases?
  - Was the change incremental or abrupt?
  - Rationale Q3: It is related to Goal 1, since it provides mainly the causes, conditions, and consequences for the organization's structure.
- Q4 Why do you think that your organization, in its context, adopted this structure and not some other? (ask about the supplementary properties too).
  - Rationale Q4: It reinforces the point of the previous question, but also provides data to analysis regarding the Goal 2. It is also possibly expected to provide contingencies about other structures (Goal 1).
- Q5 Do you think your organization, in its context, should choose another organizational structure? Why? (ask about the supplementary properties too).
  - Rationale Q5: It encourages the interviewee to talk about causes, consequences and conditions of some other structure (Goal 1). It also provides data regarding Goal 2.
- Q6 How do you handle the disadvantages of the adopted structure? (ask about the supplementary properties too).
  - Rationale Q6: It is expected mostly to provide the contingencies of the organization's structure (Goal 1).

Appendix C

Examples of strong codes concrete implications

C.1 SC04 - Enough infra people to align with dev teams

Code class: condition for collaborating dev & infra departments

The collaborating departments structure is centered on fostering a culture of collaboration. One recommendation that literature gives about promoting such collaboration is inviting infrastructure people to the agile ceremonies (daily, review, and retrospective meetings), so infrastructure people become aware of the project context and feel part of the delivery.

Nonetheless, organizations should be aware that such an initiative is likely to fail if the company has much more developers than infrastructure people. In the beginning, infrastructure people will start to attend the agile ceremonies, but their task queues may not decrease. Instead, ideas in these meetings may generate more workload for them. After some time, infrastructure people think it is better to stop attending the meetings, and things are back as before.

Suppose the company has a low ratio of infrastructure people to developers and, even so, it wishes to try this collaborative approach. In that case, it must be aware that some measure is necessary. For example, giving more autonomy to developers (SC14) so infrastructure people do not become more overloaded than before.

Some supporting statements:

- "There are technically enough SRE teams to align with the same structures that the dev teams have." (I43)
- "When I joined the company, we were more like collaborating dev & infra departments. So, we [infra] were the bottleneck. We had more engineers working in dev teams compared to the infra team. One reason to change our approach was that

people were struggling to deploy into production or to access the infra. So they [devs] were stuck, while we [infra] were firefighting, and we hadn't time to help people [devs]." (I48)

- "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)

C.2 SC28 - Compatible with existing rigid structures (low impact on organogram) / Only a few people needed to form a platform team

Code class: causes of API-mediated dev & infra departments

Some practitioners advocate that their organizations should adopt single departments to eliminate hand-offs and bottlenecks. However, in a big company with a centralized pool of infrastructure professionals, it is hard to break the hierarchy and scatter infrastructure people across different departments. When performing such a transformation, many people will lose their positions as bosses, which may lead to friction.

On the other hand, assembling an experimental platform team can be cheaper and bring less resistance. The platform team can be formed as a group within operations, possibly bringing some developers to it. It will start as a non-compulsory platform, and if it has no success, people can more easily go back to their groups of origin. If the platform achieves some initial success, the platform team can grow little by little, considering that adding more services or clients to the platform does not require proportionally more infra people within the platform team (SC38). In this way, hierarchies are affected at a slow pace.

Some supporting statements:

- "[In a company adopting API-mediated departments] Cross-functional teams not adequate for the company structure having a development and an operations department (devs and ops in totally different structures)." (I40)

- "You don't have managers willing to give up their main employees and you don't have the possibility to hire people. 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. There are about 15 people on the platform, it's not enough to cross-functional." (I41)

- "It is the best model [platform team] for the size of the company. I don't have enough people to put one [infra] person on each team. I think this allows us to have a leaner platform team." (I44)

C.3 SC19 - Not suitable for applying corporate governance and standards

Code class: avoidance reason for single dev & infra departments

With the infrastructure staff scattered across different departments, several managers are concerned with different teams taking different approaches to infrastructure management. Organizations considering migrating to single departments should be aware of this concern.

Enforcing corporate standards may require more effort in single departments when compared to the scenario with a centralized infrastructure group. Therefore, top managers must evaluate: if enforcing standards is really necessary, are single departments worth it? If the organization enforces corporate standards over single departments, the company must plan it effectively, considering the need for innovation. Having a homogeneous infrastructure is really necessary?

The principal claim of managers to require homogeneous infrastructure is the worry with the on-boarding time when professionals move from one team to another. Such justification could trigger relevant discussions in practitioners' forums: is this worry valid? Do employees indeed switch teams so often? Moreover, do not employees frequently move from company to company nowadays? If so, maybe professionals must be prepared to adapt to new technology anyway. Researchers could also take this quandary and start new research around it.

Some supporting statements:

- "So there are advantages in standardizing things. I think isolated super-independent teams tend to get a bit messy. [about standardization] Both infra and programming language standardization, so things don't get out of control, and people do not use exotic databases... then nobody knows how to maintain it. This has certainly been a concern." (I54)

- "To accelerate you need a certain standard. It's no use talking 'ah, everyone does what they want, everyone puts in production.' I find it difficult for a large institution, with a reputation to uphold, to work in this way. It won't be like that..." (I55)

- "They are afraid of being too open and start to lose control over the best practices and care with security, availability, attacks... Those [infra] people would lose their roots and become less connected to their area of expertise." (I68)

C.4 SC11 - Discomfort/frustration/friction/inefficiency with blurred responsibilities (people don’t know what to do or what to expect from others)

Code class: consequence of collaborating dev & infra departments

The collaborating departments structure is centered on fostering a culture of collaboration. The idea is that development and infrastructure professionals will collaborate to perform operational activities.

However, sometimes problems arise because of unclear responsibilities over some tasks. We saw cases in which a group had expectations about what the other group would do and, in the end, such expectations were not fulfilled.

Companies wishing to foster a culture of collaboration must consider this issue. Maybe imposing rigid rules on responsibilities can hind the collaboration. So, a possible solution is aligning different groups with a common goal, so people become more focused on "what can I do so we achieve the goal" than on "what is my responsibility?". Nevertheless, even in this case, care is needed. Sometimes people may feel wronged when the goal is not achieved and their groups did everything that was needed, especially when the achievement of the goal is linked to some reward.

