Platform Teams: An Organizational Structure for Continuous Delivery

Leonardo Leite, Fabio Kon
University of São Paulo, Brazil
leofl@ime.usp.br,kon@ime.usp.br

Gustavo Pinto
Federal University of Pará, Brazil
gpinto@ufpa.br

Paulo Meirelles
Federal University of São Paulo, Brazil
paulo.meirelles@unifesp.br

ABSTRACT

Software-producing organizations are seeking to release faster and more efficiently new versions of their products to their customers to remain competitive in the fierce software market. Continuous delivery practices arise as a potential solution since every commit to the repository could result in a production-candidate version of a product, accelerating time to market, and improving customer satisfaction. In this work, we employed Grounded Theory to investigate how organizations pursuing continuous delivery should organize their development and operations teams. We collected data from 27 IT professionals. After a careful analysis, we started the elaboration of a taxonomy with four patterns of organizational structures: (1) siloed departments, (2) classical DevOps, (3) cross-functional teams, and (4) platform teams. We observed that the platform team structure is the most distinctive classification of our taxonomy, and it has promising results regarding delivery performance. Some relevant aspects we found out about platform teams include: infrastructure specialists need coding skills; product teams have to operate their business services; and much of the non-functional concerns are handled by the platform, alleviating product teams.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; **Programming teams**; *Software post-development issues.*

KEYWORDS

Continuous Delivery, Release Process, DevOps, Software Teams

ACM Reference Format:
Leonardo Leite, Fabio Kon, Gustavo Pinto, and Paulo Meirelles. 2020. Platform Teams: An Organizational Structure for Continuous Delivery. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3391455

1 INTRODUCTION

Many software organizations are looking for ways to achieve rapid release delivery to get their products and new features to their customers faster and more efficiently. In this context, continuous delivery practices become increasingly important, mostly due to their benefits, such as accelerated time to market [3]. The automation that continuous delivery introduces creates a profound impact on various aspects of the software engineering practice (e.g., development, testing, deployment) [23]; it also impacts the organizational structure [3] since release activities involve many divisions of a company (e.g., developers, operations, and business).

Therefore, organizations moving toward continuous delivery have not only to upgrade their software tooling arsenal but also find ways to better shape and integrate their IT teams. Such integration can occur according to different patterns that we call *organizational structures*. However, there is no substantial literature tackling how organizations should structure their teams to excel in the context of continuous delivery. This lack of research is particularly unfortunate due to at least two crucial reasons: *(i)* organizations wishing to adopt continuous delivery can be disoriented regarding how to design their human resources structure toward this goal; *(ii)* once a structure is chosen, the organization might be unaware of the consequences of this choice. The existing literature presents some classifications for organizational structures [15, 18, 25]. Although some of these works are empirical studies, the presented options of structures are an arbitrary start point for them, lacking empirical elaboration.

To mitigate this gap, our research efforts employ empirical methods to answer our main research question *"which organizational structures are software-producing organizations adopting for managing IT technical teams in a continuous delivery context?"* In this short paper, containing preliminary results, we partially answer our research question by presenting the most distinctive of these structures.

By employing Grounded Theory [7] and interviewing 27 IT professionals, we discovered four organizational structures: *(i)* traditional *siloed departments*, with high impedance for cooperation among development and operations; *(ii) classical DevOps*, focusing on the communication and collaboration among development and operations; *(iii) cross-functional teams*, presenting all the required roles within a single team; and *(iv) platform teams*, exposing highly-automated infrastructure services to assist developers.

The three first structures could already be somehow expected according to the current DevOps literature [12] and models publicly adopted by tech giants, such as Google [2] and Amazon [8]. In this sense, the most significant impact of our taxonomy for practice is highlighting the *platform team* as a distinctive organizational choice. Thus, **in this paper, we focus on describing the platform team structure**, with its practical implications, such as division of responsibilities, communication, bottlenecks, monitoring, and incident handling. Some relevant aspects we found out about platform teams are:

- Infrastructure specialists need coding skills;

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391455

- Product teams have to operate their business services;
- Many non-functional concerns are handled by the platform, alleviating product teams;
- The platform acts as a façade for public clouds or internal infrastructure.

Another reason for drawing attention to platform teams is that the collected data indicate that organizations embracing this model achieve better **delivery performance** results.

