Building a Theory of Software Teams Organization in a Continuous Delivery Context

Preprint · May 2020

DOI: 10.1145/3377812.3390807

4 authors:

Leonardo Alexandre Ferreira Leite
University of São Paulo
20 PUBLICATIONS **202** CITATIONS
SEE PROFILE

Fabio Kon
University of São Paulo
310 PUBLICATIONS **4,150** CITATIONS
SEE PROFILE

Gustavo Pinto
Federal University of Pará
116 PUBLICATIONS **1,473** CITATIONS
SEE PROFILE

Paulo Meirelles
Universidade Federal do ABC (UFABC)
52 PUBLICATIONS **416** CITATIONS
SEE PROFILE

Building a Theory of Software Teams Organization in a Continuous Delivery Context

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

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

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

ABSTRACT

Based on Grounded Theory guidelines, we interviewed 27 IT professionals to investigate how organizations pursuing continuous delivery should organize their development and operations teams. In this paper, we present the discovered organizational structures: (1) siloed departments, (2) classical DevOps, (3) cross-functional teams, and (4) platform 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. Building a Theory of Software Teams Organization in a Continuous Delivery Context. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377812.3390807

1 INTRODUCTION

To remain competitive in the software market, many software organizations are looking for ways to speed up their release processes to get their products and new features to their customers faster and more efficiently. In this way, continuous delivery automation impacts various aspects of software production (e.g., development, testing, deployment). With an automated deployment pipeline, one could question the role of an engineer responsible solely for new deployments. Indeed, the adoption of continuous delivery has also an impact on the organizational structure [1], since release activities involve many divisions of a company (e.g., developers, operations, and business). The lack of clear roles and obligations may incur into lengthy negotiations and stress between IT departments [6].

Therefore, organizations moving toward continuous delivery have not only to upgrade their software tooling arsenal but also 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: (1) organizations wishing to adopt continuous delivery can be disoriented regarding how to design their human resources structure toward this goal; (2) given a chosen structure, the organization might be unaware of the consequences of this choice. The existing literature presents some classifications for organizational structures [5, 6, 8], but such designations are arbitrary, without an empirical approach to establish the existence of these structures.

To mitigate this gap, this paper addresses the following **research question**: *Which organizational structures are software-producing organizations adopting for managing IT technical teams in a continuous delivery context?*

2 APPROACH

For tackling our research question, we are building a theory in the taxonomy form. Taxonomies, common on software engineering research, are classification systems that group similar instances to increase the cognitive efficiency of its users by enabling them to reason about classes instead of individual instances [7]. If the taxonomy provides explanation, it can be considered a *theory for understanding*, a system of ideas for making sense of what exists or is happening in a domain [7].

In this work we applied Grounded Theory [2], a well-suited methodology for generating taxonomies [7] and a widely-used research approach in software engineering [3, 4, 9], to discover the existing organizational structures in the field. Grounded Theory is adequate for our purposes since it is suitable for questions like "what's going on here?" – we want to know what is going on software-producing organizations – and useful to construct a relevant conceptual and theoretical foundation for the field [9].

We conducted interviews with objective questions about the working day-to-day of participants to find out the underlying organizational structures. Initially, we had brainstorming conversations with seven specialists, who helped us to better understand the relevance of the problem and to shape the questions to be asked in follow-up interviews. We then conducted 20 semi-structured interviews with IT professionals. The only requirement was that the company should have a continuous delivery process or at least an initiative toward it.

We employed several strategies to foster diversity and to enhance comparison possibilities in our sample. We choose a broad range of organization and interviewee profiles: different company sizes (7 companies with more than 1,000 employees), domains, countries (Brazil, USA, Germany, and France), genders (7 women), experience levels (11 interviewees graduated more than ten years ago), and roles (developers, managers, infrastructure specialists, and even a

ICSE '20 Companion, October 5–11, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3390807

designer); besides including two public and one private nonprofit organizations. We also conducted the selection process with theoretical purposes in mind, thus applying theoretical sampling (e.g., selecting scenarios where it is challenging to achieve continuous delivery).

For analysis, we applied the *constant comparative method* and *coding*, which are intended to discipline the creative generation of theory. During this process, we created two artifacts for each interview: the **transcripts** (excluding minor details and meaningless noise) and the **codes** (condensing transcripts in a few words). We also created two global artifacts: the **comparison sheet** (interviewees in lines and properties in columns) and the **conceptual framework** (a diagram with discovered concepts and conceptual properties). Finally, by analyzing, comparing, and using all these artifacts, we are elaborating our **taxonomy**, the theory itself. The artifacts construction 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.

3 RESULTS

We found four organizational structures:

i) We classified eight interviewees contexts as traditional **siloed departments**, with limited collaboration among departments and barriers for continuous deployment. Some typical 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.

ii) We classified six interviewees contexts as having a **classical DevOps** structure, with intense collaboration among developers and the infrastructure team. Some typical 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; developers and the infrastructure team shares NFR responsibilities; the infrastructure staff is still in the front line of tracking monitoring and incident handling; success of classical DevOps requires strong alignment among departments.

iii) We classified three interviewees contexts as possessing **cross-functional teams**, with self-sufficient teams having both development and operations skills. Some typical characteristics are: 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.

iv) We classified three interviewees contexts as presenting **platform teams**, with the infrastructure team providing highly-automated infrastructure services to empower product teams. Some typical characteristics are: the existence of a delivery platform minimizes the need for product teams having infrastructure specialists; product teams become decoupled from the members of the platform team; usually, the communication among the development team and the platform team happens when infrastructure members provide consulting for developers; the product team is the first one to be called when there is an incident; the infrastructure people are escalated if the problem is related to some infrastructure service; although the product team becomes fully responsible for NFRs of its services, it is not a significant burden, since the platform abstracts away the underlying infrastructure and handles several NFR concerns.

4 CONCLUSION

Our work presents an initial taxonomy of organizational structures, based on recent observations from the field employing a well-accepted methodology. In particular, our proposed taxonomy points to the benefits of *i)* helping practitioners to differentiate *classical DevOps* from *cross-functional teams*, which were traditionally blended under the term DevOps, and *ii)* highlighting the *platform team* as a distinctive choice for organizations. We expect the under-development taxonomy to be applicable to empowering practitioners to discuss the current situation of organizations, supporting decisions on structural changes; another application would be supporting, for example, engineers in job interviews to evaluate the suitability of working for a given company.

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] Lianping Chen. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software* 32, 2 (2015), 50–54.
[2] Barney Glaser and Anselm Strauss. 1999. *The discovery of grounded theory: strategies for qualitative research.* Aldine Transaction.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] Paul Ralph. 2019. Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering. *IEEE Transactions on Software Engineering* 45, 7 (2019), 712–735.
[8] Matthew Skelton and Manuel Pais. 2013. DevOps Topologies. https://web.devopstopologies.com/, accessed on Jul 2019.
[9] 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.