Some supporting statements:

- "A team that experimented Classical DevOps: the team split into two, a dev team and another infra team. They ended up giving up and going back. Problem: if there is a problem, it may not be clear who owns the problem – and this creates inefficiency. This division hurts ’ownership’." (I38)
- "The groups have the expectations about other groups, but the expectation is not always fulfilled." (I51)
- "This division of responsibility is not very clear. In an ideal world, individual teams own the entire infrastructure, teams are super independent to do whatever they need to do, you don’t need to depend on the infra team. But there is not this super clear division of you do this, I do that." (I54)

C.5 SC46 Decide how much devs must be exposed to the infra internals (some places more, some places less)

Code class: contingency for API-mediated dev & infra departments

The goal of a platform is to abstract the infrastructure so that developers can effortlessly operate their services. So, in principle, the higher the abstraction, the better.

However, we observed at least two problems with too abstract platforms. First, developers expect too much from the platform. Sometimes the service is not running because of a coding or configuration issue, but developers unduly contact the platform team when they could have just read the log file and figured out the problem. The second problem is that to increase the platform abstraction, the platform team becomes more overloaded, sometimes adding features that could be avoided with developers writing one line of configuration.

Therefore, an organization with a platform must carefully discuss the level of abstraction of its platform. We saw at least one case of a company in which this debate was promoted and became highly disputed. Possibly, the organization will have to consider the skill level of its developers and also how it foresees these developers should evolve to become better developers; i.e., if becoming a better developer encompasses the notion of improving operational skills, which will contribute to services reliability, or if the focus is just "release soon".

Some supporting statements:

- "Deployment is magic, you click on a button and the application appears there. If I click on the button and got an error, I open a ticket. But why your deploy is not working? Do you understand how your pipeline works? Do you understand what settings you need to apply? Here at the company, we are considering a lot how far we want to automate and how far we want to expose people to the complexity of infrastructure." (I52)
- "The biggest discussion around here was about to what extent devs have to understand the infra. That was a really big discussion. I was one of the guys who radically thought devs should know some minimum. But the guys who were tech leads on my team thought differently, that devs didn't have to know anything, didn't even have to know Docker." (I53)
- "I think a disadvantage would be that you don't give the dev a vision of what's happening. One way we think about it is to give the dev freedom to know how we provide for it so that it can contribute to this model." (I57)

Appendix D

Answers for the open questions of our feedback forms

D.1 Phase 2 feedback

Question: “Especially in case of disagreement [of our proposed classification of structure], please, explain why you disagree. Would you classify it differently?”

Answers:

- I think the platform team also implies that the product teams are doing devops but have no distinction of that role inside the product team. Inside the product team it’s system and sw engineering using tools and services that abstract away complexity provided by the platform teams where we also have systems and sw engineering... i think the picture didn’t work for me since it implies platform only does systems engineering and product only sw engineering.

Question: “Especially in case of disagreement [of our proposed classification of supplementary property], please, explain why you disagree. ”

No answers for this question.

Question: “Would you add more DevOps organization structures [or supplementary properties] to our model? Which Ones?”

Answers:

- We have a third pillar in the Operations/Development duo which is Observability. A team that collects metrics and data for everything. Both the platform and the applications and can correlate those to provide better insight in incidents.
- No.

Question: “Would you have any other thoughts or criticism regarding the proposed model?”

Answers:

- No. I agree with the proposed model.
- You're missing all the "automated" operations. Systems that handle rolling deploys and other similar "in production but under (automated) scrutiny". Those, today, are still not properly owned/viewed. Some being provided by the platform, some defined by the development teams and some just completely separate.
- Nice job, can I share the model publicly?
- I agree with the proposed model and I think specially in large companies on different projects some times the teams/departments can work on slightly different models from the main company model.
- It's a pretty good characterization of X and Y and Z[1].
- Thanks for forwarding on your theory. I guess I feel that we have historically been mostly siloed however occasionally we fit into the more collaborative cross-functional team as the project requires it. Additionally I would say we are naturally evolving more towards the platform model with the deployment of aaS or self-service platforms.

D.2 Phase 3 feedback

Question: "Would you have any thoughts on the matter?"

Answers about **siloed departments**:

- Bureaucracy between segregated departments should be minimized. The alternatives are how to better interact among the team and evaluate the communication/operation.
- I believe that the low performance of deliveries is affected by the silos, but it's not the only factor.
- Similarly to automated platforms, when the infra folks treat the devs as their customers, even with the manual interactions necessary that doubtlessly increase friction and time to resolution, the relationship is more productive as the infra team tends to optimize for the common needs of the devs.
- It may be that I didn't understand the questions but in my experience none of these problems are directly related to the separation of departments. In my experience, this reduces bureaucracy, increases the autonomy and efficiency of the processes.
- In my previous company, those assertions were mostly true for environments fully managed by the ops team, on others where continuous delivery was set up, the dev teams was autonomous enough and there were less friction.

[1] X, Y, and Z are big techies at which the interviewee has already worked.

- I think that one of the reason the devs lack autonomy is because lack of knowledge about infra by the devs and this cause a lack of autonomy and freedom to do things like pipelines.

Answers about **collaborating departments**:

- Sounds like your research overlaps with the Teams Topologies "types" and google's "engagement models". I'm sure you've seen these already but just in case, here's some links: https://web.devopstopologies.com/ and https://cloud.google.com/blog/products/devops-sre/how-sre-teams-are-organized-and-how-to-get-started
- Collaboration breaks down when the infra team (the one providing the platform) is understaffed and is incapable of fulfilling the needs of the dev teams. The real true contingency for this one is to automate as much as possible on the infra land as time goes by (usually moving towards an API-mediated situation) because, end of the day, if the company is successful, the ratio between devs and infra team is only going to grow.
- Collaborative areas tend to reach their goals faster, as they have a better understanding of the whole and a better distribution of knowledge for problem solving.
- On a regular day of an infra worker and a dev worker there's almost no time to get together and share knowledge between these two areas. In my opinion a top-to-bottom decision making approach is the most viable way to implement organizational structures.
- Another contingency is the constant alignment of expectations between the teams, as it is very common for one of the teams to act on problems at the wrong time and this creates a great friction that can affect the entire organization.
- I didn't understand the meaning of contingencies in the last question (even looking in the dictionary).