2 DELIVERY PERFORMANCE

Delivery performance is a combination of three metrics: frequency of deployment, time from commit to production, and mean time to recovery [5]. Delivery performance also correlates to the organizational capability of achieving higher-level (commercial and noncommercial) goals [5]. We used this construct in our research as an indication of how successful the organization has been in adopting continuous delivery. We, therefore, asked each participant in our study about the frequency of deployment, the time from commit to production, and the mean time to recovery to define the delivery performance in the interviewee's context.

Based on a survey with 27,000 responses in 2017, Forsgren *et al.* [5] applied cluster analysis to these metrics and discovered three groups: *High performers* were characterized as having multiple deployments per day, commits taking less than 1 hour to reach production, and incidents repaired in less than 1 hour. *Medium performers* deployed between once per week and 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 taking between one day and one week to repair incidents.

In our research, we are not interested in distinguishing medium from lower performers; we are interested only in identifying high performers and non-high performers. However, the above clusters are problematic for our purpose since there is a gap between the high and the medium performers clusters. We circumvent 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 at most one high-performance threshold by only one point in the scale adopted for the metric. The scales for each metric are:

Frequency of deployment: multiple deploys per day; between once per day and once per week; between once per week and once per month; between once per month and once every six months; fewer than once every six months.

Time from commit to production: less than one hour; less than one day; between one day and one week; between one week and one month; between one month and six months; more than six months.

Mean time to recovery: less than one hour; less than one day; between one day and one week; between one week and one month; between one month and six months; more than six months.

Now we proceed, in Sections 3 and 4, to present our empirical approach. The emerging taxonomy is briefly presented in Section 5, while Section 6 describes in details the platform team structure. We also present limitations of this work (Section 7), related works (Section 8), and our conclusion (Section 9).

3 GROUNDED THEORY

Our research aims to generate a theory on organizational structures in the context of continuous delivery. For this purpose, we employed Grounded Theory (GT), a widely-used qualitative research approach in software engineering [9, 21, 26, 28], whose focus is on generating a theory rather than validating preconceived hypotheses. Since there are multiple GT variants available, Stol *et al.* [26] advise researchers to state which variant they adopt. In this paper, we based our research approach on the seminal book "The discovery of Grounded Theory" [7], which describes what is known as the "classical Grounded Theory" [26].

The *constant comparative method* is the core of GT, relying on rigorous analysis of qualitative data. *Coding* is the process of condensing original data in a few words with conceptual relevance. The code is then compared to similar codes of the previous analysis, giving emergence to theoretical concepts [7].

Besides the rigorous analysis of qualitative data, GT also relies on the researcher's *theoretical sensitivity*, the ability to have theoretical insight into a substantive area. Our theoretical sensitivity comes from direct experience in the IT industry and our previous works on DevOps and software engineering [4, 12, 14, 24].

In GT, data collection and analysis are interspersed, so the emerging theory guides which data to sample next, considering gaps and questions suggested by previous analysis. This process—called *theoretical sampling*—does not consider usual statistical notions of verification methods, such as significant samples. Instead, researchers must establish the theoretical purpose of the sample, defining multiple comparison groups, maximizing variation among groups to find out similarities and minimizing variation to find out differences. We approached theoretical sampling mainly by valuing the diversity of people and organizations in our sample. Another application of theoretical sampling we did was interviewing a company which we expected to have cross-functional teams since we had only a few interviewees presenting this structure.

Ideally, the researcher carries the analysis until *theoretical saturation* is achieved, which means that new data does not meaningly impact the theory elaborated until that moment. Since we are still conducting more interviews to refine our theory further, we do not claim to have already reached theoretical saturation. Nonetheless, the interviews conducted so-far were enough to discover and characterize the platform teams, which is the subject of this paper.

4 INTERVIEWS

We applied the GT techniques on data acquired from interviews with IT professionals. In this section, we present the processes related to interview selection, conduction, and analysis.

Brainstorm sessions

We conducted what we called "brainstorm sessions" with seven specialists based on our research question and concerns raised by our survey on the DevOps literature [12]. We chose these experts for their experience with DevOps and proximity to our research group. These sessions helped to better shape the interview script,

targeting concerns learned from these experts. After these brainstorm sessions, we started the semi-structured interviews.

Selecting participants

We sent 45 interview invitations using a convenience approach: the first invitations were contacts close to the network of our research group. We also contacted other participants by indication of our interviewees. The only requirement was that the participant should work with continuous delivery or at least be implementing efforts toward it. From the 45 invited professionals, we could interview 20 of them. Following ethical procedures [27], we anonymized all the interviewees and their organizations.

We employed several strategies to foster diversity and to enhance comparison possibilities in our sample. First, we interviewed participants that worked in scenarios where it was particularly challenging to achieve continuous delivery (e.g., IoT or defense systems). Second, we tried to choose a broad range of organization and interviewee profiles. For instance, we selected both small and large companies (35% with more than 1,000 employees), private and governmental companies, from different domains and countries. We covered men and women (35% of women) and also chose interviewees with varying roles, such as developers, managers, infrastructure specialists, and even a designer.

Table 1 shows the description of the participants (the numbers on the left refer to the number of interviews). As one can see, this table only presents an aggregated profile of participants. We took this decision to hinder de-anonymization [21, 27]. Location refers to where the interviewee's team is located; we had a few participants working remotely for globally distributed teams. Enabler team is a specialized technical team that supports developers but does not own any service. We could not know which companies contracted the consultants' services; therefore, our demographic evaluation considers the company that employs the interviewee. The interviewees worked in the following business domains: IoT, finances, defense, public administration, justice, real estate, education, Internet, big data, research, cloud, games, mobility, office automation, software consulting, and support to software development.

Table 1: Description of participants and their organizations.

	Role		Degree
#9	Developer	#11	Undergraduate
#3	Development manager	#9	Masters
#2	Infrastructure manager		**Location**
#2	External consultant	#11	Brazil
#1	Infrastructure engineer	#4	USA
#1	Executive manager	#3	Globally distributed
#1	Enabler team member	#1	Germany
#1	Designer	#1	France
	Gender		**Organization size**
#13	Man	#9	From 200 to 1000 employees
#7	Woman	#7	More than 1000 employees
	Time since graduation	#4	Less than 200 employees
#11	More than 10 years		**Organization type**
#5	From 5 to 10 years	#17	Private for profit
#4	Less than 5 years	#2	Governmental
#1	Private nonprofit		

Conducting semi-structured interviews

Our goal is to discover existing organizational structures and not verifying in the field a preconceived set of structures. In this way, we conducted semi-structured interviews [1], avoiding using only closed questions. The set of questions that we used to guide the interviews are available online[1].

Before starting the interviews, we built an interview protocol to guide the process based on our previous experience with interviews [4], on other relevant works [1, 9], and on the tips offered by ijnet[2], a website for journalists. The interview protocol also contains the questions that drove the interviews, which were derived mainly from the brainstorm sessions and our survey of the DevOps literature [12]. The themes addressed by the interview questions were the following: *(i)* interviewee company and role; *(ii)* responsibility for deployment, building new environments, non-functional requirements, configuring and tracking monitoring, and incident handling, especially after-hours; *(iii)* which cloud vendor is used; *(iv)* whether microservices are used or not; *(v)* delivery performance; *(vi)* future improvements in the organization; *(vii)* effectiveness of inter-team communication; *(viii)* inter-team alignment for the success of the projects; *(ix)* description of DevOps team or DevOps role, if existing; *(x)* the existence and sharing policy for specialized roles, like security or database; and finally *(xi)* practices about knowledge acquisition and sharing.

Analyzing the interviews

We followed the core Grounded Theory principles of *constant comparative method* and *coding*, which are intended to discipline the creative generation of the theory. During this process, we created two artifacts for each interview: the **transcripts** and the **codes**.

Transcripts. We heard each audio record and transcribed it. We did not transcribe the full interview. Instead, we synthesized relevant parts of the interviews, excluding minor details and meaningless noise [6], while preserving some interesting excerpts. For instance, from one interview, we transcribed the following part of the conversation:

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

Codes. After transcribing, we then derived the coding by condensing the transcripts in a few words. The above fragment of transcription, for example, led to the following coding:

> *Developers → owns the availability of their services*
> *Broken SLA → blocks feature development*
> *Broken SLA → page for developers*
> *Broken SLA → if needed, calls infra*

Finally, by analyzing, comparing, and using all the coding, we started to elaborate our **taxonomy**, the theory itself. Coding is carried by one researcher and reviewed by the other ones, while the taxonomy elaboration is paired by two researchers and reviewed by the other ones.