Answers about **API-mediated departments**:

- One of the key differentiators in my experience for API-mediated departments to provide a good platform is whether they consider the dev teams to be their customers the same way the dev teams consider the company customers to be their customers. When that mindset is present, the interactions from the platform providing team tend to be much more driven to identify the right balance between hiding complexity and providing transparency therefore mitigating a lot of the risks/consequences you described.
- "API-mediated platform teams are a double-edged sword: on one hand, they abstract the infrastructure from developers in user-facing teams, helping them to become more agile as they can delegate their infrastructure needs to a third party; on the other hand, they abstract the infrastructure from developers in user-facing teams, alienating them from implementation details that can affect their products. As any software engineering decision, adopting API-mediated infrastructure teams is a decision with many trade-offs. It helps companies to increase the impact of

cloud-based infrastructure while reducing the number of people required for the infrastructure role. Conversely, it can create silos of knowledge and a "me against you" if the company does not provide its developers with the right amount of infrastructure knowledge. For me, finding this equilibrium is the greatest challenge of applying this kind of organizational pattern."

- A lot "depends" upon the situation so I often answered "No correlation / I don't know". The contingency can be dependent on the system, the team, and the organization.

Answers about **single department**:

- I do not necessarily attribute single departments with lack of standardization and collaboration with dev teams. I believe you can have multiple "single dev + infra" teams that can collaborate. In our company, we have two separate dev teams, each with their own infrastructure engineers. But we hold biweekly "future of software development" meetings with the heads of the dev departments to map common infrastructures and establish best practices. Once those are established, they are documented for future teams and usage. The idea is that over time, we will establish all the standards and guidelines, and inform all future devs of existing initiatives they can take advantage of and avoid duplication. This has proven to be a successful approach so far, and I do believe it scales up, as long representatives have to join. If the company was much larger, these could be held at less frequent paces (looking more like a conference than a meeting), with intermediary needs for sync being fulfilled by a head of software (Director/VP)
- Consequence of adopting a single structure: difficulty in diagnosing problems; high recovery time in case of serious infrastructural failures.
- Based on the link at the top of this questionnaire it seems you're calling "single dev/infra" both teams were infra responsibilities are shared across all team members (no specialization) and teams were infra specialists are embedded in the team. I personally think those are completely different models. IMO, the single team with embedded infra professional is closer to the "Collaborating dev & infra departments" model than to "single team". The key for single team to work (and thus for collecting its benefits) is exactly to enable the "you built it, you run it" part, and it seems to me the single team with specialized infra resources is closer to "devs on the team built it, infra specialists on the team run it"
- To establish the devs and ops in a single team, maintaining standards and governance, it is necessary to be able to keep the team stable (without many changes of employees) and to have well-documented common processes between teams.
- This one presents the same problem as all the other embedded functions in a multi-functional team structure: lack of tie between the experts. The most common proposed contigency is guilds or whatever other name representing an informal (in the sense of not represented by the organization structure reporting tree) gathering of experts to define the direction that specialty will head towards. Having an expert in a team, so long as the expert collaborates with other team members, tends to solve the problem of developing in-house skills (to varying levels) on that team without requiring formal training.

- Criticizing the format of the questions, I think they were formulated a little strangely, perhaps as a suggestion I think the options could be (Agree, somewhat/partially agree, neither agree nor disagree, partially disagree, disagree) it would give more the idea of opinion and not so much true or false questions (as in a test). Second, I found strange the way the questions were prepared, for example in the second question ("Startup scenario [...]" is a cause of single departments) a translation would be: "In a startup scenario (basically chaotic) is a cause of single departments" in my opinion it doesn't make much sense; maybe something that would make more sense to formulate (to select alternatives that people agree or disagree with) would be: "A startup scenario tends to use single departments" (A startup environment leans towards single departments). Another example would be in the 6th question ("More costs: duplication of infra work among teams, high infra professionals salary, underused infra professionals" is an avoidance reason for single departments), in my opinion there is 1 point that tends in favor of single departments and 2 against, in the case of "duplication of infrastructure work between teams" is a point in favor of having a dedicated infrastructure department. The wording of the phrase " "more costs: ..." is an avoidance reason for ..." also sounds a little strange to me, a translation would be " "Higher costs: ..." are reasons for evasion of single departments", perhaps another suggestion to formulate the sentence would be "More costs: ... are common reasons/excuses to use (a structure of) single departments" (More costs: ..., are common reasons to apply a single department structure). About devops teams, I have had experience in 2 companies so far, the first had a tech team of 6 people, which naturally required much less cloud operations (and tech in general) which ended up tending to a single department model, and the second had a dedicated infra team from the beginning (when we were only 6 devs in total, 1 of them was infra focused already). Both were startups. I think that most of the statements/questions in the form can make sense when maybe there are less technical people behind the management or because it doesn't make much sense for the business model they are doing/elaborating. On the other hand, any company with a greater focus on developing digital products, I really believe that it strongly needs an infrastructure department. Infrastructure monitoring and management alone already demands a lot from a team and having experts on the subject helps a lot to reduce costs, have more security, speed up / facilitate everyone's life, stability, among many other benefits.

- As the cloud services offering get more mature it becomes leases important the necessity to build and maintain our own dedicated infra team allowing teams to focus on the product instead of infra with less friction.

- Currently in my organization people are discussing the "dedicated department" existence not only on infrastructure context, but also regarding design matters. Arguments like "hanging on to designers/ops professionals availability to move things forward is slowing down deliveries - devs could do most of simple things themselves" were raised. My contribution on the matter was that sharing these resposabilities with the dev team (being it ops or design tasks) strongly depends on culture and documentation that draw a very clear line of where devs should stop, step back and let more seasoned professionals take the lead. In our case, we do not have devs who are mature enough for single department strucutre (devs write

code AND manage ops). From this perspective, the way is still to have a dedicated department for infrastructure, as it is today.