[1] http://ccsl.ime.usp.br/devops/2020-03-11/interview-questions.html
[2] ijnet.org/en

5 THE TAXONOMY FOR ORGANIZATIONAL STRUCTURES

Our grounded taxonomy for organizational structures has four key elements: *(i)* siloed departments, *(ii)* classical DevOps, *(iii)* cross-functional teams, and *(iv)* platform teams. In this section, we briefly describe the three first structures, while we shall describe platform teams in more detail in the next section. Table 2 indicates the found organizational structures and achieved delivery performance in our interviews. As we discuss in Section 6.1, this table indicates that all the interviewed organizations having consolidated platform teams also have a high delivery performance. In this paper, interviews are signaled by a token in the format "#I*N*"; thus, "#I2", for example, refers to the second interview or interviewee; brainstorm sessions, those conducted before the interviews, are indicated as "#B*N*".

Table 2: Observed organizational structures and their associated delivery performance

Organizational structure	Delivery performance	Interviews	Number of interviews
Siloed departments	Not-high	#I5 #I7 #I10 #I13 #I15 #I19	6
Siloed departments (adopting a platform)	High	#I20	1
Siloed departments (adopting a platform)	Not-high	#I8	1
Classical DevOps	High	#I2 #I17	2
Classical DevOps	Not-high	#I6 #I11 #I18	3
Classical DevOps (adopting a platform)	Not-high	#I14	1
Cross-functional	High	#I1	1
Cross-functional	Not-high	#I3	1
Cross-functional (adopting a platform)	Not-high	#I16	1
Platform team	**High**	**#I4 #I9 #I12**	**3**

5.1 Siloed departments

With siloed departments, developers and infrastructure specialists are segregated from each other; there is little face-to-face communication among these groups, and blaming each other for failures is commonplace. In our interviews, we found eight organizations adhering to this organizational structure.

Characteristics. Developers and operators have well-defined and different roles. Developers have a minimal vision of what happens in production; monitoring and handling incidents are mostly done by the infrastructure team. Developers often neglect non-functional requirements (NFR); security can be seen as an infrastructure concern only. DevOps initiatives are centered on adopting continuous integration tools rather than improving collaboration among silos. As a consequence, communication and collaboration among teams are hard.

5.2 Classical DevOps

The classical DevOps structure follows the initial intents of the DevOps movement by focusing on collaboration among developers and the infrastructure team; it approximates people and breaks down the walls, as it is so vividly devised in the novel "The Phoenix Project" [11]. We consider that the SRE model from Google [2] is a Classical DevOps structure: there are developers from one side and SREs (infrastructure specialists) on the other side, but they collaborate well for the project success. We found six organizations adhering to this organizational structure.

Characteristics. Roles remain well-defined, although developers and operators are closer (e.g., for database management, infrastructure staff creates and tunes the database, whereas developers write queries and manage the schema), which fosters a culture of collaboration. Usually, there are no conflicts regarding who is responsible for each task. DevOps is achieved through a delivery pipeline. NFR responsibilities are shared among developers and the infrastructure team. However, the infrastructure staff is still on the front line of tracking monitoring and incident handling.

5.3 Cross-functional teams

This structure is aligned with the Amazon motto "*You built it, you run it*" [8]. This gives more freedom to the team, along with a great deal of responsibility. We found three organizations adhering to this organizational structure.

Characteristics. A single team encompasses both developers and infrastructure specialists to take total responsibility for the life cycle of a set of services. This structure is the one that most supports communication and collaboration among people with different skills. Everyone in the team can be assigned to incident handling. The challenge here is to guarantee that each unit has all the necessary skills.

6 PLATFORM TEAMS

Platform teams are infrastructure teams that provide highly automated infrastructure services that can be self-serviced by developers for the deployment of new services, usually microservices. In this organizational structure, the infrastructure team is no more a "support team"; it behaves like a product team, having the "platform" as its product and developers as internal customers. Figure 1 conceptually illustrates this structure, with rectangles representing concepts and rounded boxes representing conceptual properties.