- In my experience, platforms with Heroku end up being more expensive than using AWS directly, but the cost benefit was well worth it for a company that doesn't have an application that demands a lot of infrastructure resources. The price difference is not that big, it reduces maintenance a lot (since the service provides several) and greatly reduces the "skill-cap". I, for example, am not a very skilled person in infrastructure, but with a small team we were able to scale the application well using Heroku and other connected PaaS.
- I answered false in the first one because a team can depend on another one that has a more experienced person and/or with more privileged access to the CI/CD systems.

References

[Ada10] William C. Adams. “Chapter 16: Conducting semi-structured interviews”. In: *Handbook of Practical Program Evaluation*. 3rd. Jossey-Bass, 2010 (cit. on pp. 35, 39).

[AH09] John Allspaw and Paul Hammond. *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr*. At Velocity 2009. Slides available on https://pt.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr, accessed in Nov 2021. 2009 (cit. on p. 23).

[And83] Paul A. Anderson. “Decision Making by Objection and the Cuban Missile Crisis”. In: *Administrative Science Quarterly* 28.2 (1983), pp. 201–222 (cit. on pp. 8, 39, 83).

[BA04] Kent Beck and Cynthia Andres. *Extreme Programming explained: embrace change*. Addison-Wesley Professional, 2004 (cit. on pp. 12, 19).

[Bai+18] Xiaoying Bai et al. “Continuous Delivery of Personalized Assessment and Feedback in Agile Software Engineering Projects”. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ICSE-SEET ’18. 2018, pp. 58–67 (cit. on p. 18).

[Ban+99] Mousumi Banerjee et al. “Beyond kappa: A review of interrater agreement measures”. In: *Canadian Journal of Statistics* 27.1 (1999), pp. 3–23 (cit. on p. 87).

[Bas+16] Ali Basiri et al. “Chaos Engineering”. In: *IEEE Software* 33.3 (2016), pp. 35–41 (cit. on p. 14).

[BB06] David Budgen and Pearl Brereton. “Performing Systematic Literature Reviews in Software Engineering”. In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE ’06. ACM, 2006, pp. 1051–1052 (cit. on p. 3).

[Bey+16] Betsy Beyer et al. *Site Reliability Engineering: How Google runs production systems*. O’Reilly, 2016 (cit. on pp. 14, 16, 20, 52, 63).

[BHJ16] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture”. In: *IEEE Software* 33.3 (2016), pp. 42–52 (cit. on pp. 14, 18, 19, 91).

[BR20] Sebastian Baltes and Paul Ralph. “Sampling in software engineering research: A critical review and guidelines”. In: *arXiv preprint arXiv:2002.07764* (2020) (cit. on p. 68).

[Bro18] Donovan Brown. *Our DevOps journey - Microsoft’s internal transformation story*. DevOneConf 2018, https://www.youtube.com/watch?v=cbFzojQOjyA, accessed in Jul 2018. 2018 (cit. on p. 18).

[BSK20] Alanna Brown, Michael Stahnke, and Nigel Kersten. *2020 State of DevOps Report*. https://www2.circleci.com/2020-state-of-devops-report.html, accessed in Dec 2021. 2020 (cit. on p. 27).

[Cha08] Kathy Charmaz. "Chapter 7: Grounded Theory as an emergent method". In: *Handbook of Emergent Methods*. Ed. by Sharlene Nagy Hesse-Biber and Patricia Leavy. The Guilford Press, 2008 (cit. on p. 31).

[Che15] Lianping Chen. "Continuous delivery: Huge benefits, but challenges too". In: *IEEE Software* 32.2 (2015), pp. 50–54 (cit. on pp. 1, 19, 20).

[Chr16] Henrik Bærbak Christensen. "Teaching DevOps and Cloud Computing Using a Cognitive Apprenticeship and Story-Telling Approach". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '16. ACM, 2016, pp. 174–179 (cit. on pp. 12, 14, 18).

[CK18] Daniel Cukier and Fabio Kon. "A maturity model for software startup ecosystems". In: *Journal of Innovation and Entrepreneurship* 7 (2018) (cit. on pp. 32, 35).

[Con68] Melvin E. Conway. "How do committees invent". In: *Datamation* 14.4 (1968), pp. 28–31 (cit. on p. 22).

[CSA15] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. "On the journey to continuous deployment: Technical and social challenges along the way". In: *Information and Software Technology* 57 (2015), pp. 21–31 (cit. on p. 19).

[Cuk13] Daniel Cukier. "DevOps Patterns to Scale Web Applications Using Cloud Services". In: *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity*. SPLASH '13. ACM, 2013, pp. 143–152 (cit. on pp. 19, 20).

[Cun92] Ward Cunningham. "The WyCash Portfolio Management System". In: *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA '92. ACM, 1992, pp. 29–30 (cit. on p. 90).

[Dd14] Dimitra Dodou and Joost C.F. de Winter. "Social desirability is the same in offline, online, and paper surveys: A meta-analysis". In: *Computers in Human Behavior* 36 (2014), pp. 487–495 (cit. on p. 87).

[DD16] Jennifer Davis and Ryn Daniels. *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. O'Reilly Media, 2016 (cit. on pp. 2, 26, 50).

[Deb08] Patrick Debois. *Agile Infrastructure & Operations*. At Agile 2008 Toronto. Slides available on https://docplayer.net/23874035-Agile-infrastructure-operations.html, accessed in Nov 2021. 2008 (cit. on pp. 13, 48).

[Deb11] Patrick Debois. "Devops: A software revolution in the making". In: *Cutter IT Journal* 24.8 (2011), pp. 3–5 (cit. on p. 19).

[DG94] D. Harold Doty and William H. Glick. "Typologies As a Unique Form Of Theory Building: Toward Improved Understanding and Modeling". In: *Academy of Management Review* 19.2 (1994), pp. 230–251 (cit. on pp. 29, 87).

[DMC16] Elisa Diel, Sabrina Marczak, and Daniela S. Cruzes. "Communication Challenges and Strategies in Distributed DevOps". In: *11th IEEE International Conference on Global Software Engineering (ICGSE)*. 2016, pp. 24–28 (cit. on p. 19).

[Don93] Ann Donnellon. “Crossfunctional teams in product development: Accomodating the structure to the process”. In: *Journal of Product Innovation Management* 10.5 (1993), pp. 377–392 (cit. on p. 26).

[Don99] Lex Donaldson. “Teoria da contingência estrutural”. In: *Handbook de estudos organizacionais*. Atlas, 1999 (cit. on p. 22).

[DPL15] Andrej Dyck, Ralf Penners, and Horst Lichter. “Towards Definitions for Release Engineering and DevOps”. In: *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. 2015, pp. 3–3 (cit. on p. 15).

[EAD17] Floris Erich, Chintan Amrit, and Maya Daneva. “A qualitative study of DevOps usage in practice”. In: *Journal of Software: Evolution and Process* 29.6 (2017) (cit. on p. 27).

[Ebe+16] Christof Ebert et al. “DevOps”. In: *IEEE Software* 33.3 (2016), pp. 94–100 (cit. on pp. 18, 91).

[Eis89] Kathleen Eisenhardt. “Building Theories from Case Study Research”. In: *Academy of Management Review* 14.4 (1989), pp. 532–550 (cit. on p. 83).

[Fay13] Henri Fayol. *General and industrial management*. Reprint of 1949 Edition; originally published in French in 1916. Martino Fine Books, 2013 (cit. on p. 23).

[FFB13] Dror G. Feitelson, Eitan Frachtenberg, and Kent L. Beck. “Development and deployment at Facebook”. In: *IEEE Internet Computing* 17.4 (2013), pp. 8–17 (cit. on p. 19).

[FHK18] Nicole Forsgren, Jez Humble, and Gene Kim. “Chapter 2: Measuring Performance”. In: *Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations*. IT Revolution Press, 2018 (cit. on pp. 21, 25, 87).

[FJT16] Breno B. Nicolau de França, Helvio Jeronimo Junior, and Guilherme Horta Travassos. “Characterizing DevOps by Hearing Multiple Voices”. In: *Proceedings of the 30th Brazilian Symposium on Software Engineering*. SBES ’16. ACM, 2016, pp. 53–62 (cit. on pp. 8, 31).

[FK18] Nicole Forsgren and Mik Kersten. “DevOps Metrics”. In: *Communications of the ACM* 61.4 (2018), pp. 44–48 (cit. on p. 18).

[For+17] Nicole Forsgren et al. *2017 State of DevOps Report*. https://puppet.com/resources/whitepaper/2017-state-of-devops-report, accessed in Dec 2020. 2017 (cit. on p. 87).

[For+20] Nicole Forsgren et al. “A Taxonomy of Software Delivery Performance Profiles: Investigating the Effects of DevOps Practices”. In: *Proceedings of The Americas Conference on Information Systems 2020*. AMCIS 2020. AIS, 2020, pp. 1–6 (cit. on pp. 20, 27, 87).

[Fou19] ITIL Foundation. *ITIL 4^{th} edition, Glossary*. https://purplegriffon.com/downloads/resources/itil4-foundation-glossary-january-2019.pdf, accessed in Nov 2021. 2019 (cit. on p. 16).

[Ful09] Jeffrey Fulmer. “What in the world is infrastructure”. In: *PEI Infrastructure Investor* 1.4 (2009), pp. 30–32 (cit. on p. 16).

[GA08] Svetla Georgieva and George Allan. “Best Practices in Project Management Through a Grounded Theory Lens”. In: *The Electronic Journal of Business Research Methods* 6.1 (2008), pp. 43–52 (cit. on p. 36).

[Gal08] Jay R. Galbraith. “Chapter 18: Organization Design”. In: *Handbook of Organization Development*. Ed. by Thomas G. Cummins. Sage Publications, 2008, pp. 325–352 (cit. on p. 23).

[GC14] Eliyahu M. Goldratt and Jeff Cox. *The Goal: A Process of Ongoing Improvement*. 30th anniversary edition. North River Press, 2014 (cit. on pp. 22, 36, 48).

[Gla02] Barney G. Glaser. “Conceptualization: On Theory and Theorizing Using Grounded Theory”. In: *International Journal of Qualitative Methods* 1.2 (2002), pp. 23–38 (cit. on p. 78).

[Gla05] Barney Glaser. *The grounded theory perspective III: Theoretical coding*. pp. 70. The Sociology Press, 2005 (cit. on p. 40).

[Gla78] Barney Glaser. *Theoretical sensitivity: Advances in the methodology of Grounded Theory*. The Sociology Press, 1978 (cit. on pp. 40, 77).

[Gra06] Jim Gray. “A conversation with Werner Vogels”. In: *ACM Queue* 4.4 (2006), pp. 14–22 (cit. on pp. 2, 18, 19, 23, 53).

[Gre06] Shirley Gregor. “The nature of theory in information systems”. In: *MIS quarterly* (2006), pp. 611–642 (cit. on pp. 4, 29, 30, 40).

[GS99] Barney Glaser and Anselm Strauss. *The discovery of grounded theory: strategies for qualitative research*. Originally published in 1967. Aldine Transaction, 1999 (cit. on pp. 4, 27, 29–32, 39, 41, 42, 44, 45, 68, 69, 73, 83, 87).

[Gub81] Egon G. Guba. “Criteria for assessing the trustworthiness of naturalistic inquiries”. In: *Educational Technology Research and Development (ECTJ)* 29 (1981), pp. 75–91 (cit. on pp. 39, 46, 83, 84).

[Gue91] David Guest. *The hunt is on for the Renaissance Man of computing*. The Independent (London). Sept. 1991 (cit. on p. 19).

[Hal84] Richard H. Hall. *Organizações, estruturas e processo*. Prentice-Hall, 1984 (cit. on p. 22).

[Ham07] James Hamilton. “On Designing and Deploying Internet-Scale Services”. In: *Proceedings of the 21st Large Installation System Administration Conference*. LISA ’07. USENIX, 2007, pp. 231–242 (cit. on p. 19).