Throughout our semi-structured interviews, we found three organizations fully embracing this model. The other four organizations are in the process of adopting the platform model. Nevertheless, we could observe most of the platform team characteristics in these transitioning organizations. Although we did not classify the organizational structures of the brainstorm sessions, we also perceived the platform team pattern in three of these sessions. In total, from the 27 professions we talked to, we discussed this structure with ten of them (two of them working in the same company). Five of these organizations have more than 1,000 employees and four of them between 200 and 1,000 employees.

We observed the following practical implications (PI) for organizations having a platform team:

PI#1) *Product team are fully accountable for the non-functional requirements of its services.* The product team becomes now the

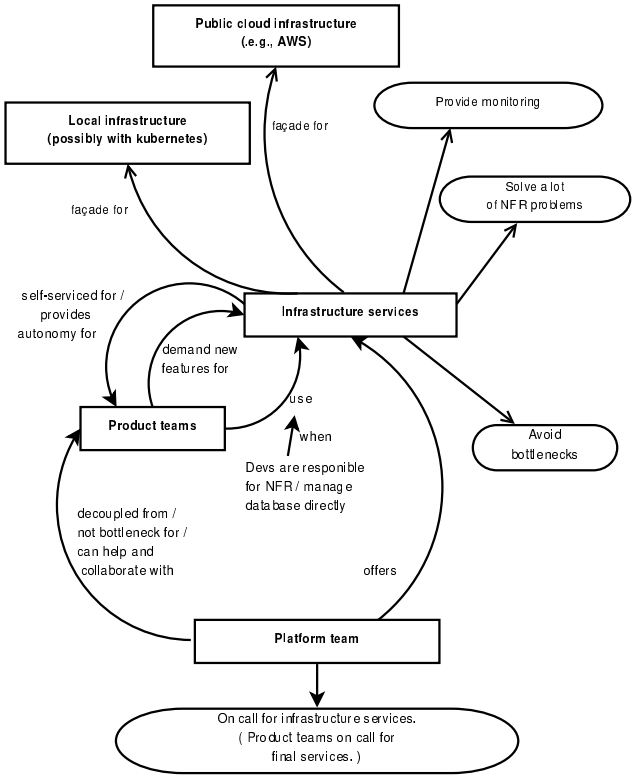

Figure 1: Conceptual illustration of platform teams

first one to be called when there is an incident; the infrastructure people are escalated if the problem is related to some infrastructure service. This situation differs from the Classical DevOps scenario, in which usually the infrastructure staff is the first one to take care of any incident, summoning developers only if needed.

PI#2) *Developers are not afraid of NFR concerns.* Although the product team becomes fully responsible for NFRs of its services, it is not a significant burden that developers try to refuse. It happens since the platform itself handles many NFR concerns, such as load balancing and auto-scaling. Moreover, we saw cases that, despite the decoupling among developers and infrastructure people provided by the platform, the infrastructure people are very supportive in coping with developers for the sake of services availability and performance.

PI#3) *Product teams become decoupled from the members of the platform team.* Usually, the communication among these teams happens when developers and infrastructure people are gathered to solve incidents, as we saw in #I9; when infrastructure people provide consulting for developers for mastering non-functional concerns, as observed in #I9; or when developers demand new capabilities from the platform, as noticed in #I12. In this way, the decoupling between the platform and product teams does not imply the absence of collaboration among these groups.

PI#4) *The infrastructure team is no more requested for operational tasks.* Therefore, we cannot merely call platform-team members as "operators" since they also engineer the infrastructure solution. We remark that, in other industries, "operator" is a title attributed to menial workers. This characteristic also matches the SRE philosophy of reducing toil work.

PI#5) *In general, product teams do not need to encompass infrastructure specialists.* The existence of a delivery platform avoids the need for product teams having infrastructure specialists, as it would be the case for cross-functional teams. On the other hand, since developers are responsible for the deployment, it requires developers to have some basic knowledge about the infrastructure and mainly the platform itself, differently from a siloed structure.

PI#6) *However, sometimes there are infrastructure operators within the product team.* When using a platform, the complexity of a specific application might require the product team to behave like a fully cross-functional team, having infrastructure operators (#I16).