[HCM17] Waqar Hussain, Tony Clear, and Stephen MacDonell. “Emerging Trends for Global DevOps: A New Zealand Perspective”. In: *Proceedings of the 12th International Conference on Global Software Engineering*. ICGSE ’17. IEEE Press, 2017, pp. 21–30 (cit. on pp. 15, 19).

[Her09] Cheri Ann Hernandez. “Theoretical coding in Grounded Theory methodology”. In: *Grounded Theory Review* 8.3 (2009) (cit. on p. 40).

[HF10] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010 (cit. on pp. 12, 13).

[HM11] Jez Humble and Joanne Molesky. “Why enterprises must adopt DevOps to enable continuous delivery”. In: *Cutter IT Journal* 24.8 (2011), pp. 6–12 (cit. on pp. 1, 8, 15, 18, 20, 61, 63).

[HN17] Rashina Hoda and James 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 (cit. on pp. 31, 35).

[HNM11] Rashina Hoda, James Noble, and Stuart Marshall. "The impact of inadequate customer collaboration on self-organizing agile teams". In: *Information and software technology* 53.5 (2011), pp. 521–534 (cit. on pp. 40, 70, 77).

[Hod21] Rashina Hoda. "Socio-Technical Grounded Theory for Software Engineering". In: *IEEE Transactions on Software Engineering* (2021). Early access, pp. 1–1 (cit. on pp. 31, 32).

[HR06] James Herbsleb and Jeff Roberts. "Collaboration In Software Engineering Projects: A Theory Of Coordination". In: *International Conference on Information Systems 2006 Proceedings*. ICIS 2006. 2006, pp. 553–568 (cit. on p. 26).

[HS05] Hsiu-Fang Hsieh and Sarah E. Shannon. "Three approaches to qualitative content analysis". In: *Qualitative health research* 15.9 (2005), pp. 1277–1288 (cit. on p. 39).

[Hum10] Jez Humble. *Continuous Delivery vs Continuous Deployment*. https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/, accessed in Aug 2019. 2010 (cit. on p. 14).

[Hum12] Jez Humble. *There's No Such Thing as a "Devops Team"*. https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team, accessed in Aug 2018. 2012 (cit. on pp. 20, 50, 52).

[Jaa18] Martin Gilje Jaatun. "Software Security Activities that Support Incident Management in Secure DevOps". In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ARES 2018. ACM, 2018, 8:1–8:6 (cit. on p. 18).

[Jes92] Bob Jessop. "Chapter 3: Fordism and post-Fordism: a critical reformulation". In: *Pathways to industrialization and regional development*. Routledge, 1992, pp. 54–74 (cit. on p. 22).

[Jos+15] Ankur Joshi et al. "Likert scale: Explored and explained". In: *British journal of applied science & technology* 7.4 (2015), pp. 396–403 (cit. on pp. 60, 79).

[Jov+17] Miloš Jovanović et al. "Transition of organizational roles in Agile transformation process: A Grounded Theory approach". In: *Journal of Systems and Software* 133 (2017), pp. 174–194 (cit. on pp. 31, 40, 83).

[KBS18] Gene Kim, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. 3rd ed. IT Revolution Press, 2018 (cit. on pp. 15, 36, 48).

[KC07] Barbara Kitchenham and Stuart Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Keele Universit, University of Durham, 2007 (cit. on p. 3).

[KC12] John P. Kotter and Dan S. Cohen. *The heart of change: real-life stories of how people change their organizations*. Harvard Business Review Press, 2012 (cit. on pp. 24, 25).

[Kim+16] Gene Kim et al. "Chapter 7: How to design our organization and architecture with Conway's law in mind". In: *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations*. IT Revolution Press, 2016 (cit. on pp. 26, 61–63).

[KK03] Per Kroll and Philippe Kruchten. *The Rational Unified Process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003 (cit. on p. 12).

[Kni14] Henrik Kniberg. *Spotify engineering culture (part 1)*. https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1, accessed in Aug 2019. 2014 (cit. on pp. 19, 52, 53).

[Knu74] Donald E. Knuth. "Computer programming as an art". In: *Communications of the ACM* 17.12 (1974), pp. 667–673 (cit. on p. 90).

[Lei+13] Leonardo Leite et al. "A systematic literature review of service choreography adaptation". In: *Service Oriented Computing and Applications* 7.3 (2013), pp. 199–216 (cit. on p. 3).

[Lei+19] Leonardo Leite et al. "A Survey of DevOps Concepts and Challenges". In: *ACM Computing Surveys* 52.6 (2019), 127:1–127:35 (cit. on pp. 3, 5, 11, 15, 32, 35, 36, 50, 81).

[Lei+20a] Lenardo Leite et al. "Platform Teams: An Organizational Structure for Continuous Delivery". In: *IEEE/ACM 42nd International Conference on Software Engineering Workshops*. ICSEW'20. 2020, pp. 505–511 (cit. on p. 6).

[Lei+20b] Leonardo Leite et al. "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 (cit. on p. 6).

[Lei+21] Leonardo Leite et al. "The organization of software teams in the quest for continuous delivery: A Grounded Theory approach". In: *Information and Software Technology* 139 (2021), p. 106672 (cit. on pp. 7, 46, 70).

[Lew45] Kurt Lewin. "The Research Center for Group Dynamics at Massachusetts Institute of Technology". In: *Sociometry* 8.2 (1945), pp. 126–136 (cit. on p. 30).

[LKM20] Leonardo Leite, Fabio Kon, and Paulo Meirelles. "Understanding context and forces for choosing organizational structures for continuous delivery". In: *10th CBSoft's Workshop on Theses and Dissertations*. WTDSoft 2020. 2020 (cit. on p. 7).

[Lop+21] Daniel Lopez-Fernandez et al. "DevOps Team Structures: Characterization and Implications". In: *IEEE Transactions on Software Engineering* (2021) (cit. on pp. 2, 8, 26, 27, 31, 64, 83).

[LPB19] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study". In: *Journal of Systems and Software* 157 (2019), p. 110384 (cit. on pp. 31, 50).

[Lun12] Fred C. Lunenburg. "Organizational structure: Mintzberg's framework". In: *International journal of scholarly, academic, intellectual diversity* 14.1 (2012) (cit. on p. 24).

[Man+19] Andi Mann et al. *2019 State of DevOps Report*. https://www2.circleci.com/2019-state-of-devops-report.html, accessed in Mar 2022. 2019 (cit. on p. 92).

[Mar20] Karl Marx. "Preface". In: *A contribution to the critique of political economy*. Originally published in German in 1859. Lector House, 2020 (cit. on p. 25).

[MB20] Ruth W. Macarthy and Julian 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 (cit. on pp. 2, 26, 31).

[MBK18] Andi Mann, Michael Stahnke Alanna Brown, and Nigel Kersten. *2018 State of DevOps Report*. https://puppet.com/resources/whitepaper/2018-state-of-devops-report, accessed in Jul 2019. 2018 (cit. on pp. 26, 61–63).

[McK82] Bill McKelvey. *Organizational systematics - taxonomy, evolution, classification*. University of California Press, 1982 (cit. on p. 29).

[Mel15] Claudia Melo. "Productivity of agile teams: an empirical evaluation of factors and monitoring processes". PhD thesis. University of São Paulo, 2015 (cit. on p. 84).

[MH13] Matthew Miles and Micahel Huberman. "Chapter 2: Focusing and Bounding the Collection of Data - the Substantive Start". In: *Qualitative Data Analysis: A Methods Sourcebook*. 3rd. Sage Publications, 2013 (cit. on p. 37).

[Mor11] Gareth Morgan. "Reflections on Images of Organization and Its Implications for Organization and Environment". In: *Organization & Environment* 24.4 (2011), pp. 459–478 (cit. on p. 24).

[Mor16] Kief Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016 (cit. on p. 56).

[MR04] Mary Lynn Manns and Linda Rising. *Fearless Change: Patterns for Introducing New Ideas*. Addison Wesley, 2004 (cit. on pp. 24, 25).

[MS93] James March and Herbert Simon. *Organizations*. 2nd. Wiley Blackwell, 1993 (cit. on p. 25).

[NMB08] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. "The influence of organizational structure on software quality". In: *2008 ACM/IEEE 30th International Conference on Software Engineering*. 2008, pp. 521–530 (cit. on p. 26).

[NSP16] Kristian Nybom, Jens Smeds, and Ivan 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 (cit. on pp. 1, 2, 18–20, 26, 58, 61–63).

[OAB12] Helena H. Olsson, Hiva Alahyari, and Jan 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 (cit. on p. 91).

[Ohn88] Taiichi Ohno. *Toyota Production System: Beyond Large-Scale Production*. Originally published in Japanese in 1978. Productivity Press, 1988 (cit. on p. 23).

[Oli12] Nelio Oliveira. "Chapter 2: Organizational structure, format, shape design and architecture". In: *Automated organizations: Development and structure of the modern business firm*. Physica-Verlag HD, 2012 (cit. on pp. 2, 22).

[Pat14] Michael Quinn Patton. *Qualitative research & evaluation methods: Integrating theory and practice*. Sage publications, 2014 (cit. on p. 68).

[Pen+18] Birgit Penzenstadler et al. "Software Engineering for Sustainability: Find the Leverage Points!" In: *IEEE Software* 35.4 (2018), pp. 22–33 (cit. on p. 24).

[Pfl14] Niels Pflaeging. *Organize for complexity: how to get life back into work to build the high-performance organization*. 5th. Betacodex Publishing, 2014 (cit. on pp. 25, 26).

[Pin86] Lawrence T. Pinfield. "A Field Evaluation of Perspectives on Organizational Decision Making". In: *Administrative Science Quarterly* 31.3 (1986), pp. 365–388 (cit. on pp. 8, 39, 83).

[PP06] Mary Poppendieck and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006 (cit. on p. 24).

[Pre05] Roger S. Pressman. *Software engineering: a practitioner's approach*. 6th. Palgrave Macmillan, 2005 (cit. on p. 12).

[Rad09] Hans Radder. "The philosophy of scientific experimentation: a review". In: *Automated Experimentation* 1.1 (2009) (cit. on p. 30).

[Ral19] Paul Ralph. "Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering". In: *IEEE Transactions on Software Engineering* 45.7 (2019), pp. 712–735 (cit. on pp. 26, 27, 29, 31, 39, 46, 59, 77, 83, 87, 88).

[Roc13] James Roche. "Adopting DevOps Practices in Quality Assurance". In: *Communications of the ACM* 56.11 (2013), pp. 38–43 (cit. on p. 14).

[Rot09] Mike Rother. *Toyota kata: managing people for improvement, adaptiveness and superior results*. McGraw-Hill, 2009 (cit. on pp. 24, 88).

[Rou15] Margaret Rouse. *What is NoOps? – Definition from WhatIs.com*. https://searchcloudapplications.techtarget.com/definition/noops, accessed in Mar 2022. 2015 (cit. on p. 20).

[Sab96] Karl Sabbagh. *21st Century Jet – Building the Boeing 777 [Film]*. Skyscraper Productions. 1996 (cit. on p. 23).

[Sal+21] Marc Sallin et al. "Measuring Software Delivery Performance Using the Four Key Metrics of DevOps". In: *Agile processes in software engineering and Extreme Programming*. Ed. by Peggy Gregory et al. Springer International Publishing, 2021, pp. 103–119 (cit. on p. 21).

[Sal15] Johnny Saldaña. *The coding manual for qualitative researchers*. Sage, 2015 (cit. on p. 40).

[San+16] Ronnie Santos et al. "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 (cit. on pp. 31, 44).

[SB20] Mojtaba Shahin and M. Ali 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*. ICSSP '20. ACM, 2020, pp. 175–184 (cit. on pp. 52, 63, 64, 83).

[SB98] Carolyn B. Seaman and Victor R. Basili. "Communication and organization: an empirical study of discussion in inspection meetings". In: *IEEE Transactions on Software Engineering* 24.7 (1998), pp. 559–572 (cit. on p. 26).

[SBZ16] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives". In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '16. ACM, 2016, 44:1–44:10 (cit. on pp. 14, 19).