PI#7) *Infrastructure specialists possess coding skills.* 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", so it can produce scripts and use infrastructure-as-code [16] to automate the delivery path. One strategy we saw to cope with this need in #I14 was to hire previous developers for the infrastructure team.

The platform team structure contrasts with the other ones for the following reasons: different from cross-functional teams, there is an infrastructure team; different from classical DevOps, developers consume infrastructure as a service and operate their services; and different from siloed departments, platform teams cultivate an open dialogue with developers.

6.1 Discussion

We observed, as summarized by Table 2, organizations with platform teams presenting the best outcomes in terms of delivery performance. All the three organizations that have fully embraced the platform team structure are high performers, while no other structure provided such property. In particular, the only high-performer siloed organization is adopting a platform structure.

An explanation for the relationship among the platform team structure and delivery performance is that this structure decouples the infrastructure and the product teams. This decoupling prevents the infrastructure team from becoming a bottleneck in the delivery path; at the same time, the platform provides the necessary power to developers to operate their services without the need for further specialization in infrastructure. As stated by #I20: *"Developers can't hide behind this anymore. Now they have autonomy for going from zero to production without having to wait for anyone."* The platform team structure also contributes to increase service reliability by placing the product team in the front-line of handling 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."* #I16 provided some examples of issues handled by the platform,

such as throttling and high-speed communications between datacenters. In this way, the platform contributes to developers being autonomous and accountable for managing their services[3].

There is a recent trend in discussing NoOps, which some consider being a structure in which a dedicated operations team is not needed anymore, given the current high degree of infrastructure automation [20]. We consider that adopting platform teams moves in this direction. If each team operates its own services, whether business or infrastructure services, then there is no such thing as operators supporting a service built by someone else.

In some cases, we observed organizations ultimately deploying applications on public clouds, such as Amazon WS or Google Cloud. 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 in such a way that developers do not need to understand so many details about the cloud. Interviewee #I16 mentioned that *"the [platform] value is adding services that are actually enhancing the usability of the [cloud] infrastructure."* Thus, in such scenarios, we state that the platform can be seen as a "**cloud façade**." As an example, Nubank has publicly talked in events about having a platform team (or platform squad, as it is called there) operating over a public cloud [4].

As reported by #B1, #I1, #I8, #I12, and #I20, there are also platforms built on top of internal physical servers, hiding from developers such infrastructure complexities as the use and even the existence of Kubernetes, an open-source platform used for managing the lifecycle of Docker containers. An example of a publicly known platform like this is Estaleiro[5] from Serpro (the largest Latin American public software-producing organization). However, one should note that these platforms cannot be seen as silver bullets. For instance, #B1 reported that not every project he worked on could be deployed in the Kubernetes platform, given security restrictions associated with specialized hardware and network isolation.

We also observed a small company, in #I14, with a "platform mindset", but without the resources to build a platform of its own. In this case, we saw the usage of an open-source platform, Rancher[6], and the infrastructure team preparing Terraform[7] templates encompassing good practices, so developers would need only to provide a few data fields for deployment. Rancher, a graphical façade for developers to interact with Kubernetes, was also adopted in #I20.

Although the organization of #I20 did an excellent job in transitioning to a platform structure and achieving high-delivery performance, according to the interviewee, the *"old world"* still coexists with the *"new one."* In the same way, as reported by Nybom *et al.* [18], there are some responsibility conflicts and *"dissident forces:"* some operations people do not like developers with administrative powers, while some developers do not want such powers. The interviewee declared that *"it's not yet 'everybody together'; some people are still worried about covering their sides."* This example shows how, despite all the benefits earned, transitioning structures in large organizations is a hard endeavor. It also shows how technical solutions do not dismiss a culture of collaboration.

[3] #I9 referred to whoownsmyavailability.com.
[4] https://www.youtube.com/watch?v=iZ1taqc5-G8,
https://www.youtube.com/watch?v=gqXzvVJXfoY
[5] https://pt.slideshare.net/rikatz/estaleiro-o-uso-de-kubernetes-no-serpro
[6] http://rancher.com
[7] http://terraform.io

7 LIMITATIONS

This work has many limitations. First, we cannot claim full generalizability. Our work is limited by the number of interviews, and the context of the participants. We mitigate this limitation by choosing participants working on different companies and with different levels of skills. Moreover, our method, Grounded Theory, does not guarantee that two researchers working in parallel with the same data would achieve identical results [7]. Still, this method was applied by only one researcher (although revised by another research).