[Sch+16a] Gerald Schermann et al. "An empirical study on principles and practices of continuous delivery and deployment". In: *PeerJ PrePrints* 4 (2016), e1889 (cit. on p. 1).

[Sch+16b] Gerald Schermann et al. “Towards quality gates in continuous delivery and deployment”. In: *24th IEEE International Conference on Program Comprehension*. ICPC. 2016, pp. 1–4 (cit. on p. 1).

[Sch98] Thomas Schwandt. “Chapter 7: Constructivist, interpretivist approaches to human inquiry”. In: *The landscape of qualitative research: theories and issues*. Sage publications, 1998 (cit. on p. 31).

[Sen+99] Peter Senge et al. *The dance of change: The challenges to sustaining momentum in learning organizations*. Broadway Business, 1999 (cit. on pp. 24, 25).

[SER04] Manuel E. Sosa, Steven D. Eppinger, and Craig M. Rowles. “The Misalignment of Product Architecture and Organizational Structure in Complex Product Development”. In: *Management Science* 50.12 (2004) (cit. on p. 24).

[SF18] Klaas-Jan Stol and Brian Fitzgerald. “The ABC of Software Engineering Research”. In: *ACM Transactions on Software Engineering and Methodology* 27.3 (2018) (cit. on p. 29).

[SF95] James Stoner and R. Edward Freedman. *Administração*. Prentice-Hall, 1995 (cit. on p. 22).

[Sha+17] Mojtaba Shahin et al. “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 (cit. on pp. 2, 26, 27, 61, 63–65).

[Sha+19] Mojtaba Shahin et al. “An empirical study of architecting for continuous delivery and deployment”. In: *Empirical Software Engineering* 24 (2019), pp. 1061–1108 (cit. on pp. 21, 27, 46).

[Sin20] Toby Sinclair. *12 Organisational Design Principles that Embrace Complexity*. Available on https://www.tobysinclair.com/post/organisational-design-principles-that-embrace-complexity, accessed in Mar 2021. 2020 (cit. on p. 24).

[Siq+18] Rodrigo Siqueira et al. “Continuous Delivery: Building Trust in a Large-Scale, Complex Government Organization”. In: *IEEE Software* 35.2 (2018), pp. 38–43 (cit. on pp. 20, 32).

[Sir+11] David G. Sirmon et al. “Resource orchestration to create competitive advantage: Breadth, depth, and life cycle effects”. In: *Journal of management* 37.5 (2011), pp. 1390–1412 (cit. on pp. 8, 39, 83).

[Smi14] Adam Smith. “Chapter 1: Of the division of labour”. In: *The Wealth of Nations*. Originally published in 1776. Createspace, 2014 (cit. on pp. 22, 92).

[Som11] Ian Sommerville. *Software engineering*. 9th. Addison-Wesley, 2011 (cit. on p. 12).

[SP13] Matthew Skelton and Manuel Pais. *DevOps Topologies*. https://web.devopstopologies.com/, accessed in Nov 2021. 2013 (cit. on pp. 20, 26, 61–63).

[SP19] Matthew Skelton and Manuel Pais. *Team Topologies: Organizing business and technology teams for fast flow*. IT Revolution Press, 2019 (cit. on pp. 26, 27, 61, 63).

[SRF16] Klaas-Jan Stol, Paul Ralph, and Brian 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 (cit. on pp. 31, 41, 43).

[Ste+16] Igor Steinmacher et al. “Overcoming Open Source Project Entry Barriers with a Portal for Newcomers”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. ACM, 2016, pp. 273–284 (cit. on p. 39).

[Str19] Per Erik Strandberg. “Ethical Interviews in Software Engineering”. In: *International Symposium on Empirical Software Engineering and Measurement 2019*. ESEM ’19. 2019 (cit. on pp. 43, 44, 87).

[Vel+14] Nicole Forsgren Velasquez et al. *2014 State of DevOps Report*. https://puppet.com/resources/whitepaper/2014-state-devops-report, accessed in Aug 2019. 2014 (cit. on pp. 20, 21, 52).

[Vla55] Gregory Vlastos. “On Heraclitus”. In: *The American Journal of Philology* 76.4 (1955), pp. 337–368 (cit. on p. 87).

[vv13] Guus van Waardenburg and Hans van Vliet. “When agile meets the enterprise”. In: *Information and Software Technology* 55.12 (2013), pp. 2154–2171 (cit. on pp. 40, 70).

[WAL15a] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. “Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications”. In: *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 60–65 (cit. on p. 14).

[WAL15b] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. “Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments”. In: *On the Move to Meaningful Internet Systems*. OTM 2015 Conferences. Springer International Publishing, 2015, pp. 348–358 (cit. on p. 14).

[Wat14] Michael Grant Waterman. “Reconciling agility and architecture: a theory of agile architecture”. PhD thesis. Victoria University of Wellington, 2014 (cit. on pp. 83, 84).

[Wen+20] Melissa Wen et al. “Leading successful government-academia collaborations using FLOSS and agile values”. In: *Journal of Systems and Software* 164 (2020), p. 110548 (cit. on p. 32).

[Win14] Frederick Taylor Winslow. *The Principles of Scientific Management*. Reprint of 1911 Edition. Martino Fine Books, 2014 (cit. on p. 22).

[WNA15] Michael Waterman, James Noble, and George 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 (cit. on pp. 31, 70).

[Woo16] Eoin Woods. “Operational: The Forgotten Architectural View”. In: *IEEE Software* 33.3 (2016), pp. 20–23 (cit. on pp. 13, 16, 17).

[YH98] Dale E. Yeatts and Cloyd Hyten. *High-Performing Self-Managed Work Teams: A Comparison of Theory to Practice*. Sage Publications, 1998 (cit. on pp. 22, 23).

[Yin09] Robert K. Yin. *Case Study Research, Design and Methods*. 4th. Sage Publications, 2009 (cit. on pp. 7, 39, 41, 46, 83, 86, 88).

[YK16] Hasan Yasar and Kiriakos Kontostathis. “Where to Integrate Security Practices on DevOps Platform”. In: *International Journal of Secure Software Engineering (IJSSE)* 7.4 (2016), pp. 39–50 (cit. on p. 14).