Regarding our definition of high delivery performance, although we relied on previous work [5], we needed to adapt the original definitions due to the reasons exposed in Section 1. Therefore, by tweaking the descriptions related to delivery performance, one could reach different conclusions based on the same data we have. Another issue with delivery performance is that although a high performance is a good indicator of a well functioning software-producing organization, not achieving high performance cannot necessarily be seen as a bad indicator, since organizations can have its reasons to limit the deployment frequency, including business decisions.

We are aware that large companies usually present groups at different maturity levels, and that they could be classified differently if we took different interviewees. To verify this, we interviewed two persons from the same company (#I16 and #I18). Indeed, the organizational patterns were not identical. Therefore, the reader must note that our descriptions do not characterize the whole organizations, but the contexts observed by our respondents. In #8, for example, the interviewee works in an enabler team, and the reported delivery performance is the typical case he is used to observe despite the existing variability within the organization.

8 RELATED WORK

There is a recent research track discussing the usage of continuous delivery practices [3, 13, 17, 19, 22]. Benefits perceived by these works include accelerated time to market, improved productivity and efficiency, faster feedback to development, enhanced customer satisfaction, and a more predictable defect rate [3, 13, 17]. On the other hand, some challenges are coordination of supplier integration, business models, and difficulty in overviewing project status [19]. Chan *et al.* state that, in the adoption of continuous delivery, organizations may also have to deal with challenges that are not only technical but also organizational and process-related [13]. However, it is so far unclear what constitutes a functional characterization of development and operations teams that excel in continuous delivery practices.

The literature about the classifications of organizational structures for managing IT technical teams in continuous delivery or DevOps contexts is limited [10, 15, 18, 25]. In all these related works, the presented sets of structures are arbitrary, without considerations about how they were conceived. Distinct from them, we found the organizational structures through a systematic research process based on field observations. Nybom *et al.* present three distinct

scenarios to DevOps adoption [18]: (i) mixing development and operations responsibilities among all engineers; (ii) mixing personnel, but keeping existing roles differentiated; and (iii) creating a DevOps team as a bridge between development and operations. None of them seems to indicate a platform team.

The third implicit organization scheme presented by Humble and Molesky envisions an operations group acting as a product team offering support services (continuous integration, monitoring services, etc.) to the entire organization [10]. This structure somehow resembles the platform team structure, but in a more limited way, without considering a highly automated way for product teams creating new services.

The 2018 State of DevOps Report surveys respondents about the organizational structures used in their DevOps journeys, offering a closed set of alternatives to answer the question [15]. However, the text does not fully explain the options; it only presents the names of the proposed organizational structures. In this way, associating the platform team to one of the structures presented by the survey would be a too error-prone activity.

Finally, Skelton and Pais present "DevOps topologies" and DevOps anti-patterns [25], being the most informal of our comparison sources – a blog post. However, the DevOps topology Type 3 seems to fit our platform team definition. This topology is named "Ops as Infrastructure-as-a-Service (Platform)" and presents "operations as a team who simply provides the elastic infrastructure on which applications are deployed and run; the internal Ops team is thus directly equivalent to Amazon EC2, or Infrastructure-as-a-Service." However, the presentation is too short, not presenting further details about how organizations are applying this topology. Moreover, from an academic perspective, it is also not desirable the opinion-based judgment that the authors do to separate good DevOps topologies from DevOps anti-patterns. We also note that the term "platform" emerged from our interviewees.

9 CONCLUSION

Continuous delivery is a practice that helps developers accelerate project time to market since every commit could potentially release a new version of the software. However, it is not clear how software teams should be organized to take real advantage of this practice (e.g., should developers assume infrastructure roles?). We interviewed 27 developers that are facing or transitioning to continuous delivery. After understanding the organization of teams, we introduced the concept of "platform teams." In this structure, according to our observations, an infrastructure team delivers highly automated infrastructure services that are used by product teams. Such infrastructure services decrease the necessity for product teams to include infrastructure specialists and promote high delivery performance.

ACKNOWLEDGMENTS

This research was supported by CNPq (proc. 465446/2014-0, 309032/2019-9, and 406308/2016-0), FAPESP proc. 15/24485-9, and FAPESPA.

REFERENCES

[1] William C. Adams. 2010. Conducting semi-structured interviews. In *Handbook of Practical Program Evaluation* (3rd ed.). Jossey-Bass.
[2] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media.
[3] Lianping Chen. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software* 32, 2 (2015), 50–54.
[4] Daniel Cukier and Fabio Kon. 2018. A maturity model for software startup ecosystems. *Journal of Innovation and Entrepreneurship* 7, Article 14 (2018).
[5] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. Measuring Performance. In *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution Press.
[6] Svetla Georgieva and George Allan. 2008. Best Practices in Project Management Through a Grounded Theory Lens. *The Electronic Journal of Business Research Methods* 6, 1 (2008), 43–52.
[7] Barney Glaser and Anselm Strauss. 1999. *The discovery of grounded theory: strategies for qualitative research.* Aldine Transaction.
[8] Jim Gray. 2006. A conversation with Werner Vogels. *ACM Queue* 4, 4 (2006), 14–22.
[9] Rashina Hoda and James Noble. 2017. Becoming Agile: A Grounded Theory of Agile Transitions in Practice. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE '17).* 141–151.
[10] Jez Humble and Joanne Molesky. 2011. Why enterprises must adopt DevOps to enable continuous delivery. *Cutter IT Journal* 24, 8 (2011), 6.
[11] Gene Kim, Kevin Behr, and George Spafford. 2018. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win* (3 ed.). IT Revolution Press.
[12] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *Comput. Surveys* 52, 6 (2019), 127:1–127:35.
[13] Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V. Mäntylä, and Tomi Männistö. 2015. The highways and country roads to continuous deployment. *IEEE Software* 32, 2 (2015), 64–72.
[14] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. 2018. Building a collaborative culture: a grounded theory of well succeeded DevOps adoption in practice. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018).* 6:1–6:10.
[15] Andi Mann, Michael Stahnke Alanna Brown, and Nigel Kersten. 2018. 2018 State of DevOps Report. https://puppet.com/resources/whitepaper/2018-state-of-devops-report, accessed on Jul 2019.
[16] Kief Morris. 2016. *Infrastructure as Code: Managing Servers in the Cloud.* O'Reilly Media.
[17] Steve Neely and Steve Stolt. 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *2013 Agile Conference.* 121–128.
[18] Kristian Nybom, Jens Smeds, and Ivan Porres. 2016. On the Impact of Mixing Responsibilities Between Devs and Ops. In *International Conference on Agile Software Development (XP 2016).* Springer International Publishing, 131–143.
[19] Helena H. Olsson, Hiva Alahyari, and Jan Bosch. 2012. 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.* 392–399.
[20] Margaret Rouse. 2015. NoOps Definition. https://searchcloudapplications.techtarget.com/definition/noops, accessed on August 2019.
[21] Ronnie Santos, Fabio Silva, Cleyton Magalhaes, and Cleviton Monteiro. 2016. 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).* 971–981.
[22] Gerald Schermann, Jürgen Cito, Philipp Leitner, and Harald C. Gall. 2016. Towards quality gates in continuous delivery and deployment. In *24th IEEE International Conference on Program Comprehension (ICPC).* 1–4.
[23] Gerald Schermann, Jürgen Cito, Philipp Leitner, Uwe Zdun, and Harald C. Gall. 2016. An empirical study on principles and practices of continuous delivery and deployment. *PeerJ PrePrints* 4 (2016), e1889.
[24] Rodrigo Siqueira, Diego Camarinha, Melissa Wen, Paulo Meirelles, and Fabio Kon. 2018. Continuous Delivery: Building Trust in a Large-Scale, Complex Government Organization. *IEEE Software* 35, 2 (2018), 38–43.
[25] Matthew Skelton and Manuel Pais. 2013. DevOps Topologies. https://web.devopstopologies.com/, accessed on Jul 2019.
[26] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE '16).* 120–131.
[27] Per Erik Strandberg. 2019. Ethical Interviews in Software Engineering. In *International Symposium on Empirical Software Engineering and Measurement 2019 (ESEM '19).*
[28] Michael Waterman, James Noble, and George Allan. 2015. How Much Up-front?: A Grounded Theory of Agile Architecture. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE '15).* 347–357.